ETRAX FS

Designer’s Reference

Axis Communications AB

21st August 2007

While every care has been taken in the preparation of this manual, Axis Communications AB cannot be held responsible for any technical or typographical errors, and reserves the right to make changes to this manual and to the product without prior notice. If you do detect any inaccuracies or omissions, please inform us at:

E-mail: technology@axis.com

WWW: http://developer.axis.com

Axis Communications AB Emdalavägen 14 S-223 69 Lund, Sweden

Phone: +46 46 272 18 00

Fax: +46 46 13 61 30

Contents

[0] Auto generated by axw2tex on the 21st August 2007.

Chapter 1

Introduction

1.1 Functional Block Diagram

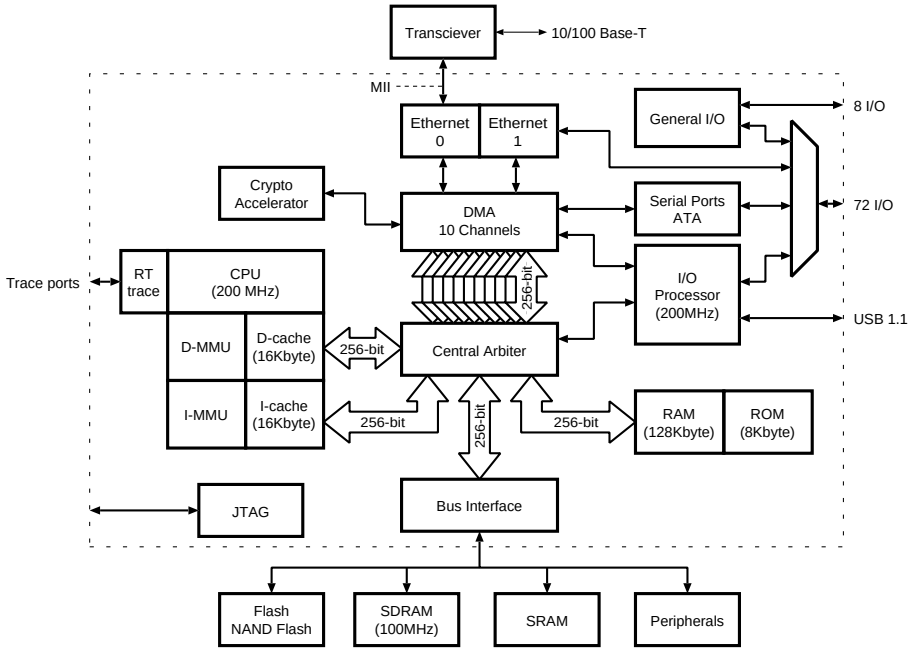


Figure 1.1: *ETRAX FS interface*

1.2 Overview of the AXIS ETRAX FS

Designed for Embedded Linux Powerful 200 MHz CPU with MMU for real Linux support.

High Performance with Low Power Consumption The 200 MHz 32-bit RISC ISA enables compact code and exceptional price/performance with low power consumption. On-chip 16 KB I-cache and 16 KB D-cache, and 128 KB on-chip RAM take full advantage of the CPU performance.

I/O Protocol Processor for Flexible Device Attachment Patented micro-code programmable I/O processor consisting of three 200 MHz 32-bit processors with local memory and hardware accelerators for real-time performance. The I/0 processor is capable of running at least two I/O protocols simultaneously.

Built in Memory Controllers for Low Product Cost The ETRAX FS has 4 GB of address space. It supports SDRAM, SRAM, EPROM, EEPROM, and NOR/NAND Flash PROM without external logic for fewer components and lower cost.

Crypto Accelerator Hardware accelerated wire-speed cryptography allowing efficient implementation of protocols and applications.

Designed for Networking Dual 10/100 Mbit/s full duplex Ethernet MAC and hardware support for IP checksum calculation makes the ETRAX FS ideal for networking high performance devices.

Integrated I/O There are several integrated DMA-driven I/O ports: 2 synchronous serial ports, 4 asynchronous serial ports and ATA.

Linux Kernel Source Code and Development Environment All necessary software, source code, tools, and documentation can be downloaded for free from the Axis developer site: http://developer.axis.com/.

Partnership Development Axis is committed to open-source development. Reference designs and advanced technical support enable development teams to quickly get to market with competitive products.

1.3 Technical Specifications for the ETRAX FS

32-Bit RISC CPU 200 MHz RISC CPU with a 32-bit data and address width. 16-bit instruction set optimized for compact code. Instruction pre-fetch and dynamic branch prediction. A 256-bit wide system bus. 5-stage pipeline. 1-cycle multiplication. User and kernel mode for protected memory access.

Direct Memory Access (DMA) 10 DMA channels each with 64 bytes FIFO for low latency and high throughput data transfers to and from internal and external units. 400 MB/s peak bandwidth per DMA channel. Support for real time clients through multiple virtual channels concept with fast channel switching.

Timers and Watchdog Two programmable 32-bit timers with selectable input clock frequencies. 8-bit counter able to count wraps for the 32-bit counters. One fixed 32-bit read only counter. Watchdog timer.

General Purpose Ports 80 read/write configurable I/O pins (multiplexed with other I/O functions). 8 pins can be configured as inputs for interrupts.

Memory Management Unit (MMU) Separate instruction and data MMU featuring 4 GB of virtual uniform address space for each user process. Address space protection. Supports zero-copy shared memory schemes and includes two 64-entry Translation Lookaside buffers.

Dual Ethernet Controller's Dual 10/100 Mbit/s Ethernet MAC (compatible with IEEE 802.3 and Fast Ethernet standards).

4 Asynchronous Serial Ports Full buffering and parity control. One handshake signal in each direction. Supports polled, interrupt driven and DMA controlled operation. Independent RX and TX operation. Support baud rates from 56.25 baud to 12.5 Mbaud.

Clock Generator Internal 200 MHz operating frequency, generated by a PLL from an external 12 MHz clock signal.

Cache Memory Multi-processor enabled I-and D-cache. Each cache is 16 KB and 2-way set associative and has a 256-bit interface to the system bus for high internal bandwidth. Snooping cache coherence mechanism (MESI protocol).

Interrupt Control Vectorized interrupt: Internal (I/O ports, network interface, DMA, and timer), and external (IRQ and NMI).

2 Synchronous Serial Ports I2S master and slave mode with internal and external clock. Universal SPI interface up to 16.67 MHz, some modes up to 50 MHz. Partial I2C support. Compatible with many generic synchronous serial protocols. Supports IEC 60958 mode. Up to 50 MHz internal clock generator. Up to 100 Mbit/s transmission and 300 Mbit/s reception in chip-to-chip mode.

Central Arbiter Memory arbitration providing non-blocking access to internal memory during external memory accesses. 1.6 GB/s peak internal throughput.

On-chip Debug In circuit debug with JTAG interface requiring no functional SW on the target. Hardware watchpoints, breakpoints and single step. Real-time execution path tracing via dedicated high-speed I/O.

Internal RAM 128-KB and 256-bit wide RAM with 20 ns cycle time.

ATA Single ATA controller. Support for PIO mode 4 (8 MB/s), Multiword DMA mode 2 (16 MB/s), and UltraDMA mode 2 (33 MB/s). Configuration of up to 4 ATA ports for up to 8 IDE drives.

Software and Development Tools Linux kernel and device drivers, GNU tool chain including compiler, debugger and other tools, and documentation are available free of charge from: http://developer.axis.com.

I/O Processor One master and two slave 32-bit processors running at 200 MHz. Access to internal system bus and DMA. Local memory. Function blocks also available to main CPU are clock generators, timers, trigger logic, hardware acceleration of CRC. Capable of micro code based implementation of at least two I/O protocols simultaneously. Controls 72 100 MHz I/O pins. Can be programmed to support parallel and serial ports, PC-Card/CardBus/PCI, USB 2.0 FS/HS host and device, SCSI-2/SCSI-3, ATA PIO mode 4(8 MB/s)/Multiword DMA mode 2(16 MB/s)/UltraDMA mode 2 and 5(33 and 100 MB/s), and proprietary interfaces.

USB PHY Port One FS/LS USB phy port accessible from the I/O processor. Support for host and device mode.

Bootstrap Program Download Supports initial loading to internal RAM from NOR/NAND flash PROM, serial port, and network. Code loaded to internal RAM can be designed to enable download of program to initially empty Flash PROM, or other external memory.

External Bus Interface and Memory Controllers Memory controllers for SDRAM (100 MHz), SRAM, EPROM, parallel EEPROM, NOR/NAND flash PROM. Bus width configurable to 16 or 32 bits. Support for 64-bit SDRAM DIMM and SO-DIMM modules.

Package 256 pin Plastic Ball Grid Array. 27x27 mm.

Crypto Accelerator Configurable, hardware accelerated DES, 3DES, AES, MD5, SHA-1, and IP checksum calculation for data cryptography, 2*100 Mbit/s. Equally efficient serial or parallel configuration of up to three algorithms.

Slave Mode Support for allowing an external chip (bus master) to read and write mode registers and internal memory. While in slave mode, the chip looks like an I/O device to the external bus master.

Operating Conditions Supply voltage core: 1.4 - 1.6 V

Supply voltage I/O: 3.0 - 3.6 V

Ambient temperature range: -40 - +85 °C

Power consumption: 465 mW (typical)

Chapter 2

CPU

2.1 Architectural description

2.1.1 References

Reference	Description
[DEFS]	CRIS v32 support function register constants and data types, 25.9
[GAS]	Free Software Foundation, Inc., GNU Assembler Manual, Free Software Foundation, Inc., 1999. http://sourceware.org/binutils
[GCC]	Free Software Foundation, Inc., Using the GNU Compiler Collection (GCC), Free Software Foundation, Inc., 2003. http://gcc.gnu.org
[MACROS]	CRIS v32 support function register access macros, http://developer.axis.com

Table 2.1: *CPU references*

2.1.2 Registers

The processor contains fourteen 32-bit general registers (R0 - R13), one 32-bit stack pointer (R14 or SP), one 32-bit address calculation register (R15 or ACR), and one 32-bit Program Counter (PC). Bit 0 in PC cannot be set and is ignored when PC is set. For example, bit 0 cannot be set through jump instructions and is always assumed to be 0.

Register R15 is a general register but it is intended to be used as a temporary register (e.g., for address calculations). The special features of register R15 are described in section 2.1.8.

The processor architecture also defines 16 special registers (P0 - P15) listed in the table below.

Mnemonic	Reg.No.	Description	Size(bits)
BZ	P0	Zero Byte Constant Register	8
VR	P1	Version Register	8

PID	P2	Process ID	32
SRS	P3	Support Register Select	8
WZ	P4	Zero Word Constant Register	16
EXS	P5	Exception Status	32
EDA	P6	Exception Data Address	32
MOF	P7	Multiply Overflow Register	32
DZ	P8	Zero Dword Constant Register	32
EBP	P9	Exception Base Pointer	32
ERP	P10	Exception Return Pointer	32
SRP	P11	Subroutine Return Pointer	32
NRP	P12	NMI Return Pointer	32
CCS	P13	Condition Code Stack	32
USP	P14	User Mode Stack Pointer	32
SPC	P15	Single Step PC	32

Table 2.2: *Special registers*

Three of the special registers (P0, P4 and P8) are reserved as zero registers. A read from one of these registers returns zero. A write to them has no effect. The zero registers are used implicitly by some instructions (e.g., CLEAR). The programmer never needs to explicitly use the zero registers.

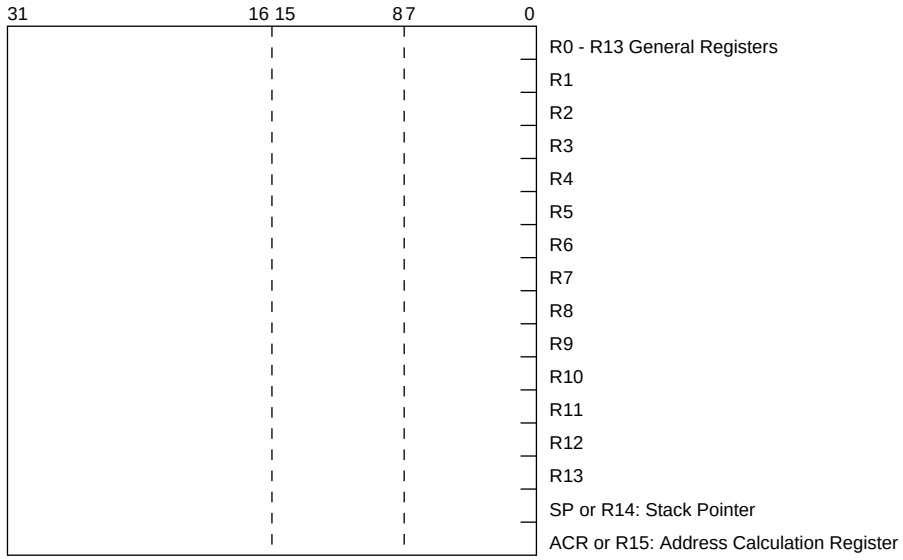


Figure 2.1: *General registers*

2.1.2.1 Support function registers

These registers are used to control support functions such as the MMU. There may be several banks of such registers, and each bank may consist of up to 16 registers. The

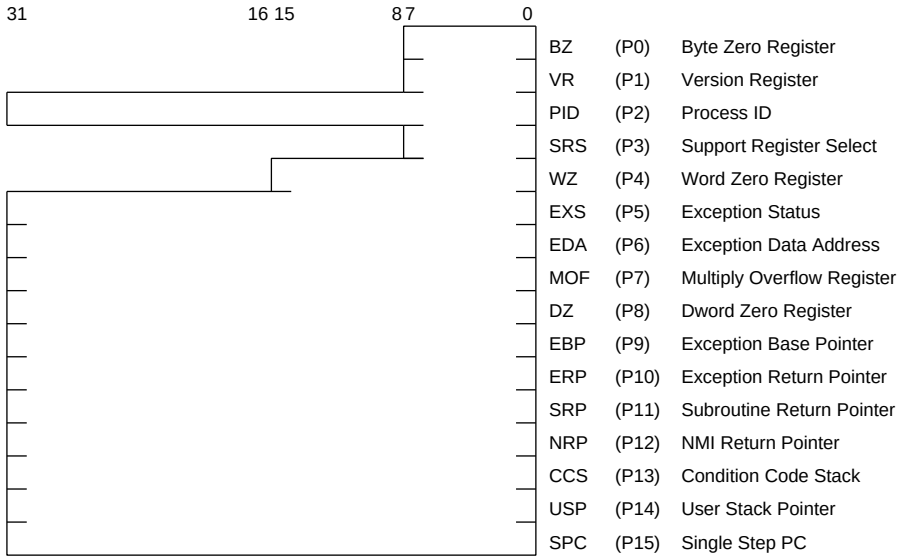


Figure 2.2: *Special registers*

special register Support Register Select (SRS) is used to select which register bank to access.

The banks which are currently available are listed in table 2.3 below.

Bank	Name	Description
0	B_GC	General configuration and Guru mode registers
1	B_IM	Instruction MMU registers
2	B_DM	Data MMU registers
3	B_BP	Breakpoint registers
255	B_Z	All support registers are always zero

Table 2.3: *Available support function register banks*

Reads or writes to other register banks will give an undefined result and should be avoided. Support function registers are accessed through two instructions, MOVE Rs,Sd and MOVE Ss,Rd that move data between the current bank of support function registers and the general registers.

Writes to Support Function Registers (i.e. MOVE Rs,Sd) have a delayed effect. A minimum of three CPU cycles are required for the value to be written in the register. Additional rules for accessing each bank of registers may be documented in each support function block.

The same delayed effect is also valid for writes to the SRS register. A minimum of three CPU cycles is required between a write to SRS and a following access to a support function register for the correct bank to be selected.

The layout and semantics of each support function register bank is defined by each support function.

The general configuration and Guru mode registers (Bank 0) are described in 25.9. To access the registers, fields and register constants from a C program, a set of macros and data types are defined in [MACROS] and 25.9.

2.1.3 Flags and condition codes

The Condition Code Stack (CCS) register contains three levels of ten different flags each, plus two separate flags.

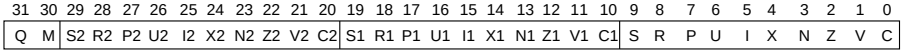


Figure 2.3: *Condition Code Stack*

Code	Description
C	Carry
V	Overflow
Z	Zero
N	Negative
X	Extend
I	Interrupt enable
U	User mode
P	Sequence broken flag
R	Restore P flag (instead of setting it to one) on RFE. Also used as carry flag by the MCP instruction.
S	Cause single step exception on instruction following the one where (SPC==PC). Also enables hardware breakpoints.
M	NMI enable flag. Cleared at reset. Once set it can only be cleared by servicing an NMI exception.
Q	Pending single step. Set when serving another exception before the single step exception.
S1-C1	First level of hardware stack for flags, used by exceptions
S2-C2	Second level of hardware stack for flags, may be used by NMI

Table 2.4: *Condition Code Stack*

When an exception occurs, the flags are shifted left into the next higher level:

```
{S2,R2,P2,U2,I2,X2,N2,Z2,V2,C2} = {S1,R1,P1,U1,I1,X1,N1,Z1,V1,C1};
{S1,R1,P1,U1,I1,X1,N1,Z1,V1,C1} = {S,R,P,U,I,X,N,Z,V,C};
          {S,R,P,U,I,X,N,Z,V,C} = 0;
```

When returning from an exception with the RFE instruction the flags are restored by shifting right:

```
          {S,R,P,U,I,X,N,Z,V,C} = {S1,R1,R ? P1:1,U1,I1,X1,N1,Z1,V1,C1};
{S1,R1,P1,U1,I1,X1,N1,Z1,V1,C1} = {S2,R2,P2,U2,I2,X2,N2,Z2,V2,C2};
{S2,R2,P2,U2,I2,X2,N2,Z2,V2,C2} = 0;
```

This is effectively a hardware stack for the flags. Only exception service routines that may cause new exceptions (e.g., routines that turns on interrupts) need to save and restore the CCS.

The flags can be tested using one of the 16 condition codes specified in the condition codes table 2.5 below. The boolean function descriptions of the condition codes and flag behavior rules are described using C syntax.

Code	Alt	Condition	Encoding	Boolean Function
CC	HS	Carry Clear	0000	!C
CS	LO	Carry Set	0001	C
NE		Not Equal	0010	!Z
EQ		Equal	0011	Z
VC		Overflow Clear	0100	!V
VS		Overflow Set	0101	V
PL		Plus	0110	!N
MI		Minus	0111	N
LS		Low or Same	1000	C || Z
HI		High	1001	!C && !Z
GE		Greater or Equal	1010	N && V || !N && !V
LT		Less Than	1011	N && !V || !N && V
GT		Greater Than	1100	N && V && !Z || !N && !V && !Z
LE		Less or Equal	1101	Z || N && !V || !N && V
A		Always True	1110	1
SB		Sequence Broken	1111	P

Table 2.5: *Condition codes*

Flag behavior for different instructions is described in chapter 2.3. In those cases where the new value of the flag is not explicitly specified, apply the rules listed in the table below.

General case:
N = R_{msb}
Z = !R_{msb} && ... && !R_{lsb} && (Z || !X)
Addition: (ADD, ADDQ, ADDS, ADDU, ADDC)
N = R_{msb}
Z = !R_{msb} && ... && !R_{lsb} && (Z || !X)
V = S_{msb} && D_{msb} && !R_{msb} || !S_{msb} && !D_{msb} && R_{msb}
C = S_{msb} && D_{msb} || D_{msb} && !R_{msb} || S_{msb} && !R_{msb}
Subtraction: (CMP, CMPQ, CMPS, CMPU, NEG, SUB, SUBQ, SUBS, SUBU)
N = R_{msb}
Z = !R_{msb} && ... && !R_{lsb} && (Z || !X)
V = !S_{msb} && D_{msb} && !R_{msb} || S_{msb} && !D_{msb} && R_{msb}
C = S_{msb} && !D_{msb} || !D_{msb} && R_{msb} || S_{msb} && R_{msb}
Multiply: (MULS and MULU)
N = M_{msb}
Z = !M_{msb} && ... && !M_{lsb} && !R_{msb} && ... && !R_{lsb} && (Z || !X)
MULS: V = ((M_{msb} || .. || M_{lsb}) && !R_{msb}) || ((!M_{msb} || .. || !M_{lsb}) && R_{msb})
MULU: V = M_{msb} || ... || M_{lsb}
Bit test: (BTST, BTSTQ)
N = D_n
Z = !D_n && ... && !D_{lsb} && (Z || !X)
Move to memory:

The P flag is set when returning from an exception by the instruction Restore From Exception (RFE/RFN) or if there is a cache miss on a conditional write. The C flag is set if a conditional write fails, and cleared if it succeeds. For normal writes the C flag is unaffected. P = RFE || (X && cache miss) C = X && P || !X && C
Move to CCS: (MOVE s,CCS and RFG)
All bits set according to source data except in User Mode where the I, U and S flags are not affected.
Condition Code Manipulation: (SETF, CLEARF)
CCS set or cleared according to mask bits in the instruction. If X is not in the list, it is cleared. U, I and S are not affected in User mode.
Condition Code Stack Manipulation: (RFE, RFN, SFE)
CCS is shifted or restored according to the instruction. The P flag is set by RFE and RFN. M is set by RFN, but Q is not affected. U, I and S are not affected in User mode.

Table 2.6: *Flag behavior*

Explanation of terms used in table 2.6 above:

$\mathbf{S}_{msb}$ Most significant bit of source operand.

$\mathbf{D}_{msb}$ Most significant bit of destination operand.

$\mathbf{D}_{n}$ Selected bit in the destination operand.

$\mathbf{D}_{lsb}$ Least significant bit of destination operand.

$\mathbf{R}_{msb}$ Most significant bit of result operand.

$\mathbf{R}_{lsb}$ Least significant bit of result operand.

$\mathbf{M}_{msb}$ Most significant bit of Multiply Overflow register (MOF).

$\mathbf{M}_{lsb}$ Least significant bit of Multiply Overflow register (MOF).

2.1.4 Data organization in memory

Data types supported by the CRIS v32 CPU are shown in the table below.

Name	Description	Size modifier
Byte	8-bit integer	.B
Word	16-bit integer	.W
Dword	32-bit integer	.D

Table 2.7: *Data types supported by the CRIS v32 CPU*

Each address location contains one byte of data. Data is stored in memory with the least significant byte at the lowest address (i.e., little endian). The CRIS v32 CPU has a 256-bit wide data bus. However, most instructions only use the three normal data types. The full width is only used during MOVEM instructions. A conversion is performed by the bus interface when narrower external memory is accessed.

Data can be aligned to any address. If the data crosses a 32-byte cache line boundary, the CPU will split the data access into two separate accesses. So, the use of unaligned word and dword data will normally not degrade performance significantly.

Maximum data size for MOVEM instructions is 64 bytes while a maximum of 32 bytes is transferred on the data bus in each cycle. If the data is aligned to a 32-byte cache line boundary, the CPU will split a full 64-byte data access into two and if not aligned into three separate accesses.

2.1.5 General instruction format

The basic instruction word is 16 bits long and instructions must be word (16 bit) aligned. The most common instructions follow the same general instruction format:

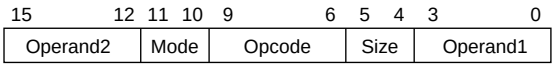


Figure 2.4: *General instruction format*

Some deviations from this format are specified in chapter 2.3.

2.1.5.1 The Opcode field

This field selects which instruction should be executed. For some opcodes, the meaning of the opcode is different depending on the Size and/or Mode field.

2.1.5.2 The Operand1 field

For most instructions, the Operand1 field selects the register for the source operand. For some instructions such as MOVE from register to memory, this field instead selects the register for the address of the destination. The meaning of this field is affected by the Mode field.

2.1.5.3 The Operand2 field

For most instructions, the Operand2 field selects which register will be the destination for the operation. For some instructions such as MOVE register to memory, this field selects instead the register that will be the data source. This field always specifies a register directly and is not affected by the Mode field.

2.1.5.4 The Mode field

The Mode field specifies the addressing mode of the instruction. The mode field affects only the operand of the Operand1 field. The following addressing modes are available:

Code	Mode	Description
00	Quick immediate mode	The Size and Operand1 fields are used as a 6 bit immediate value, extended to 32 bits. The choice of zero-extend or sign- extend depends on the instruction.
01	Register mode	The operand is contained in the register specified by the Operand1 field.
10	Indirect mode	The operand is contained in the memory location pointed to by the register specified by the Operand1 field.
11	Autoincrement mode (Operand1 != 15)	Same as Indirect mode, but the register specified by the Operand1 field is incremented after the operation. The size of the increment depends on the operand size. Using ACR (R15) as address register in Autoincrement mode is not possible. It actually results in using the PC as the address register to allow immediate constants. So R15 can thus not be used as the address register in the Autoincrement mode.
11	Immediate mode (Operand1 == 15)	The operand is contained directly after the instruction word in memory. For word and dword sized operands the PC is incremented by the size of the operand. For byte sized operands the PC is incremented by two to maintain word alignment of instructions.

Table 2.8: *The mode field of the general instruction format*

2.1.5.5 The size field

The Size field selects the size of the operation. For most instructions, The rest of the register is unaffected by the operation. Three different sizes are available as shown in the table below.

Code	Size
00	Byte (8 bits)
01	Word (16 bits)
10	Dword (32 bits)

Table 2.9: *The size field of the general instruction format*

The size code 11 is used in conjunction with the Opcode field to encode special instructions that do not need different sizes.

2.1.6 Addressing modes

2.1.6.1 General

The CRIS v32 CPU has five basic addressing modes, which are encoded in the Mode field of the instruction word. The basic addressing modes are:

- Quick immediate

- Register
- Indirect
- Autoincrement
- Immediate

2.1.6.2 Quick immediate addressing mode

In the quick immediate addressing mode, the Size and Operand1 fields of the instruction are combined into a 6-bit immediate value, extended to 32 bits, or interpreted as a 5-bit shift count. The 6-bit value may be sign- or zero-extended depending on the instruction.

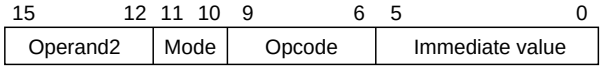


Figure 2.5: *Quick immediate addressing mode instruction format*

```
Assembler syntax:   <expression>
Example:            12
```

2.1.6.3 Register addressing mode

In the register addressing mode, the operand is contained in the register specified by the Operand1 or Operand2 field. The register can be a general register, a special register, or a support function register depending on the instruction.

2.1.6.3.1 General register addressing mode

```
Assembler syntax:   Rn
Example:            R6
```

2.1.6.3.2 Special register addressing mode

```
Assembler syntax:   Pn
Example:            SRP
```

2.1.6.3.3 Support function register addressing mode

```
Assembler syntax:   Sn
Example:            S6
```

2.1.6.4 Indirect addressing mode

In the indirect addressing mode, the operand is contained in the memory location pointed to by the register specified by the Operand1 field.

```
Assembler syntax:   [Rn]
Example:            [R6]
```

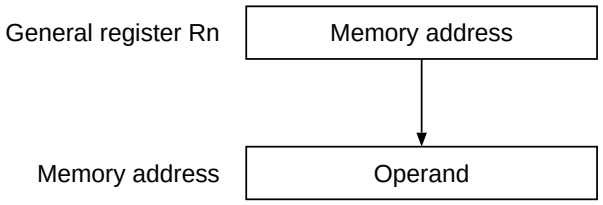


Figure 2.6: *Indirect addressing mode*

2.1.6.4.1 Autoincrement addressing mode

In the autoincrement addressing mode, the operand is contained in the memory location pointed to by the register specified by the Operand1 field. After the operand address is used, the specified register is incremented by 1, 2 or 4, depending upon the size of the operand.

```
Assembler syntax:   [Rn+]
Example:            [R6+]
```

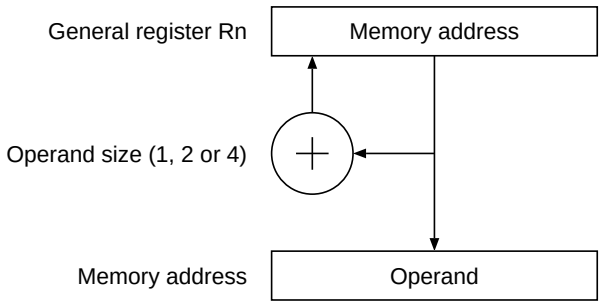


Figure 2.7: *Autoincrement addressing mode*

Note that ACR (R15) can not be used in autoincrement mode as this coding is used for immediate addressing mode. See section 2.1.6.4.2.

2.1.6.4.2 Immediate addressing mode

The immediate addressing mode is a special case of the autoincrement addressing mode, with PC as the address register. The immediate value follows directly after the instruction word. When the immediate data size is one byte, PC will be incremented by 2 to maintain word alignment of instructions. Immediate data to special registers is always 32 bits in size, regardless of the size of the special register. Surplus bits are ignored.

```
Assembler syntax:   <expression>
Example:            325
```

2.1.7 Branches jumps and subroutines

2.1.7.1 Conditional branch

The Bcc instruction (where cc represents one of the 16 condition codes described in the section 2.1.3, is a conditional relative branch instruction. If the condition is true, a signed immediate offset is added to the PC. The PC to which the offset is added is the PC of the branch instruction itself.

The Bcc instruction exists in two forms, one with an 8-bit offset contained within the basic instruction word, and one with a 16-bit immediate offset following directly after the instruction word. The assembler automatically selects between the 8-bit offset and the 16-bit offset form.

The Bcc instruction is a delayed branch instruction. This means that the instruction following directly after the Bcc instruction will always be executed, even if the branch is taken. The instruction position following the Bcc instruction is called a delay slot.

- **Example:**

```
; The branch to LOOP will be taken 4 times, and register R0
; decremented by 1 after each turn. After leaving the loop,
; R0 will have the value -1.

        :
        MOVEQ   4,R0
LOOP:
        BNE     LOOP
        SUBQ    1,R0          ; Delay slot instruction, executed
                              ; even if the branch is taken.
        :
```

There are some restrictions as to which instructions can be placed in the delay slot. Valid instructions for the delay slot are all instructions except:

- Bcc

- JAS, BAS, JASC, BASC, JUMP, HALT, FIDXI, FTAGI
- Immediate addressing other than quick immediate.

The maximum offset range that can be reached by the Bcc instruction directly is -32768 to +32766. If a larger offset is needed, the branch must be combined with a jump to reach the branch target. The assembler resolves this situation automatically and inserts the necessary code. The assembler can optionally give a warning message each time it makes this adjustment. There is also a 32 bit version of BA (Branch Always), see 2.1.7.2 for details.

2.1.7.2 Unconditional branch

The BAS (Branch And Save) instruction is an unconditional relative branch instruction. The current value of PC is first saved in a special register, and then a signed 32-bit immediate offset is added to PC. The BAS instruction is a delayed branch instruction. This means that the instruction following directly after the BAS instruction will always be executed.

The instruction position following the BAS instruction is called a delay slot. As a result, the value saved in the special register is the address of the instruction following the BAS instruction plus 6. The BAS instruction takes a 32-bit immediate offset following directly after the instruction word.

- **Examples:**

```
BAS lab,SRP          ; Jump to 'lab' using the offset (lab - PC).
                     ; (lab - PC) is added to PC. Return address
                     ; is saved in SRP.
BAS .+124,BZ         ; Jump to address PC + 124. Return address is
                     ; saved in BZ (i.e. not saved).
```

The BAS instruction can be used for both regular branches and subroutine branches. Regular branches are made by using BZ (P0) as the destination special register. Branches to subroutines are made by using SRP as the destination special register. There are two aliases for the BAS instructions that are supported by the assembler:

1. BA offset - Alias for BAS offset, BZ
2. BSR offset - Alias for BAS offset, SRP

The BA alias is useful for longer branches than the normal Bcc with 16-bit offset can handle. The BSR alias is useful in e.g., dynamically loaded shared libraries for efficient calling of subroutines when the position relative to the executing code is known, but the absolute address varies.

- **Example:**

```
(save SRP)
:
SUBQ    4,SP
BSR     sub             ; Branch and save return address in SRP
MOVE.D  R1,[SP]         ; Delay slot instruction, always executed
MOVE.D  [SP+],R1        ; Return point
:
(restore SRP)
```

2.1.7.3 Jump instructions

The JAS (Jump And Save) instruction is an unconditional absolute jump instruction. The current value of PC is first saved in a special register and then a 32-bit absolute value is loaded into PC.

The JAS instruction is a delayed jump instruction. This means that the instruction following directly after the JAS instruction will always be executed. The instruction position following the JAS instruction is called a delay slot. Because of the delayed jump the value saved in the special register is the address of the instruction following the JAS instruction plus 2.

The JAS instruction exists in two forms, one with the absolute value contained in a general register, and one with a 32-bit immediate value following directly after the instruction word.

For the register form of JAS, it is harmless for the instruction in the delay slot to modify the register holding the address. JAS will use the value before it has been modified.

· **Examples:**

```
(save SRP)
:
JAS  R3,SRP             ; Jump target is the address contained
ADDQ 1,R3               ; in register R3. Return address is
                        ; saved in SRP. R3 is incremented _after_
                        ; JAS has used the value.
JAS 124,BZ              ; Jump to address 124. Return address is
NOP                     ; saved in BZ (i.e. not saved).
:
(restore SRP)
```

The JAS instruction can be used for both regular jumps and subroutine jumps. Regular jumps are made by using BZ (P0) as the destination special register. Jumps to subroutines are made by using SRP as the destination special register. There are two aliases for the JAS instructions that are supported by the assembler:

1. JUMP Rs - Alias for JAS Rs, BZ

   (JUMP address) Alias for JAS address, BZ

2. JSR Rs - Alias for JAS Rs, SRP
   (JSR address) Alias for JAS address, SRP

· **Example:**

```
(save SRP)
:
SUBQ    4,SP
JSR     R6              ; Jump and save return address in SRP
MOVE.D  R1,[SP]         ; Delay slot instruction, always executed
MOVE.D  [SP+],R1        ; Return point
:
(restore SRP)
```

2.1.7.4 Jump and branch with context

The JASC (Jump And Save with Context) and BASC (Branch And Save with Context) instructions work like the regular JAS and BAS instructions except that the address saved in the special register is the address of the instruction following the JASC or BASC instruction plus 6. This leaves four bytes unused between the JASC or BASC instruction and the return point. These four bytes can, for example, be used for C++ exception handling information.

In case of the register addressing form, the unused bytes are placed directly after the delay slot instruction.

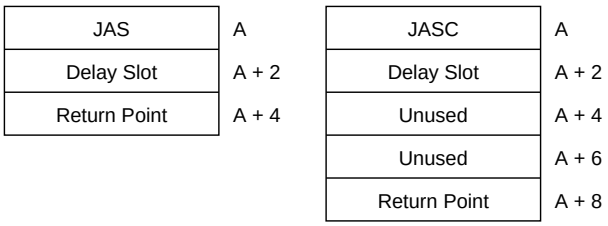


Figure 2.8: *Register form of the JASC instruction*

In case of immediate addressing form, the unused bytes are placed after the immediate value and the delay slot instruction.

There are three aliases for the JASC and BASC instructions that are supported by the assembler.

1. JSRC Rs - Alias for JASC Rs,SRP
2. JSRC address - Alias for JASC address,SRP
3. BSRC offset - Alias for BASC offset,SRP

· **Example:**

JAS/BAS	A
Immediate	A + 2
Value	A + 4
Delay slot	A + 6
Return point	A + 8

JASC/BASC	A
Immediate	A + 2
Value	A + 4
Delay slot	A + 6
Unused	A + 8
Unused	A + 10
Return Point	A + 12

Figure 2.9: *Immediate form of the JAS/BAS and JASC/BASC instructions*

```
(save SRP)
:
SUBQ    4,SP
JSRC    R6              ; Jump and save return address in SRP
MOVE.D  R1,[SP]         ; Delay slot instruction, always executed
.WORD   0               ; Unused
.WORD   0               ; Unused
MOVE.D  [SP+],R1        ; Return point
:
(restore SRP)
```

2.1.7.5 Return instructions

The JUMP instruction is an unconditional absolute jump instruction. The JUMP instruction exists only in one form with the absolute value contained in a special register. It is intended to be used to return from subroutines and exceptions. There are three aliases for the JUMP instruction that are supported by the assembler for this purpose:

1. RET - Alias for JUMP SRP
2. RETE - Alias for JUMP ERP
3. RETN - Alias for JUMP NRP

The return instructions are delayed jump instructions. This means that the instruction following directly after the return instruction will always be executed. The instruction position following the return instruction is called a delay slot.

2.1.7.6 Switches and table jumps

A common element in many high level languages is the switch statement. A typical switch construct in C can look like this:

```
switch (sel_val) {
```

```
    case 6:
      a = b + c;
      break;
    case 7:
      a = b * (c + d);
      break;
    case 8:
      a = b + c + d;
      break;
    default:
      a = b + c;
      break;
  }
```

A switch construct in the CRIS v32 assembler can be implemented in several different ways. Three examples based on jump tables are shown below. The first example uses a table of absolute addresses. The second example uses relative addressing, while the third example uses a table of branch instructions.

- **Example:** A switch construct with a table of absolute addresses

```
            MOVE     [sel_val],R0   ; Load selector value to R0.
            SUBQ     6,R0           ; Adjust table index by subtracting
                                    ; the lowest selector value.
            BOUND.B  3,R0           ; Adjust index to point to default
                                    ; case if it is out of range.
            LAPCQ    L_TAB,ACR      ; Calculate address of table base.
            ADDI     R0.D,ACR,ACR   ; Add dword index to base.
            MOVE.D   [ACR], R0      ; Load jump address.
            JUMP     R0             ; Table jump.
            NOP                     ; Delay slot.
    L_TAB:  .DWORD   L6             ; Address to case 6.
            .DWORD   L7             ; Address to case 7.
            .DWORD   L8             ; Address to case 8.
            .DWORD   L_DEF          ; Address to default case.
    L6:
            :
            (Perform case 6)
            :
            BA       L_END          ; Break
            Op or NOP               ; Delay slot
    L7:
            :
            (Perform case 7)
            :
            BA       L_END          ; Break
            Op or NOP               ; Delay slot
    L8:
            :
            (Perform case 8)
            :
            BA       L_END          ; Break
```

```
        Op or NOP               ; Delay slot
L_DEF:
        :
        (Perform default case)
        :
L_END:
```

· **Example:** A switch construct with a table of relative addresses

```
        MOVE     [sel_val],R0   ; Load selector value to R0.
        SUBQ     6,R0           ; Adjust table index by subtracting
                                ; the lowest selector value.
        BOUND.B  3,R0           ; Adjust index to point to default
                                ; case if it is out of range.
        LAPCQ    L_TAB,ACR      ; Calculate address to table base.
        ADDI     R0.W,ACR,ACR   ; Add word index to base.
        ADDS.W   [ACR], ACR     ; Add PC-relative word offset
                                ; to "case" to address of the entry.
        JUMP     ACR            ; Table jump.
        NOP                     ; Delay slot.
L_TAB:  .WORD    L6 - .         ; Relative address to case 6.
        .WORD    L7 - .         ; Relative address to case 7.
        .WORD    L8 - .         ; Relative address to case 8.
        .WORD    L_DEF - .      ; Relative address to default case.
L6:
        :
        (Perform case 6)
        :
        BA       L_END          ; Break
        Op or NOP               ; Delay slot
L7:
        :
        (Perform case 7)
        :
        BA       L_END          ; Break
        Op or NOP               ; Delay slot
L8:
        :
        (Perform case 8)
        :
        BA       L_END          ; Break
        Op or NOP               ; Delay slot
L_DEF:
        :
        (Perform default case)
        :
L_END:
```

· **Example:** A switch construct with a table of branch instructions

```
        MOVE     [sel_val],R0   ; Load selector value to R0.
```

```
        SUBQ    5,R0            ; Adjust table index so the lowest
                                ; value is 1.
        BOUND.B 4,R0            ; Adjust index to point to default
                                ; case if it is out of range.
        LSLQ    2,R0            ; Adjust to dword index.
        LAPCQ   L_TAB,ACR       ; Calculate address to table base.
        ADD.D   ACR,R0          ; Calculate address of the "case".
        JUMP    R0              ; Table jump.
        NOP                     ; Delay slot.
L_TAB:  BA      L6              ; Relative address to case 6
        NOP                     ; Delay slot.
        BA      L7              ; Relative address to case 7
        NOP                     ; Delay slot.
        BA      L8              ; Relative address to case 8
        NOP                     ; Delay slot.
        BA      L_DEF           ; Relative address to case L_DEF
        NOP                     ; Delay slot.
L6:
        :
        (Perform case 6)
        :
        BA      L_END           ; Break
        Op or NOP               ; Delay slot
L7:
        :
        (Perform case 7)
        :
        BA      L_END           ; Break
        Op or NOP               ; Delay slot
L8:
        :
        (Perform case 8)
        :
        BA      L_END           ; Break
        Op or NOP               ; Delay slot
L_DEF:
        :
        (Perform default case)
        :
L_END:
```

2.1.7.7 Subroutines

The JSR and BSR instructions of the CRIS v32 CPU do not automatically push the return address for a subroutine on the stack. Instead, the return address is stored in a special register called the Subroutine Return Pointer (SRP).

For terminal subroutines (subroutines that do not call other subroutines), the return address can be kept in the SRP throughout the subroutine. In this way, the overhead for a subroutine call can be reduced to two single-cycle instructions.

For non-terminal subroutines, the contents of the SRP must be explicitly pushed on the stack. It is preferred that this is done as the first instruction of the subroutine.

Terminal subroutine:

```
SUB_ENTRY:
              :                    ; No pushing of SRP needed.
              :
           ( Perform desired function )
              :
              :
           RET                     ; Return, take address from SRP.
           (op or NOP)             ; Delay slot after return.
```

Non-terminal subroutine:

```
SUB_ENTRY:
           SUBQ 4, SP
              :                    ; Insert instruction(s) to hide
              :                    ; latency (optional).
           MOVE SRP,[SP]           ; Push SRP on the stack.
              :
              :
           ( Perform desired function )
              :
           MOVE [SP+], SRP
              :                    ; Insert instruction(s) to hide
              :                    ; latency (optional).
           RET                     ; Return, take address from SRP.
           (op or NOP)             ; Delay slot after return.
```

2.1.8 Address calculation instructions

The address calculation instructions, ADDO, ADDOQ, ADDI, LAPC and LAPCQ, are primarily intended to be used to calculate addresses used by other instructions. The address calculation instructions are often used with ACR (R15) as destination. The instruction using the address can then simply use indirect addressing with ACR as the address register.

- **Example:**

```
ADDOQ -4,R5,ACR
MOVE.D R3,[ACR]
```

The N, Z, V and C flags are not affected by the address calculation instructions to make it possible to place them between instructions modifying the flags and instructions using the flags.

2.1.9 PC Relative addressing

PC relative addressing is performed by the instructions LAPC and LAPCQ. They store the sum of PC and an immediate offset in a general register. LAPC takes a 32 bit immediate signed offset while LAPCQ contains a 4 bit unsigned offset for PC offsets smaller than 32. This is useful in e.g., dynamically loaded shared libraries where local variables needs to be accessed in an efficient way.

- **Example:**

```
LAPC .+34,R1          ; Add 34 and the current value
                      ; of PC, store in R1.
MOVE.D R3,[R1]
LAPCQ .+4,R1          ; R1 = PC+4
```

2.1.10 Exceptions

The CRIS v32 CPU uses vectorized exceptions. The term exception is used as a common name for interrupts, Non Maskable Interrupts (NMIs), and bus faults. The following steps describes the exception sequence for all types of exceptions (except Guru Mode exceptions, which are described in section 2.1.10.6):

1. Store the contents of PC to either the NMI Return Pointer (NRP) for NMIs, and for all other exceptions to the Exception Return Pointer (ERP). Set the D field (of ERP or NRP) if the instruction interrupted was in a delay slot. See section 2.1.10.4.1 for details on the D bit. NRP uses the same format as ERP. Note that the return address is not automatically pushed on the stack.
2. Shift the CCS:

```
{S2,R2,P2,U2,I2,X2,N2,Z2,V2,C2} = {S1,R1,P1,U1,I1,X1,N1,Z1,V1,C1};
{S1,R1,P1,U1,I1,X1,N1,Z1,V1,C1} = {S,R,P,U,I,X,N,Z,V,C};
{S,R,P,U,I,X,N,Z,V,C}           = 0;
```

3. Enter kernel mode. If it is not an NMI, update EXS and EDA registers. If it is an NMI, clear the NMI enable flag (M).
4. Fetch the exception handler address [EBP + vector index]. This address is translated by the MMU if it is enabled. The MMU must not cause exceptions during this access. If it does the behavior is undefined.
5. Jump to the fetched address

Only bits 10 to 31 of the Exception Base Pointer (EBP) are used when fetching the jump address. The remaining bits are ignored.

Unlike many other microprocessors, the CRIS v32 CPU does not automatically push the condition codes and the return address on the stack. The exception return address

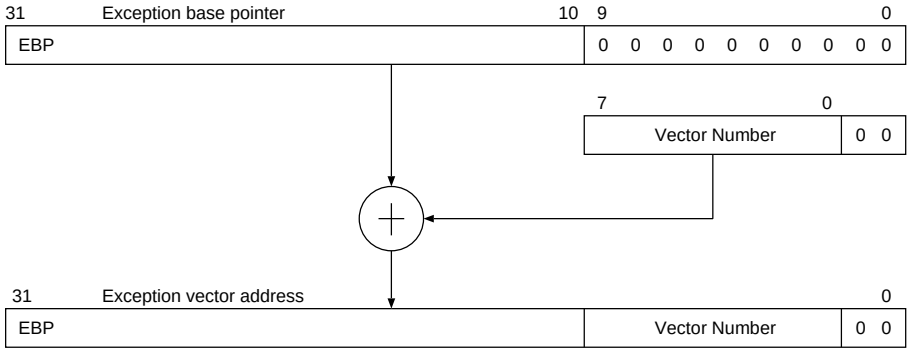


Figure 2.10: *Exception vector address calculation*

is stored in the Exception Return Pointer (ERP). If nested exceptions are used, the ERP must be pushed on the stack before interrupts are reenabled in the exception routine. The Condition Code Register must also be saved when nested exceptions are used. At the end of the exception routine the saved ERP and CCS must be restored before returning.

The Interrupt Enable flag is automatically set to zero by the CCS shift.

Returning from interrupt and bus fault exceptions can be done in a uniform way by the Return From Exception (RETE) instruction. When returning from an NMI or Guru mode exception the Return From NMI (RETN) instruction is used instead.

The Restore From Exception (RFE) instruction is used to restore (un-shift) the CCS content when returning from an exception. The RFE instruction is intended to be used in the delay slot of the RETE instruction. In the same way, the RFN and RFG instructions are intended to be used in the delay slot of the RETN instruction.

2.1.10.1 Exception examples

- **Example:** Single level exception

```
ENTRY:    SUBQ   sizeof(regs),SP ; Reserve stack for used registers
          MOVEM  Rn,[SP]         ; Save used registers
            :
            :
          (Perform desired function)
            :
            :
          MOVEM [SP+],Rn         ; Restore registers
          RETE                   ; Return, take address from ERP
          RFE                    ; Restore CCS in delay slot
```

- **Example:** Nested exceptions

```
ENTRY:      SUBQ   4,SP
            MOVE   CCS,[SP]         ; Save CCS
            SUBQ   4,SP
            MOVE   ERP,[SP]         ; Save ERP
            <save other status
             if necessary>
            EI                      ; Enable Interrupts
            SUBQ   sizeof(regs),SP ; Reserve stack for used registers
            MOVEM  Rn,[SP]          ; Save registers
              :
              :
            (Perform desired function)
              :
              :
            MOVEM [SP+],Rn          ; Restore registers
            MOVE  [SP+],CCS         ; Restore CCS
                                    ; <-- Interrupts are disabled
            MOVE  [SP+],ERP         ; Restore ERP
            RETE                    ; Return, take address from ERP
            RFE                     ; Restore CCS in delay slot
```

Note that the sequences described above will restore the state also if the exception routine is called by software (using the instruction sequence JAS ENTRY, ERP; SFR). If, for example, the interrupts were disabled at the time of the JAS instruction, the interrupts will also be disabled after return from the exception routine.

2.1.10.2 Exception vectors

The vector numbers are defined in table 2.10 below:

Index(hex)	Function
0	NMI
1	(Reserved)
2	(Reserved)
3	Single step
4	Instruction TLB refill
5	Instruction TLB invalid address
6	Instruction TLB access violation
7	Instruction TLB execute violation
8	Data TLB refill
9	Data TLB invalid address
a	Data TLB access violation
b	Data TLB write error
c	Hardware Breakpoint
d	(Reserved)
e	(Reserved)
f	(Reserved)
10	Break 0
11	Break 1
12	Break 2

13	Break 3
14	Break 4
15	Break 5
16	Break 6
17	Break 7
18	Break 8
19	Break 9
1a	Break 10
1b	Break 11
1c	Break 12
1d	Break 13
1e	Break 14
1f	Break 15
20	Interrupts
..	-”-
ff	-”-

Table 2.10: *Defined vector numbers*

2.1.10.3 Exception priority

In case of multiple simultaneous exceptions the following priority order applies (for any given instruction):

1. Guru mode exception
2. NMI
3. Hardware breakpoint
4. Instruction TLB refill
5. Instruction TLB invalid address
6. Instruction TLB access violation
7. Instruction TLB execute violation
8. Data TLB refill
9. Data TLB invalid address
10. Data TLB access violation
11. Data TLB write error
12. Interrupts (priority between interrupts is decided by interrupt controller)
13. Break instruction
14. Single step

Note that the low priority of the break instruction and single step exceptions comes from fact that these exceptions are given after the instruction has been executed (but before the next instruction is executed). As all other exception related to an instruction will come before it is executed they will have higher priority.

2.1.10.4 Exception status registers

These registers are loaded when an exception (except NMI) is triggered.

2.1.10.4.1 ERP - Exception return pointer

The exception return pointer has the following format.

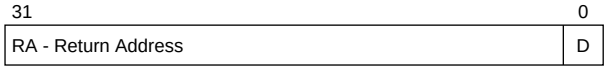


Figure 2.11: *Exception return pointer format*

The ERP contains two fields:

ERP Fields	Description
RA	Bits 31 to 1 of the address of the instruction is where execution should resume after the exception has been serviced (bit 0 of the address is always zero for instruction addresses). This is either the address of the instruction that was interrupted when the exception was triggered, or the address of the instruction preceding this instruction if the interrupted instruction was in a delay slot. If the interrupted instruction was in a delay slot, the D bit is set. Note that an interrupted instruction is counted as being in a delay-slot even for a non-taken branch.
D	The D field is set when the interrupted instruction was in a delay slot. If set, the address of the interrupted instruction is RA+2 for register addressing forms, and RA+4 or RA+6 for the immediate addressing forms of the jump or branch instruction.

Table 2.11: *ERP fields*

2.1.10.4.2 EXS - Exception status

The Exception status has the following format.

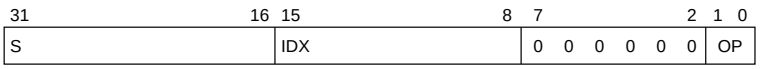


Figure 2.12: *Exception status format*

The EXS contains the following fields:

EXS Fields	Description	
OP	Contains the memory operation performed by the interrupted instruction when an instruction was interrupted, otherwise undefined. The contents of this field depends only on the type of the interrupted instruction, not on the type of exception. The memory operation is coded as follows:	
	00	No memory operation
	01	Read
	10	Write
	11	Flush
IDX	Contains the exception vector index, the vector number for the exception service routine.	
S	The 16 bit status field can be used by support function exceptions, e.g., breakpoint exceptions. The contents of this field depends on the unit causing the exception. For interrupt and NMI exceptions this field is undefined.	

Table 2.12: *EXS fields*

Bits 7 to 2 in EXS are always loaded with zero when an exception is triggered.

2.1.10.4.3 EDA - Exception data address

The exception data address (EDA) is the base address of the memory access performed by the interrupted instruction. If no memory operation was performed, the value of this register is undefined.

Observe that, in the case of accesses that are sequenced due to a crossed data cache line border, the EDA may contain the base address of any of the sequenced accesses depending on when the instruction was interrupted.

2.1.10.4.4 SPC - Single Step PC

When a single step or breakpoint exception is taken, the address of the interrupted instruction is saved in the SPC. This makes it easy to start/continue single stepping after the exception routine.

2.1.10.5 Non Maskable Interrupts (NMI)

NMI exceptions may come at any time, even when other exceptions are in progress. The differences from a normal exception are:

- NMIs are disabled M=0 when entering the NMI service routine.
- The return address is stored in NRP instead of ERP.
- The NMI service routine should exit with the sequence:

```
RETN
RFN
```

- NMI service routines must not modify the S and R bits in the CCS.

The M flag is cleared at reset. Once set it is can only be cleared by an NMI exception as described above. The M flag can be set by a MOVE to CCS or by RFN, which always sets the M flag.

2.1.10.6 Guru mode

Guru mode is a totally unmaskable exception level. Unlike other exceptions, it fetches its jump vector at a hardwired address (i.e. it does not use EBP). When in guru mode, all new exceptions are ignored. The Guru mode is intended to be used in combination with on chip ROM to implement a stub-less debugging mode.

Guru mode is always entered when requested by external logic. The CPU can also be configured to enter Guru mode at breakpoint and/or single step or break 15 exceptions. This is configured in the rw_gc_cfg support function register in bank 0.

2.1.10.6.1 Entering guru mode

When Guru mode is entered the following actions take place:

- CCS is saved in support function register S1 (rw_gc_ccs) in bank 0.
- CCS is set to 0.
- NRP is saved in support function register S3 (rw_gc_nrp) in bank 0.
- The restart PC is stored in NRP.
- SRS is saved in support function register S2 (rw_gc_srs) in bank 0.
- SRS is set to 0 (bank 0).
- Exceptions status is saved in S4 (rw_gc_exs). The register rw_gc_exs has the same layout as special register EXS. The IDX field of rw_gc_exs is set to zero if guru mode was entered due to an external request. Otherwise it is set according to 2.1.10.2.
- Exception data address is saved in S5 (rw_gc_eda). The register rw_gc_eda has the same function as EDA in standard exceptions.
- The hidden guru mode flag (g-flag) is set.
- The jump vector is fetched at a fixed address (0x3c0000f0).
- Execution is started at the fetched address.

Support function registers S8 through S11 (rw_gc_r0 through rw_gc_r3) in bank 0 are available for storing a couple of general registers when in guru mode. The start of the guru mode exception handler thus typically starts like this:

```
guru_handler:  MOVE  R0,S8
               MOVE  R1,S9
               MOVE  R2,S10
               MOVE  R3,S11
               ...
```

Detailed information about the guru mode support function registers (Bank 0) can be found in 25.9.

2.1.10.6.2 Leaving guru mode

When leaving guru mode the state of the CPU has to be restored. This is done with the RETN and RFG instructions:

```
RETN
RFG
```

RFG restores the CCS, NRP and SRS saved when guru mode was entered, and clears the g-flag. A typical guru mode exception handler epilogue looks like this:

```
MOVE  S11,R3
MOVE  S10,R2
MOVE  S9,R1
MOVE  S8,R0
RETN
RFG
```

2.1.10.6.3 Protected resources in guru mode

If the gb bit in rw_gc_cfg is set, the following registers may only be written from guru mode:

- SPC
- All hardware breakpoint registers

2.1.11 MMU support

2.1.11.1 Overview

There are two separate memory management units attached to the CPU, one for the instruction fetch mechanism and one for data accesses. To support the memory management units a number of features are available:

- The CPU can be in one of two different operation modes: User mode and Kernel mode. The MMUs use the operation mode to select the appropriate mapping between logical and physical addresses. Which mode the CPU operates in is decided by the U flag in the Condition Code Stack (CCS).
- A set of MMU-Fault exceptions that triggers when the MMU fails to translate a logical address. There are eight such exceptions:

  1. Instruction TLB refill - The translation is not in the TLB.
  2. Instruction TLB invalid address - The translated address is invalid.
  3. Instruction TLB access violation - User mode access to kernel area.
  4. Instruction TLB execute violation - Execute access to execute protected area.
  5. Data TLB refill - The translation is not in the TLB.
  6. Data TLB invalid address - The translated address is invalid.
  7. Data TLB access violation - User mode request to kernel area.
  8. Data TLB write error - Write request to write protected area.

The S field of the EXS register is not used by the MMUs and will be set to zero when an MMU exception is triggered.

The User and Kernel modes have different stack pointers. In both modes the user mode stack pointer can be referenced as USP, while the currently active stack pointer is referenced as SP (or R14). Thus, in User mode, SP and USP refer to the same register, while in Kernel mode they are separate registers.

Observe that, when modifying USP in User mode, the modified value is not available as SP the three following cycles. In the same way, when modifying SP in User mode, the modified value is not available as USP the three following cycles.

Once User mode is entered (by setting the U flag), Kernel mode can only be reentered by triggering an exception. Explicitly entering Kernel mode for e.g., operating system calls are typically made through the BREAK instruction.

2.1.11.2 Protected resources

A few registers, flags and instructions need to be protected from being modified while the CPU is in User mode. The protected resources are:

- The Q, M, S, U and I flags in the CCS
- EBP (Exception Base Pointer)
- PID (Process/Page ID)
- SRS (Support Register Select)
- SPC (Single step PC)
- Any Support function register (through MOVE Rs,Sd)

- FIDXI, FIDXD, FTAGI and FTAGD instructions
- HALT instruction

Any attempt to modify a protected register or flag in User mode will just be silently denied. It will not cause any exception. The protected registers and flags are readable in both User and Kernel modes.

2.1.11.3 Transitions between operation modes

A transition between the User and Kernel modes can take place for the following reasons:

- Transition to User mode:
  - SETF U
  - RFE when U1 is set
- Transitions to Kernel mode:
  - System Reset
  - Any exception

The stack pointers will be automatically exchanged at a transition between the two operation modes.

The following example shows how User mode can be entered for the first time in an application:

- **Example:**

```
MOVE   user_stack_pointer,USP
JUMP   user_mode_program_entry
SETF   U                        ; set U flag in delay slot
```

2.1.11.4 MMU registers

Registers for controlling the MMUs are available as support function registers. The MMU registers are located in support register banks 1 and 2 for instruction and data MMU respectively. These can be accessed in Kernel mode through the MOVE Rs,Sd or MOVE Ss,Rd instructions. More information about support function registers is found in section 2.1.2.1.

The specific layout and semantics of these registers is specified in the MMU Module documentation.

2.1.12 Multiply and divide

2.1.12.1 General

The CRIS v32 CPU has two multiply instructions, Signed Multiply (MULS) and Unsigned multiply (MULU).

There are no divide instructions, so divide operations are performed by a sequence of Divide Step (DSTEP) instructions.

2.1.12.2 Multiply

The MULU and MULS are fast (one cycle) multiply operations. The multiply is performed on 32 by 32 bits, giving a 64-bit result. The lower 32 bits are stored to the destination register specified with the instruction, while the upper 32 bits are stored in the Multiply Overflow (MOF) register.

For multiply with byte or word sized operands, the operands are extended to 32 bits before the multiply. sign-extend is used with Signed Multiply (MULS), while zero-extend is used with Unsigned Multiply (MULU).

2.1.12.3 Divide

Divide operation can be performed using the DSTEP instruction. The DSTEP instruction does the following:

1. Shifts the destination register one step to the left.
2. If the destination register is unsigned-greater-than or equal to the source operand, the source operand is subtracted from the destination register.

· **Example:** 16-bit by 16-bit unsigned divide

```
DIV_BEGIN:
       MOVE.W   [num],R1    ; Move numerator to a register,
                            ; and clear the upper 16 bits.
       MOVE.W   [denom],R0  ; Move denominator to a register.
       LSLQ     16,R0       ; Shift left, clear the lower 16 bits
                            ; of the denominator register.
       SUBQ     1,R0        ; Subtract one from the denominator.
       DSTEP    R0,R1       ; Perform 16 iterations of DSTEP
       DSTEP    R0,R1       ; instructions.
       DSTEP    R0,R1
       DSTEP    R0,R1
       DSTEP    R0,R1
       DSTEP    R0,R1
       DSTEP    R0,R1
       DSTEP    R0,R1
       DSTEP    R0,R1
```

```
DSTEP    R0,R1
DSTEP    R0,R1
DSTEP    R0,R1
DSTEP    R0,R1
DSTEP    R0,R1
DSTEP    R0,R1
DSTEP    R0,R1       ; The last iteration. The quotient is
                     ; in the lower half of R1, and the
                     ; remainder is in the upper half of R1.
```

2.1.13 Extended arithmetic

Extended arithmetic (arithmetic with more than 32 bits) is supported by either using the X flag or using instructions that always use carry propagation. The X flag is set by the AX (SETF X) instruction, and cleared by all other instructions.

When the X flag is set, instructions involving an addition or subtraction are modified in the following ways, which is also the behavior of instructions that always use carry propagation:

1. The C flag is added to the result of an addition, and subtracted from the result of a subtraction. This is valid even if the addition/subtraction result is not the result operand of the instruction.
2. If the result operand is zero, the Z flag will maintain its old value, instead of being set.

The change of the Z flag behavior applies to all instructions that affect the Z flag except CLEARF, MOVE s,CCS, and SETF.

The addition/subtraction of the C flag affects the following instructions:

ADD, ADDI, ADDQ, ADDS, ADDU, ADDO, ADDOQ, CMP, CMPQ, CMPS, CMPU, NEG, SUB, SUBQ, SUBS, and SUBU.

The addition/subtraction of the C flag has an undefined effect on the following instructions:

BOUND, ABS, DSTEP.

Below are 2 examples of extended arithmetic using the X flag:

- **Example:** Add a 48-bit signed value contained in R4:R3 to a 64 bit value stored in R2:R1

```
EXT_ADD:
          ADD.D   R3,R1          ; Add the low dwords.
          AX                     ; Set X flag.
          ADDS.W  R4,R2          ; Add the upper 16 bits of source
```

- **Example:** Test if a 40-bit value contained in memory pointed to by R1 is zero

```
EXT_TEST:
          TEST.D  [R1+]           ; Test low 32 bits.
          AX
          TEST.B  [R1]            ; Test upper 8 bits.
```

The instruction ADDC always uses the C flag as input and output for carry propagation. Similarly, the instruction MCP uses the R flag as input and output carry. ADDC and MCP use these flags for more efficient handling of large number add and multiply operations rather than using the X flag.

The ADDC instruction adds two dword operands using the C flag as carry input. This is the same behavior as a normal ADD.D with the X flag set. ADDC can be used to pack several add with carry together without the need for setting the X flag in between.

Below is an example of how to summarize a vector of four dword elements in memory using the ADDC instruction.

- **Example:**

```
CLEAR.D R1              ; Clear result register.
MOVE    [vec_base],R0   ; Load vector base address to R0.
ADD.D   [R0+],R1        ; Add without carry
ADDC    [R0+],R1        ; Use input carry
ADDC    [R0+],R1        ; Use input carry
ADDC    [R0+],R1        ; Use input carry
```

The MCP instruction can be used to propagate the carry during large number multiply. The inner loop of a large number multiplication looks something like this:

```
mof = 0;
R = 0;
for (j = 0; j < size; j++) {
   {R,r[j]} = r[j] + mof + R            // mcp
   {mof,x}  = w * b[j];                 // mulu
   {C,r[j]} = r[j] + x + C;             // addc
}
```

· **Example:** Inner loop of multiplication of two large numbers using the MCP instruction

```
;;; Multiply R9 with 8 long vector at address in R10,
;;; add result to vector at address in R11

  mul_p:  MOVEM   [R11],R8        ; load vector at R11 to R0-R8

          MOVE.D  [R10+],R12      ; index 0
          MOVE    0,CCS           ; clear R and C flags
          MULU.D  R9,R12
          ADD.D   R12,R0

          MOVE.D  [R10+],R12      ; index 1
          MCP     MOF,R1
          MULU.D  R9,R12
          ADDC    R12,R1

          MOVE.D  [R10+],R12      ; index 2
          MCP     MOF,R2
          MULU.D  R9,R12
          ADDC    R12,R2

          MOVE.D  [R10+],R12      ; index 3
          MCP     MOF,R3
          MULU.D  R9,R12
          ADDC    R12,R3

          MOVE.D  [R10+],R12      ; index 4
          MCP     MOF,R4
          MULU.D  R9,R12
          ADDC    R12,R4

          MOVE.D  [R10+],R12      ; index 5
          MCP     MOF,R5
          MULU.D  R9,R12
          ADDC    R12,R5

          MOVE.D  [R10+],R12      ; index 6
          MCP     MOF,R6
          MULU.D  R9,R12
          ADDC    R12,R6

          MOVE.D  [R10+],R12      ; index 7
          MCP     MOF,R7
          MULU.D  R9,R12
          ADDC    R12,R7

          MCP     MOF,R8
          ADDC    0,R8            ; index 8

          MOVEM   R8,[R11]        ; write back result vector
```

```
RET
NOP
```

2.1.14 Integral read-write operations

Since some exceptions, like MMU-faults and breakpoints can interrupt the CPU in almost any cycle, it is not possible to ensure the integrity of a piece of code just by disabling the interrupts. Instead integral read-write operations can be implemented by using the Load-Locked, Store-Conditional principle:

```
Start:
Initialize the lock;
Read variable;
Modify variable;
Write back variable if and only if the sequence hasn't been interrupted;
Go to Start if write failed;
```

The P flag in the CCS and the instruction Branch on Carry Cleared BCC, are used to accomplish this. The P flag is set when returning from an exception by the instruction Restore From Exception (RFE) or Restore From NMI (RFN). To make this mechanism useful in a multi-processor environment with a cache coherence protocol in action, the P flag is also set if there is a cache miss on a conditional write.

A conditional write is accomplished by setting the X flag with the instruction before an instruction that writes to memory. Only normal write instructions can result in conditional writes, the MOVEM instruction can not be used for this purpose. A conditional write is only carried out if the P flag hasn't been set (i.e. is zero) and the cache doesn't miss on the write. The Carry flag is set if the write failed, cleared if it succeeded. The conditional write must be aligned within one single 32-byte cache line for the mechanism to work. Otherwise, parts of the data might be incorrectly written to the cache. This can be avoided by e.g using only byte sized conditional writes.

Pseudo code for the conditional write to memory:

```
if (!P) {
  write conditionally to cache;
  if (cache misses) {
    C = 1;
  } else {
    C = 0;
  }
} else {
  C = 1;
}
```

The conditional write to cache operation only takes effect if the cache has a valid copy of the requested memory.

A code example of how the feature can be used to implement a spin-lock:

· **Example:**

```
start_lock:  CLEARF  P                        ; start "unbreakable"
lock_loop:   MOVE.B  [memory_location], r1    ; sequence read
             BEQ     start_lock               ; if zero, the lock is taken
             NOP                              ; by someone else, retry
             AX
             CLEAR.B [memory_location]        ; write conditional
             BCS     lock_loop                ; try again if failed
             CLEARF  P
```

2.1.15 Single step

Single stepping of instructions means that a single step exception is generated after every executed instruction in a program. A feature useful in debuggers. Single stepping is enabled by setting the S flag and giving SPC (Single step PC) a suitable value.

A single step exception is generated whenever either of the following conditions are true:

· The S flag is set and the PC of the instruction that was just executed is equal to SPC.

· The S flag and the Q flag are both set.

When a single step exception is taken, SPC is updated with the PC of the instruction that was just to be executed. This means that the next single step exception will be generated after the next instruction has been executed (if the S flag is still set). The exception routine itself is of course not single stepped as the S flag will be cleared when the CCS is shifted.

The Q flag means ”pending single step exception” and is set if a single step exception can’t be taken immediately due to a break instruction exception (see section 2.1.10.3 for exception priorities). Thus the single step exception will be taken when the higher priority break exception has finished. The Q flag is cleared when the single step exception is finally taken. There is in general no need for software to consider the Q flag.

2.1.15.1 Single step examples

· **Example:** Initiating single step

  This is typically done from an exception routine, e.g., a break n handler.

  This code will generate the first single step exception after the instruction restarted after returning from this exception has been executed:

```
                ...
enable_sstep:  MOVE  CCS,R0
```

```
                OR.D  1<<19,R0  ;; Set S1 flag (S1 will be copied
                                ;; to S by RFE)
                MOVE  R0,CCS
                MOVE  ERP,R0
                MOVE  R0,SPC

                <restore state>

                RETE
                RFE
```

This code will generate the first single step exception immediately on return from this exception handling routine:

```
                ...
enable_sstep:   MOVE  CCS,R0
                OR.D  1<<19 | 1<<31,R0  ;; Set Q and S1 flag (S1 will
                                        ;; be copied to S by RFE)
                MOVE  R0,CCS

                <restore state>

                RETE
                RFE
```

- **Example:** Single step exception handler

  Simplest possible single step exception handler:

```
sstep_handler: RETE
               RFE
```

Single step exception handler that does something useful may look something like this:

```
sstep_handler:
               <save state>

               <do useful stuff like displaying the instruction
                just executed in the single stepped program>

               <restore state>

               RETE
               RFE
```

2.1.16 Hardware breakpoints and watchpoints

Hardware breakpoints and or watchpoints consist of dedicated hardware that detects when certain virtual memory addresses are accessed or when code from certain virtual addresses is executed by the CPU.

The difference between a breakpoint and a watchpoint is that a breakpoint generates an exception while a watchpoint just notifies hardware outside the CPU (e.g., real time program trace hardware) when the specified criterion is detected. Observe that this definition differs from the one often used in debugging tools where the term watchpoint is used for a breakpoint on data accesses.

The same hardware is used for breakpoints and watchpoints and each breakpoint and/or watchpoint may be used as either a breakpoint, a watchpoint or both at once. There is one breakpoint and/or watchpoint for instruction fetches and six for data memory accesses.

To enable hardware breakpoints the following must be done:

- Provide a hardware breakpoint exception handler.
- Specify for which virtual addresses and access types the breakpoints shall trigger. This is done via registers in support function register bank 3.
- Set the S flag for the program(s) where the breakpoints are valid. As the S flag is also used for enabling single stepping, make sure SPC is set to a value that will not trigger single stepping (unless single stepping is actually wanted). This can be done by setting bit 0 of SPC to one. See section 2.1.18 for details on single stepping.

To enable the hardware watchpoints, just specify the virtual addresses and access types in the registers in support function registers bank 3.

2.1.16.1 Support function registers

The following support function registers are used to control the hardware breakpoints and/or watchpoints. They are located in support register bank 3, i.e. SRS shall be set to 3 when accessing these registers.

Reg.No.	Name	Description
0	BP_CTRL	Specifies for which access types each breakpoint shall trigger
1	BP_I0_START	Instruction break/watch-point 0 start address
2	BP_I0_END	Instruction break/watch-point 0 end address
3	BP_D0_START	Data break/watch-point 0 start address
4	BP_D0_END	Data break/watch-point 0 end address
5	BP_D1_START	Data break/watch-point 1 start address
6	BP_D1_END	Data break/watch-point 1 end address
7	BP_D2_START	Data break/watch-point 2 start address
8	BP_D2_END	Data break/watch-point 2 end address
9	BP_D3_START	Data break/watch-point 3 start address
10	BP_D3_END	Data break/watch-point 3 end address
11	BP_D4_START	Data break/watch-point 4 start address
12	BP_D4_END	Data break/watch-point 4 end address
13	BP_D5_START	Data break/watch-point 5 start address
14	BP_D5_END	Data break/watch-point 5 end address
15		Reserved

Table 2.13: *Hardware breakpoint support function registers*

The control register (BP_CTRL) has the following layout:

Bit.No.	Name	Description
0	I0_BP	When set, enables instruction breakpoint 0
1	I0_WP	When set, enables instruction watchpoint 0
2	D0_BPRD	When set, data breakpoint 0 will trigger on read accesses.
3	D0_BPWR	When set, data breakpoint 0 will trigger on write accesses.
4	D0_WPRD	When set, data watchpoint 0 will trigger on read accesses.
5	D0_WPWR	When set, data watchpoint 0 will trigger on write accesses.
6	D1_BPRD	When set, data breakpoint 1 will trigger on read accesses.
7	D1_BPWR	When set, data breakpoint 1 will trigger on write accesses.
8	D1_WPRD	When set, data watchpoint 1 will trigger on read accesses.
9	D1_WPWR	When set, data watchpoint 1 will trigger on write accesses.
10	D2_BPRD	When set, data breakpoint 2 will trigger on read accesses.
11	D2_BPWR	When set, data breakpoint 2 will trigger on write accesses.
12	D2_WPRD	When set, data watchpoint 2 will trigger on read accesses.
13	D2_WPWR	When set, data watchpoint 2 will trigger on write accesses.
14	D3_BPRD	When set, data breakpoint 3 will trigger on read accesses.
15	D3_BPWR	When set, data breakpoint 3 will trigger on write accesses.
16	D3_WPRD	When set, data watchpoint 3 will trigger on read accesses.
17	D3_WPWR	When set, data watchpoint 3 will trigger on write accesses.
18	D4_BPRD	When set, data breakpoint 4 will trigger on read accesses.
19	D4_BPWR	When set, data breakpoint 4 will trigger on write accesses.
20	D4_WPRD	When set, data watchpoint 4 will trigger on read accesses.
21	D4_WPWR	When set, data watchpoint 4 will trigger on write accesses.
22	D5_BPRD	When set, data breakpoint 5 will trigger on read accesses.
23	D5_BPWR	When set, data breakpoint 5 will trigger on write accesses.
24	D5_WPRD	When set, data watchpoint 5 will trigger on read accesses.
25	D5_WPWR	When set, data watchpoint 5 will trigger on write accesses.
26-31		Reserved

Table 2.14: *BP_CTRL register layout*

2.1.16.2 Triggering condition

When an access falls within the specified criterion of an enabled breakpoint and/or watchpoint it is said to trigger. Below is specified under which condition a breakpoint and/or watchpoint triggers.

An instruction breakpoint triggers when all of the following are true:

- The breakpoint is enabled. Bit In_BP in the BP_CTRL register is set to one.
- The address and size of the instruction to be executed conforms to:

```
address + size >  BP_In_START &&
address        <= BP_In_END
```

A data breakpoint triggers when all of the following is true:

- The data access is either a:
  - Read and Dn_BPRD is set in the BP_CTRL register, or
  - Write and Dn_BPWR is set in the BP_CTRL register
- The address and size of the access conforms to:

```
address + size >  BP_In_START &&
address        <= BP_In_END
```

An instruction watchpoint triggers when all of the following is true:

- The watchpoint is enabled. Bit In_WP in the BP_CTRL register is set to one.
- The address and size of the instruction to be executed conforms to:

```
address + size >  BP_In_START &&
address        <= BP_In_END
```

A data watchpoint triggers when all of the following is true:

- The data access is either a:
  - Read and Dn_WPRD is set in the BP_CTRL register, or
  - Write and Dn_WPWR is set in the BP_CTRL register
- The address and size of the access conforms to:

```
address + size >  BP_In_START &&
address        <= BP_In_END
```

2.1.16.3 Exceptions

When a hardware breakpoint triggers, a hardware breakpoint exception is generated. When this exception in taken, the following also happens:

- SPC is updated with the PC of the instruction that was interrupted.
- The S field of EXS is set according to the table below.

Bit.No.	Name	Description
0	I0	Set if instruction breakpoint 0 has triggered
1	D0_RD	Set if data breakpoint 0 has triggered on a read
2	D0_WR	Set if data breakpoint 0 has triggered on a write
3	D1_RD	Set if data breakpoint 1 has triggered on a read
4	D1_WR	Set if data breakpoint 1 has triggered on a write
5	D2_RD	Set if data breakpoint 2 has triggered on a read
6	D2_WR	Set if data breakpoint 2 has triggered on a write
7	D3_RD	Set if data breakpoint 3 has triggered on a read
8	D3_WR	Set if data breakpoint 3 has triggered on a write
9	D4_RD	Set if data breakpoint 4 has triggered on a read
10	D4_WR	Set if data breakpoint 4 has triggered on a write
11	D5_RD	Set if data breakpoint 5 has triggered on a read
12	D5_WR	Set if data breakpoint 5 has triggered on a write
13-15		Reserved, shall be ignored

- For data breakpoints, EDA is set to the address causing the breakpoint to trigger.

After a breakpoint exception has been handled and the execution of the interrupted program is to be resumed, some care must be taken. If the breakpoint(s) that triggered remains enabled and the state of the resumed program is unaltered, the breakpoint(s) will trigger immediately again when the program is resumed. This is generally not a desirable behavior.

One way to avoid this is to disable the breakpoint(s) and single step past the triggering instruction and then turn the breakpoint(s) on again (as described in an example below).

2.1.16.4 Examples

- **Example:** Setting up an instruction breakpoint

```
MOVE   3,SRS
NOP                  ; let update of SRS propagate
NOP
NOP
MOVE.D start_addr,R0
MOVE   R0,S1         ; instr bp 0 start addr
MOVE.D end_addr,R0
MOVE   R0,S2         ; instr bp 0 end addr
MOVE.D 0x1,R0
MOVE   R0,S0         ; enable instr breakpoint 0
```

· **Example:** Setting up a data read/write breakpoint

```
MOVE   3,SRS
NOP                  ; let update of SRS propagate
NOP
NOP
MOVE.D start_addr,R0
MOVE   R0,S3         ; data bp 0 start addr
MOVE.D end_addr,R0
MOVE   R0,S4         ; data bp 0 end addr
MOVE.D 0xC,R0
MOVE   R0,S0         ; enable data breakpoint 0
                     ; for reads and writes
```

· **Example:** Hardware breakpoint exception handler

Example of how single step can be used to step past a triggering breakpoint after it has been handled.

As SPC is set when hwbp_handler is entered, the single step exception sstep_handler will be called after the instruction that triggered the breakpoint has been restarted and executed. The single step handler then turns the breakpoint back on.

```
hwbp_handler:
       <save state>

       <do useful stuff with breakpoint status>

       <disable triggering breakpoint>

       <restore state>

       RETE
       RFE

sstep_handler:
       <save state>

       MOVE    R0,CCS
       MOVE    [SP+],R0

       <re-enable breakpoint>

       MOVE    1,SPC         ; disable single step

       RETE
       RFE
```

2.1.17 Version identification

Different versions of the CRIS CPU architecture can be identified by reading the Version Register (VR). The version register is an 8-bit read-only register that contains the

CPU version number.

In older chip implementations the VR register has been used to identify the chip version (i.e. not only the CPU version). Starting with the CRIS v32 architecture the content of the VR register belongs to the CRIS CPU only. The CPU version number may therefore stay the same through several chip implementations. The chip version is identified by reading a chip version number outside the CPU.

The content of the VR register for different CRIS architectures and chip implementations can be found in the following table:

Value	Architecture	Chip Name	Part No	Note
0	CRIS v0	ETRAX E1	13425	
1	CRIS v1	ETRAX E2	13576	
2	CRIS v2	ETRAX E3	13873	
3	CRIS v3	ETRAX E4	14517	
4-7				Reserved for future use in the ETRAX family.
8	CRIS v8	ETRAX 100 E1	15822	
9	CRIS v8	ETRAX 100 E2	16284	
10	CRIS v10	ETRAX 100LX E1	17511	
11	CRIS v10	ETRAX 100LX E2	17854	
11	CRIS v10	ETRAX 100LX E3	18816	
11	CRIS v10	ETRAX 100LX E3	19322	Lead (Pb) free.
12-15				Reserved for future use in the ETRAX 100 family.
16	CRIS v8	ARTPEC 2	19054	
16	CRIS v8	ARTPEC 2	20667	Lead (Pb) free.
17-31				Reserved for future use in the ARTPEC family.
32	CRIS v32	ETRAX FS	21050	
32	CRIS v32	ETRAX FS	24745	Lead (Pb) free.
33-255				Not assigned.

Table 2.16: *CRIS Version register*

2.1.18 Reset

On reset, the CPU will start executing at a specific address. This address points out the start address of the internal boot ROM. The CPU starts out in Kernel mode (flag U=0) with NMI and interrupts disabled (flags M=0 and I=0).

CCS = 0 Kernel Mode, Interrupts and NMI disabled

PC = boot_rom_entry Boot ROM entry address (0x3c000100)

The caches are disabled after reset but this is not visible for normal program code in ETRAX FS. The internal boot ROM will initialize and enable the caches before jumping to normal program code. Memory management is always disabled after reset. See 6.4.1 for more information.

2.2 Instruction set description

2.2.1 General

2.2.1.1 Definitions

In the instruction descriptions, the following definitions apply:

m	Size modifier, byte (00), word (01) or dword (10)
z	Size modifier, byte (0) or word (1)
Rs	Source operand, register addressing mode
[Rs]	Source operand, indirect addressing mode
[Rs+]	Source operand, autoincrement addressing mode (note)
k	8,16 or 32 bit immediate operand
s	Source operand, any of the modes Rs, [Rs] or [Rs+] or k
Ps	Source operand, special register
i	6-bit signed immediate operand
j	6-bit unsigned immediate operand
c	5-bit immediate shift value
qo	4-bit unsigned immediate operand
Rd	Destination operand, register addressing mode
[Rd]	Destination operand, indirect addressing mode
[Rd+]	Destination operand, autoincrement addressing mode
d	Destination operand, any of the modes Rd, [Rd] or [Rd+]
Pd	Destination operand, special register
o	8-bit immediate offset value
cc	Condition code
n	4 bit breakpoint exception vector index
ao	32 bit immediate address offset operand
aa	32 bit immediate absolute address operand
bo	16 bit immediate branch offset operand

Table 2.17: *Instruction description definitions*

Note: [ACR+] or [R15+] is not possible as this coding is for the immediate addressing mode.

For a description of how the flags are affected, the following definitions apply:

-	Flag not affected
0	Flag cleared
1	Flag set
*	Flag affected according to the result of the operation. Refer to section 2.1.3 for details.

Table 2.18: *How flags are affected*

Instructions, register specifications, condition code specifications, and size modifiers may be written in upper or lower case. Upper case is used throughout this manual to

distinguish instructions from normal text.

2.2.1.2 Size modifiers

Many of the CRIS instructions can operate on the three different data types byte (8 bits), word (16 bits) and dword (32 bits). The size of the operation or operand is indicated by a size modifier added to the instruction. The size modifiers are:

Name	Description	Size modifier
Byte	8-bit integer	.B
Word	16-bit integer	.W
Dword	32-bit integer or address	.D

Table 2.19: *Size modifiers*

2.2.1.3 Addressing modes

The addressing modes of the CRIS CPU are described in table 2.20 below. For a detailed description of each addressing mode, refer to section 2.1.6.

Assembler Syntax	Addressing mode
i , j	Quick immediate
Rn	Register
Pn	Special register
Sn	Support function register
[Rn]	Indirect
[Rn+]	Autoincrement
x , u	Byte immediate
xx , uu	Word immediate
xxxx , uuuu	Dword immediate

Table 2.20: *Addressing modes*

2.2.2 Instruction function summary

Instructions for the CRIS CPU are listed alphabetically in the following sections below together with their flag operation and a brief description.

2.2.2.1 Address calculation instructions

Address calculation instructions for the CRIS CPU are described in the table below.

Instruction	S	R	P	U	I	X	N	Z	V	C	Description
ADDO.m [Rs],Rd,ACR	-	-	-	-	-	0	-	-	-	-	Add sign extended source
ADDO.m [Rs+],Rd,ACR	-	-	-	-	-	0	-	-	-	-	Add sign extended source
ADDO.m k,Rd,ACR	-	-	-	-	-	0	-	-	-	-	Add sign extended source
ADDOQ o,Rs,ACR	-	-	-	-	-	0	-	-	-	-	Add 8-bit signed offset
ADDI Rs.m,Rd	-	-	-	-	-	0	-	-	-	-	Add scaled index to base

ADDI Rs.m,Rd,ACR	-	-	-	-	-	0	-	-	-	-	Add scaled index to base
LAPC k,Rd	-	-	-	-	-	0	-	-	-	-	Assign Rd sum of PC plus operand
LAPCQ qo,Rd	-	-	-	-	-	0	-	-	-	-	Assign Rd sum of PC plus constant*2

Table 2.21: *Address calculation instructions*

2.2.2.2 Arithmetic instructions

The arithmetic instructions for the CRIS CPU are listed in the table below.

Instruction	S	R	P	U	I	X	N	Z	V	C	Description
ABS Rs,Rd	-	-	-	-	-	0	*	*	0	0	Absolute value
ADD.m s,Rd	-	-	-	-	-	0	*	*	*	*	Add source to destination register
ADDC s,Rd	-	-	-	-	-	0	*	*	*	*	Add source with carry bit to register
ADDQ j,Rd	-	-	-	-	-	0	*	*	*	*	Add 6-bit unsigned immediate
ADDS.z s,Rd	-	-	-	-	-	0	*	*	*	*	Add sign extended source to register
ADDU.z s,Rd	-	-	-	-	-	0	*	*	*	*	Add zero extended source to register
CMP.m s,Rd	-	-	-	-	-	0	*	*	*	*	Compare source to register
CMPQ i,Rd	-	-	-	-	-	0	*	*	*	*	Compare with 6-bit signed immediate
CMPS.z [Rs],Rd	-	-	-	-	-	0	*	*	*	*	Compare with sign extended source
CMPS.z [Rs+],Rd	-	-	-	-	-	0	*	*	*	*	Compare with sign extended source
CMPS.z k,Rd	-	-	-	-	-	0	*	*	*	*	Compare with sign extended source
CMPU.z [Rs],Rd	-	-	-	-	-	0	*	*	*	*	Compare with zero extended source
CMPU.z [Rs+],Rd	-	-	-	-	-	0	*	*	*	*	Compare with zero extended source
CMPU.z k,Rd	-	-	-	-	-	0	*	*	*	*	Compare with zero extended source
DSTEP Rs,Rd	-	-	-	-	-	0	*	*	-	-	Divide step
MCP Ps,Rd	-	*	-	-	-	0	*	*	*	-	Multiply carry propagation
MULS.m Rs,Rd	-	-	-	-	-	0	*	*	*	-	Signed multiply
MULU.m Rs,Rd	-	-	-	-	-	0	*	*	*	-	Unsigned multiply
NEG.m Rs,Rd	-	-	-	-	-	0	*	*	*	*	Negate (2’s complement)
SUB.m s,Rd	-	-	-	-	-	0	*	*	*	*	Subtract source from register
SUBQ j,Rd	-	-	-	-	-	0	*	*	*	*	Subtract 6-bit unsigned immediate
SUBS.z s,Rd	-	-	-	-	-	0	*	*	*	*	Subtract sign extended source
SUBU.z s,Rd	-	-	-	-	-	0	*	*	*	*	Subtract zero extended source
TEST.m [Rs]	-	-	-	-	-	0	*	*	0	0	Compare source with 0
TEST.m [Rs+]	-	-	-	-	-	0	*	*	0	0	Compare source with 0

Table 2.22: *Arithmetic instructions*

2.2.2.3 Bit test instructions

The bit test instructions for the CRIS CPU are shown in the table below. The BTST and BTSTQ instructions set the N flag according to the selected bit in the destination register. The Z flag is set if the selected bit and all bits to the right of the destination register are zero. When the bit number is contained in a register, the 6 least significant bits of the register are used as an unsigned bit number.

Instruction	S	R	P	U	I	X	N	Z	V	C	Description
BTST Rs,Rd	-	-	-	-	-	0	*	*	-	-	Test bit Rs in register Rd

BTSTQ c,Rd	-	-	-	-	-	0	*	*	-	-	Test bit c in register Rd

Table 2.23: *Bit test instructions*

2.2.2.4 Cache manipulation instructions

Instructions used to manage the caches for the CRIS CPU are shown in the table below.

Instruction	S	R	P	U	I	X	N	Z	V	C	Description
FIDXD [Rs]	-	-	-	-	-	0	-	-	-	-	Flush data cache with index
FIDXI [Rs]	-	-	-	-	-	0	-	-	-	-	Flush instruction cache with index
FTAGD [Rs]	-	-	-	-	-	0	-	-	-	-	Flush data cache with address
FTAGI [Rs]	-	-	-	-	-	0	-	-	-	-	Flush instruction cache with address

Table 2.24: *Cache manipulation instructions*

2.2.2.5 Condition code manipulation instructions

The condition code manipulation instructions for the CRIS CPU are shown in the table below. The predefined assembler macros EI, DI, and AX are also shown.

Instruction	S	R	P	U	I	X	N	Z	V	C	Description
AX	-	-	-	-	-	1	-	-	-	-	Arithmetic extend (SETF X)
CLEARF <list of flags> Kernel Mode	-	-	*	*	*	0	*	*	*	*	Clear flags in list
CLEARF <list of flags> User Mode	-	-	*	-	-	0	*	*	*	*	Clear flags in list
DI Kernel Mode	-	-	-	-	0	0	-	-	-	-	Disable interrupts (CLEARF I)
EI Kernel Mode	-	-	-	-	1	0	-	-	-	-	Enable interrupts (SETF I)
RFE Kernel Mode	*	*	*	*	*	*	*	*	*	*	Restore from exception
RFE User Mode	-	*	*	-	-	*	*	*	*	*	Restore from exception
RFG Guru Mode	*	*	*	*	*	*	*	*	*	*	Restore from guru mode
RFG User and Kernel Mode	-	-	-	-	-	-	-	-	-	-	Restore from guru mode
RFN Kernel Mode	*	*	*	*	*	*	*	*	*	*	Restore from NMI exception
RFN User Mode	-	*	*	-	-	*	*	*	*	*	Restore from NMI exception
SETF <list of flags> Kernel Mode	-	-	*	*	*	*	*	*	*	*	Set flags in list
SETF <list of flags> User Mode	-	-	*	-	-	*	*	*	*	*	Set flags in list
SFE Kernel Mode	0	0	0	0	0	0	0	0	0	0	Save for exception
SFE User Mode	-	0	0	-	-	0	0	0	0	0	Save for exception

Table 2.25: *Condition code manipulation instructions*

2.2.2.6 Data transfers

The data transfer instructions for the CRIS CPU are listed in the table below.

Instruction	S	R	P	U	I	X	N	Z	V	C	Description
CLEAR.m d	-	-	-	-	-	0	-	-	-	-	Clear destination operand
MOVE Ps,Rd	-	-	-	-	-	0	-	-	-	-	Move from source special register to destination
MOVE Ps,[Rd] X flag cleared	-	-	-	-	-	0	-	-	-	-	Move from source special register to destination
MOVE Ps,[Rd] X flag set	-	-	-	-	-	0	-	-	-	*	Move from source special register to destination
MOVE Ps,[Rd+] X flag cleared	-	-	-	-	-	0	-	-	-	-	Move from source special register to destination
MOVE Ps,[Rd+] X flag set	-	-	-	-	-	0	-	-	-	*	Move from source special register to destination
MOVE Rs,Pd Pd != CCS	-	-	-	-	-	0	-	-	-	-	Move from source to destination special register
MOVE Rs,Pd Pd = CCS, Kernel Mode	*	*	*	*	*	*	*	*	*	*	Move from source to destination special register
MOVE Rs,Pd Pd = CCS, User Mode	-	*	*	-	-	*	*	*	*	*	Move from source to destination special register
MOVE [Rs],Pd Pd != CCS	-	-	-	-	-	0	-	-	-	-	Move from source to destination special register
MOVE [Rs],Pd Pd = CCS, Kernel Mode	*	*	*	*	*	*	*	*	*	*	Move from source to destination special register
MOVE [Rs],Pd Pd = CCS, User Mode	-	*	*	-	-	*	*	*	*	*	Move from source to destination special register
MOVE [Rs+],Pd Pd != CCS	-	-	-	-	-	0	-	-	-	-	Move from source to destination special register
MOVE [Rs+],Pd Pd = CCS, Kernel Mode	*	*	*	*	*	*	*	*	*	*	Move from source to destination special register
MOVE [Rs+],Pd Pd = CCS, User Mode	-	*	*	-	-	*	*	*	*	*	Move from source to destination special register
Instruction	S	R	P	U	I	X	N	Z	V	C	Description
MOVE Rs,Sd	-	-	-	-	-	0	-	-	-	-	Move from source to support function register
MOVE Ss,Rd	-	-	-	-	-	0	-	-	-	-	Move from support function register to source
MOVE.m d,Rd	-	-	-	-	-	0	*	*	-	-	Move from source to general register
MOVE.m Rs,[Rd] X flag cleared	-	-	-	-	-	0	-	-	-	-	Move from general register to destination
MOVE.m Rs,[Rd+] X flag cleared	-	-	-	-	-	0	-	-	-	-	Move from general register to destination
MOVE.m Rs,[Rd] X flag set	-	-	-	-	-	0	-	-	-	*	Move from general register to destination

MOVE.m Rs,[Rd+] X flag set	-	-	-	-	-	0	-	-	-	*	Move from general register to destination
MOVEM Rs,[Rd]	-	-	-	-	-	0	-	-	-	-	Move multiple registers to memory
MOVEM Rs,[Rd+]	-	-	-	-	-	0	-	-	-	-	Move multiple registers to memory
MOVEM [Rs],Rd	-	-	-	-	-	0	-	-	-	-	Move from memory to multiple registers
MOVEM [Rs+],Rd	-	-	-	-	-	0	-	-	-	-	Move from memory to multiple registers
Instruction	S	R	P	U	I	X	N	Z	V	C	Description
MOVEQ	-	-	-	-	-	0	-	-	-	-	Move 6-bit signed immediate to destination register
MOVS.z s,Rd	-	-	-	-	-	0	*	*	-	-	Move sign extended source to destination register
MOVU.z s,Rd	-	-	-	-	-	0	0	*	-	-	Move zero extended source to destination register

Table 2.26: *Data transfer instructions*

2.2.2.7 Jump and Branch Instructions

The jump and branch instructions of the CRIS CPU are shown in the table below. The predefined assembler macros RET and RETE are also shown. Note that all jump and branch instructions (except BREAK) have a delayed effect, see section 2.1.7.

Instruction	S	R	P	U	I	X	N	Z	V	C	Description
BA k	-	-	-	-	-	0	-	-	-	-	Alias for BAS k,P0
BAS ao,Pd	-	-	-	-	-	0	-	-	-	-	Branch and save
BASC ao,Rd	-	-	-	-	-	0	-	-	-	-	Branch and save w context
Bcc o	-	-	-	-	-	0	-	-	-	-	Conditional relative branch
Bcc bo	-	-	-	-	-	0	-	-	-	-	Branch with 16-bit offset
BREAK n	-	-	-	-	-	0	-	-	-	-	Breakpoint
BSR ao	-	-	-	-	-	0	-	-	-	-	Branch to subroutine
BSRC ao	-	-	-	-	-	0	-	-	-	-	Branch to subroutine w context
JAS Rs,Pd	-	-	-	-	-	0	-	-	-	-	Jump and save
JAS aa,Pd	-	-	-	-	-	0	-	-	-	-	Jump and save
JASC Rs,Pd	-	-	-	-	-	0	-	-	-	-	Jump and save w context
JASC aa,Pd	-	-	-	-	-	0	-	-	-	-	Jump and save w context
JSR Rs	-	-	-	-	-	0	-	-	-	-	Jump to subroutine
JSR aa	-	-	-	-	-	0	-	-	-	-	Jump to subroutine
JSRC Rs	-	-	-	-	-	0	-	-	-	-	Jump to subroutine w context
JSRC aa	-	-	-	-	-	0	-	-	-	-	Jump to subroutine w context
JUMP Rs	-	-	-	-	-	0	-	-	-	-	Jump to absolute address
JUMP aa	-	-	-	-	-	0	-	-	-	-	Jump to absolute address
JUMP Ps	-	-	-	-	-	0	-	-	-	-	Jump to special register
RET	-	-	-	-	-	0	-	-	-	-	Return from subroutine
RETE	-	-	-	-	-	0	-	-	-	-	Return from exception
RETN	-	-	-	-	-	0	-	-	-	-	Return from NMI exception

Table 2.27: *Flag operation for jump and branch instructions*

2.2.2.8 Logical Instructions

The logical instructions for the CRIS CPU are described in the table below.

Instruction	S	R	P	U	I	X	N	Z	V	C	Description
AND.m s,Rd	-	-	-	-	-	0	*	*	-	-	Bitwise logical AND
ANDQ i,Rd	-	-	-	-	-	0	*	*	-	-	AND with 6-bit signed immediate
NOT Rd	-	-	-	-	-	0	*	*	-	-	Bitwise logical NOT
OR.m s,Rd	-	-	-	-	-	0	*	*	-	-	Bitwise logical OR
ORQ i,Rd	-	-	-	-	-	0	*	*	-	-	OR with 6-bit signed immediate
SWAP<option> Rd	-	-	-	-	-	0	*	*	-	-	Swap operand bits
XOR Rs,Rd	-	-	-	-	-	0	*	*	-	-	Bitwise Exclusive OR

Table 2.28: *Logical instructions*

The SWAP instruction takes one or more of the following option flags.

Flag	Description
N	Invert all bits in the operand.
W	Swap the words of the operand.
B	Swap the two bytes within each word of the operand.
R	Reverse the bit order within each byte of the operand.

2.2.2.9 Miscellaneous data operations

Miscellaneous data operation instructions for the CRIS CPU are described in the table below.

Instruction	S	R	P	U	I	X	N	Z	V	C	Description
BOUND.m Rs,Rd	-	-	-	-	-	0	*	*	-	-	Adjust table index (unsigned min)
BOUND.m k,Rd	-	-	-	-	-	0	*	*	-	-	Adjust table index (unsigned min)
LZ	-	-	-	-	-	0	0	*	-	-	Number of leading zeros
NOP	-	-	-	-	-	0	-	-	-	-	No operation (alias for SETF)
Scc Rd	-	-	-	-	-	0	-	-	-	-	Set register according to cc

Table 2.30: *Miscellaneous data operations*

2.2.2.10 Shift instructions

The shift instructions for the CRIS CPU are shown in the table below. When the shift count is contained in a register, the 6 least significant bits of the register are used as an unsigned shift count.

Instruction	S	R	P	U	I	X	N	Z	V	C	Description
ASR.m Rs,Rd	-	-	-	-	-	0	*	*	-	-	Right shift Rd with sign fill
ASRQ c,Rd	-	-	-	-	-	0	*	*	-	-	Right shift Rd with sign fill
LSL.m Rs,Rd	-	-	-	-	-	0	*	*	-	-	Left shift Rd with zero fill

LSLQ c,Rd	-	-	-	-	-	0	*	*	-	-	Left shift Rd with zero fill
LSR.m Rs,Rd	-	-	-	-	-	0	*	*	-	-	Right shift Rd with zero fill
LSRQ c,Rd	-	-	-	-	-	0	*	*	-	-	Right shift Rd with zero fill

Table 2.31: *Shift instructions*

2.2.3 Instruction format summary

In each of the tables below, instructions are listed in numerical order according their opcodes.

2.2.3.1 Quick immediate mode instructions

The following table summarizes the quick immediate mode instructions.

Note that the (s.) field of Bcc o is the sign bit of the offset.

	Bit number/General instruction format															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Operation	Operand2				Mode		Opcode				Operand1					
Bcc o	Condition				0	0	0	0	Offset							s.
ADDOQ o,Rs,ACR	Base				0	0	0	1	Signed Displacement							
BREAK n	1	1	1	0	1	0	0	1	0	0	1	1	n			
ADDQ i,Rd	Dest. Reg.				0	0	1	0	0	0	Unsigned Immed.					
MOVEQ i,Rd	Dest. Reg.				0	0	1	0	0	1	Signed Immediate					
SUBQ i,Rd	Dest. Reg.				0	0	1	0	1	0	Unsigned Immed.					
CMPQ i,Rd	Dest. Reg.				0	0	1	0	1	1	Signed Immediate					
ANDQ i,Rd	Dest. Reg.				0	0	1	1	0	0	Signed Immediate					
ORQ i,Rd	Dest. reg.				0	0	1	1	0	1	Signed Immediate					
BTSTQ c,Rd	Dest. Reg.				0	0	1	1	1	0	0	Bit Number				
ASRQ c,Rd	Dest. Reg.				0	0	1	1	1	0	1	Shift Value				
LSLQ c,Rd	Dest. Reg.				0	0	1	1	1	1	0	Shift Value				
LSRQ c,Rd	Dest. Reg.				0	0	1	1	1	1	1	Shift Value				

Table 2.32: *Quick immediate mode instructions*

2.2.3.2 Register instructions with variable size

The first table below gives the different sizes of the instructions, and the second table summarizes the register instructions of variable size.

z	Size	zz	Size
0	byte	00	byte
1	word	01	word
		10	dword

Table 2.33: *Instruction sizes*

	Bit number/General instruction format															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Operation	Operand2				Mode		Opcode				Size		Operand1			
ADDU.z Rs,Rd	Dest. Reg.				0	1	0	0	0	0	0	z	Source Reg.			
ADDS.z Rs,Rd	Dest. Reg.				0	1	0	0	0	0	1	z	Source Reg.			
MOVU.z Rs,Rd	Dest. Reg.				0	1	0	0	0	1	0	z	Source Reg.			
MOVS.z Rs,Rd	Dest. Reg.				0	1	0	0	0	1	1	z	Source Reg.			
SUBU.z Rs,Rd	Dest. Reg.				0	1	0	0	1	0	0	z	Source Reg.			
SUBS.z Rs,Rd	Dest. Reg.				0	1	0	0	1	0	1	z	Source Reg.			
LSL.m Rs,Rd	Dest. Reg.				0	1	0	0	1	1	z	z	Source Reg.			
ADDI Rs.m,Rd	Source Reg				0	1	0	1	0	0	z	z	Dest. Reg.			
MULU.m Rs,Rd	Dest. Reg.				1	0	0	1	0	0	z	z	Source Reg.			
MULS.m Rs,Rd	Dest. Reg.				1	1	0	1	0	0	z	z	Source Reg.			
ADDI Rs.m,Rd,ACR	Source Reg				0	1	0	1	0	1	z	z	Dest. Reg.			
NEG.m Rs,Rd	Dest. Reg.				0	1	0	1	1	0	z	z	Source Reg.			
BOUND.m Rs,Rd	Dest. Reg				0	1	0	1	1	1	z	z	Source Reg.			
ADD.m Rs,Rd	Dest. Reg.				0	1	1	0	0	0	z	z	Source Reg.			
MOVE.m Rs,Rd	Dest. Reg.				0	1	1	0	0	1	z	z	Source Reg.			
SUB.m Rs,Rd	Dest. Reg.				0	1	1	0	1	0	z	z	Source Reg.			
CMP.m Rs,Rd	Dest. Reg.				0	1	1	0	1	1	z	z	Source Reg.			
AND.m Rs,Rd	Dest. Reg.				0	1	1	1	0	0	z	z	Source Reg.			
OR.m Rs,Rd	Dest. Reg.				0	1	1	1	0	1	z	z	Source Reg.			
ASR.m Rs,Rd	Dest. Reg.				0	1	1	1	1	0	z	z	Source Reg.			
LSR.m Rs,Rd	Dest. Reg.				0	1	1	1	1	1	z	z	Source Reg.			

Table 2.34: *Register instructions with variable size*

2.2.3.3 Summary of register instructions with fixed size

	Bit number/General instruction format															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Operation	Operand2				Mode		Opcode				Size		Operand1			
BTST Rs,Rd	Dest. Reg.				0	1	0	0	1	1	1	1	Source Reg.			
Scc Rd	Condition				0	1	0	1	0	0	1	1	Dest. Reg.			
RFE	0	0	1	0	1	0	0	1	0	0	1	1	0	0	0	0
SFE	0	0	1	1	1	0	0	1	0	0	1	1	0	0	0	0
RFG	0	1	0	0	1	0	0	1	0	0	1	1	0	0	0	0
RFN	0	1	0	1	1	0	0	1	0	0	1	1	0	0	0	0
HALT	1	1	1	1	1	0	0	1	0	0	1	1	0	0	0	0
ADDC Rs,Rd	Dest. Reg.				0	1	0	1	0	1	1	1	Source Reg.			
SETF <list>	P	U	I	X	0	1	0	1	1	0	1	1	N	Z	V	C
CLEARF <list>	P	U	I	X	0	1	0	1	1	1	1	1	N	Z	V	C
MOVE Rs,Pd	Special Reg.				0	1	1	0	0	0	1	1	Source Reg.			
MOVE Ps,Rd	Special Reg.				0	1	1	0	0	1	1	1	Dest. Reg. [1]			
ABS Rs,Rd	Dest. Reg.				0	1	1	0	1	0	1	1	Source Reg.			
DSTEP Rs,Rd	Dest. Reg.				0	1	1	0	1	1	1	1	Source Reg.			

[1] MOVE from the zero special registers (DZ,WZ and BZ) are used as CLEAR. The size of the clear depends on the special register used.

LZ Rs,Rd	Dest. Reg.				0	1	1	1	0	0	1	1	Source Reg.		
MOVE Rs,Sd	Support Reg.				1	0	1	1	0	1	1	1	Source Reg.		
MOVE Ss,Rd	Support Reg.				1	1	1	1	0	1	1	1	Dest. Reg.		
SWAP <opt> Rd	N	W	B	R	0	1	1	1	0	1	1	1	Dest. Reg.		
XOR Rs,Rd	Dest. Reg.				0	1	1	1	1	0	1	1	Source Reg.		
MCP Ps,Rd	Special Reg.				0	1	1	1	1	1	1	1	Dest. Reg.		

Table 2.35: *Register instructions with fixed size*

2.2.3.4 Summary of indirect instructions with variable size

m	Mode
0	Register indirect
1	Register indirect with post increment

Table 2.36: *Instruction modes*

z	Size	zz	Size
0	byte	00	byte
1	word	01	word
		10	dword

Table 2.37: *Instruction sizes*

	Bit number/General instruction format															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Operation	Operand2				Mode		Opcode				Size		Operand1			
ADDU.z [],Rd	Dest. Reg.				1	m	0	0	0	0	0	z	Source Reg.			
ADDS.z [],Rd	Dest. Reg.				1	m	0	0	0	0	1	z	Source Reg.			
MOVU.z [],Rd	Dest. Reg.				1	m	0	0	0	1	0	z	Source Reg.			
MOVS.z [],Rd	Dest. Reg.				1	m	0	0	0	1	1	z	Source Reg.			
SUBU.z [],Rd	Dest. Reg.				1	m	0	0	1	0	0	z	Source Reg.			
SUBS.z [],Rd	Dest. Reg.				1	m	0	0	1	0	1	z	Source Reg.			
CMPU.z [],Rd	Dest. Reg.				1	m	0	0	1	1	0	z	Source Reg.			
CMPS.z [],Rd	Dest. Reg.				1	m	0	0	1	1	1	z	Source Reg.			
ADDO.m [],Rd,ACR	Base				1	m	0	1	0	1	z	z	Source Reg.			
BOUND.m [],Rd	Index				1	m	0	1	1	1	z	z	Bound			
ADD.m [],Rd	Dest. Reg.				1	m	1	0	0	0	z	z	Source Reg.			
MOVE.m [],Rd	Dest. Reg.				1	m	1	0	0	1	z	z	Source Reg.			
SUB.m [],Rd	Dest. Reg.				1	m	1	0	1	0	z	z	Source Reg.			
CMP.m [],Rd	Dest. Reg.				1	m	1	0	1	1	z	z	Source Reg.			
AND.m [],Rd	Dest. Reg.				1	m	1	1	0	0	z	z	Source Reg.			
OR.m [],Rd	Dest. Reg.				1	m	1	1	0	1	z	z	Source Reg.			
TEST.m []	0	0	0	0	1	m	1	1	1	0	z	z	Source Reg.			
MOVE.m Rs,[]	Source Reg.				1	m	1	1	1	1	z	z	Dest.			

Table 2.38: *Indirect instructions with variable size*

2.2.3.5 Summary of indirect instructions with fixed size

m	Mode
0	Register Indirect
1	Register Indirect with post increment

Table 2.39: *Instruction modes*

	Bit number/General instruction format															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Operation	Operand2				Mode		Opcode				Size		Operand1			
FIDXI [Rs]	0	0	0	0	1	1	0	1	0	0	1	1	Source Reg.			
FTAGI [Rs]	0	0	0	1	1	1	0	1	0	0	1	1	Source Reg.			
LAPCQ qo,Rd	Dest. Reg.				1	0	0	1	0	1	1	1	Imm. Offset			
LAPC k,Rd	Dest. Reg.				1	1	0	1	0	1	1	1	1	1	1	1
Reserved	Dest. Reg.				1	m	0	1	1	0	0	0	Source Reg.			
Reserved	Dest. Reg.				1	m	0	1	1	0	0	1	Source Reg.			
ADDC [],Rd	Dest. Reg.				1	m	0	1	1	0	1	0	Source Reg.			
JAS Rs,Pd	Special Reg.				1	0	0	1	1	0	1	1	Source Reg.			
JAS aa,Pd	Special Reg.				1	1	0	1	1	0	1	1	1	1	1	1
JUMP Ps	Special Reg.				1	0	0	1	1	1	1	1	0	0	0	0
Bcc bo	Condition				1	1	0	1	1	1	1	1	1	1	1	1
MOVE [],Pd	Special Reg.				1	m	1	0	0	0	1	1	Source Reg.			
MOVE Ps,[]	Special Reg.				1	m	1	0	0	1	1	1	Dest. [2]			
FIDXD [Rs]	0	0	0	0	1	0	1	0	1	0	1	1	Source Reg.			
FTAGD [Rs]	0	0	0	1	1	0	1	0	1	0	1	1	Source Reg.			
BAS ao,Pd	Special Reg.				1	1	1	0	1	0	1	1	1	1	1	1
Reserved	Special Reg.				1	0	1	0	1	1	1	1	Source Reg.			
BASC ao,Pd	Special Reg.				1	1	1	0	1	1	1	1	1	1	1	1
JASC Rs,Pd	Special Reg.				1	0	1	1	0	0	1	1	Source Reg.			
JASC aa,Pd	Special Reg.				1	1	1	1	0	0	1	1	1	1	1	1
MOVEM [],Rd	Dest. Reg.				1	m	1	1	1	0	1	1	Source Reg.			
MOVEM Rs,[]	Source Reg.				1	m	1	1	1	1	1	1	Dest.			

Table 2.40: *Indirect instructions with fixed size*

2.3 Instructions in alphabetical order

2.3.1 Introduction

In this section, all the instructions of the CRIS CPU are described in alphabetical order. Each description contains the following information:

Assembler syntax: Shows the assembler syntax for the instruction. Operands, addressing modes and size modifiers are described using the definitions shown in section 2.2.1.1.

[2] MOVE from the zero special registers (DZ,WZ and BZ) are used as CLEAR. The size of the clear depends on the special register used.

Size: Lists the different data sizes for the instruction.

Operation: Describes the instruction in a form similar to the C programming language. Different data sizes are shown with the type cast method used in the C language. The behavior of the flags is usually not shown.

Description: A text description of the instruction.

Flags affected: Shows which flags are affected by the instruction. The detailed behavior of the flags is shown in section 2.1.3.

Instruction format: Shows the instruction formats.

2.3.2 ABS - Absolute Value

Assembler syntax: `ABS Rs,Rd`

Size: Dword

Description: The absolute value of the contents of the source register is stored in the destination register. The size of the operation is dword.

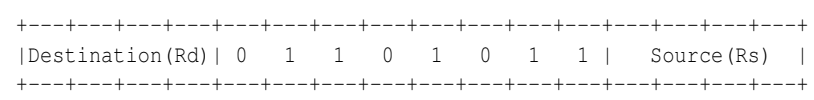
Operation:

```
if (Rs < 0) {
   Rd = -Rs;
} else {
   Rd = Rs;
}
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: `ABS Rs,Rd`



Note: If the source operand is 0x80000000, the result of the operation will be 0x80000000.

2.3.3 ADD - Add

Assembler syntax:

```
ADD.m Rs,Rd
ADD.m [Rs],Rd
ADD.m [Rs+],Rd
ADD.m k,Rd
```

Size: Byte, word, or dword

Description: The source data is added to the destination register. The size of the operation is m. The rest of the destination register is not affected.

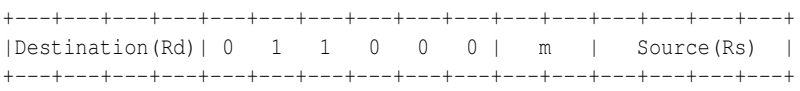
Operation:

```
(m)Rd += (m)s;
```

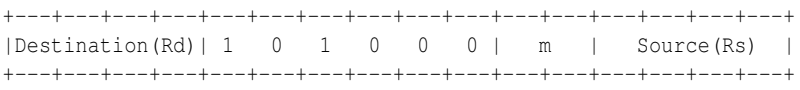
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

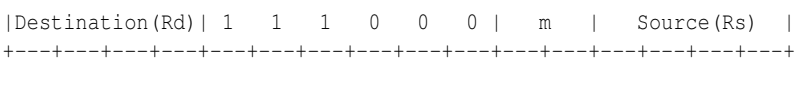
Instruction format: `ADD.m Rs,Rd`



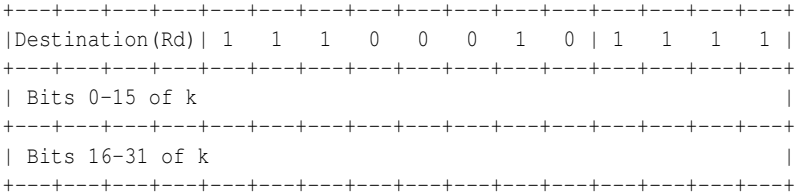
Instruction format: `ADD.m [Rs],Rd`



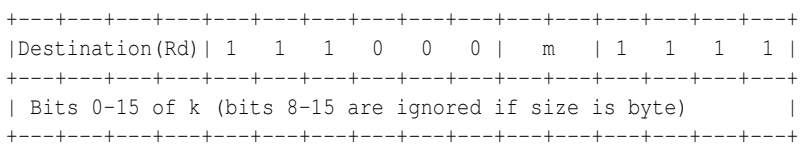
Instruction format: `ADD.m [Rs+],Rd`



Instruction format: `ADD.d k,Rd`



Instruction format: `ADD.b k,Rd` and `ADD.w k,Rd`



Mode (m)	Description
00	Byte
01	Word
10	Dword

2.3.4 ADDC - Add with Carry

Assembler syntax:

```
ADDC Rs,Rd
ADDC [Rs],Rd
ADDC [Rs+],Rd
ADDC k,Rd
```

Size: Dword

Description: The source data is added together with the carry flag to the destination register. The size of the operation is dword.

ADDC performs the same operation as the sequence:

```
   AX
   ADD.d s,Rd
```

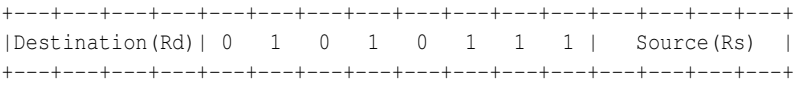
Operation:

```
Rd += s + C-flag;
```

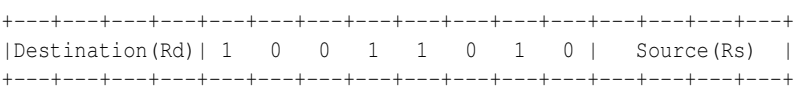
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

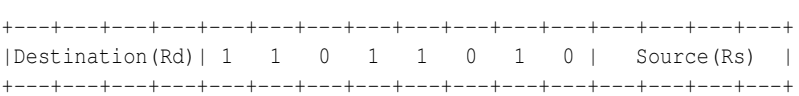
Instruction format: `ADDC Rs,Rd`



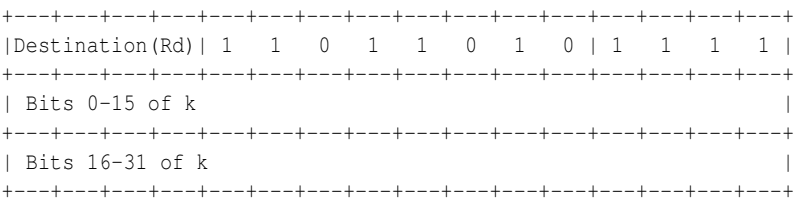
Instruction format: `ADDC [Rs],Rd`



Instruction format: `ADDC [Rs+],Rd`



Instruction format: `ADDC k,Rd`



2.3.5 ADDI - Add Index

Assembler syntax:

```
ADDI Rs.m,Rd
```

Size: Rs is a pointer to byte, word or dword. The size of the operation is dword.

Description: Add a scaled index to a base. The contents of the source register is shifted left 0, 1 or 2 positions, depending on the size modifier m, and then added to the destination register. The size of the operation is dword.

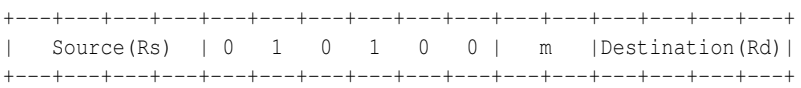
Operation:

```
Rd += Rs * sizeof(m);
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: `ADDI Rs.m,Rd`



Mode (m)	Description
00	Byte
01	Word
10	Dword

2.3.6 ADDI - Add Index (to ACR)

Assembler syntax:

```
ADDI Rs.m,Rd,ACR
```

Size: Rs is a pointer to byte, word or dword. The size of the operation is dword.

Description: Base plus index address calculation. ACR is assigned the sum of the contents of the destination register and the the shifted contents of the source operand. The shift will be 0, 1 or 2 steps depending on the specified size m. The operation will be performed on all 32 bits of the operands. The destination register is not updated.

- **Example:**

```
ADDI    R12.W,R1,ACR
ADD.d   R2,R4           ;; inserted to use latency
AND.w   [ACR],R5        ;; R5 &= [R1+R12*2]
```

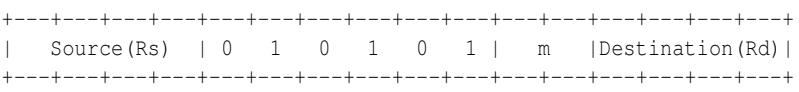
Operation:

```
ACR = Rd + Rs * sizeof(m);
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: ADDI Rs.m,Rd,ACR



Mode (m)	Description
00	Byte
01	Word
10	Dword

2.3.7 ADDO - Add Offset

Assembler syntax:

```
ADDO.m [Rs],Rd,ACR
ADDO.m [Rs+],Rd,ACR
ADDO.m k,Rd,ACR
```

Size: Source size is byte, word or dword. Operation size is dword.

Description: Base plus offset address calculation. ACR is assigned the sum of the source operand, sign extended to dword, and the contents of the destination register. The destination register is not updated.

· **Example:**

```
ADDO.W  1044,R10,ACR
ADD.D   R1,R2           ;; inserted to use latency
MOVE.D  [ACR],R1        ;; [1044 + R10] -> R1

ADDO.D  [R1],R2,ACR
SUBQ    4,R1
ADD.D   [ACR],R3        ;; R3 += [ [R1--] + R2 ]
```

The main purpose with this instruction is to offer a wider range of immediate offsets than the ADDQ o,Rs,ACR instruction does.

Operation:

```
ACR = Rd + (m)s;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

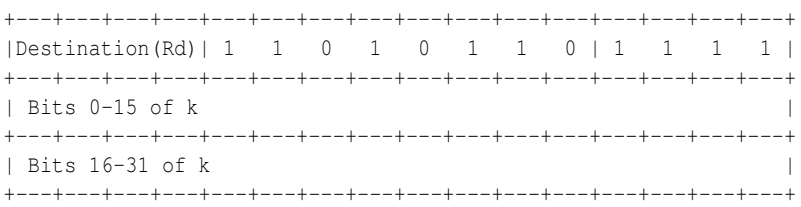
Instruction format: ADDO.m [Rs],Rd,ACR

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   0   0   1   0   1 |   m   |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

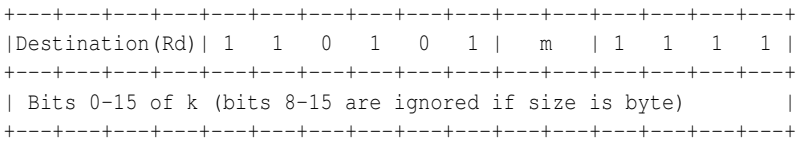
Instruction format: ADDO.m [Rs+],Rd,ACR

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   1   0   1   0   1 |   m   |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `ADDO.d k,Rd,ACR`



Instruction format: `ADDO.b k,Rd,ACR and ADDO.w k,Rd,ACR`



Mode (m)	Description
00	Byte
01	Word
10	Dword

2.3.8 ADDOQ - Add Offset Quick

Assembler syntax:

```
ADDOQ o,Rs,ACR
```

Size: Source data is 8-bit. The size of the operation is dword.

Description: Base plus offset address calculation. ACR is assigned the sum of the contents of the source register and an 8-bit signed offset o.

- **Example:**

```
ADDOQ -8,R7,ACR
MOVE.D R3,[ACR]
```

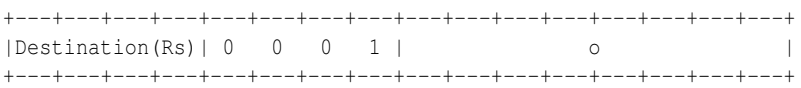
Operation:

```
ACR = Rs + o;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: ADDOQ o,Rs,ACR



2.3.9 ADDQ - Add Quick

Assembler syntax:

```
ADDQ j,Rd
```

Size: Source data is 6-bit. The size of the operation is dword.

Description: A 6-bit immediate value, zero extended to dword, is added to the destination register.

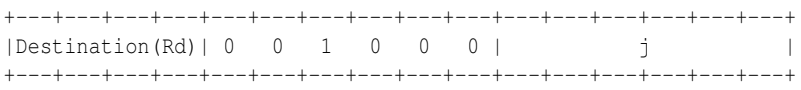
Operation:

```
Rd += j;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

Instruction format: `ADDQ j,Rd`



2.3.10 ADDS - Add with Sign Extend

Assembler syntax:

```
ADDS.z Rs,Rd
ADDS.z [Rs],Rd
ADDS.z [Rs+],Rd
ADDS.z k,Rd
```

Size: Source size is byte or word. Operation size is dword

Description: The source data is sign extended from z to dword, and then added to the destination register.

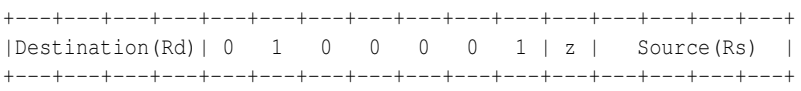
Operation:

```
Rd += (z)s;
```

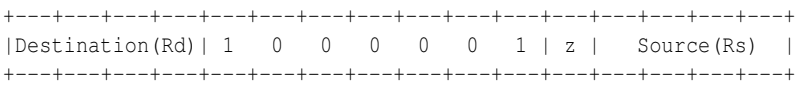
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

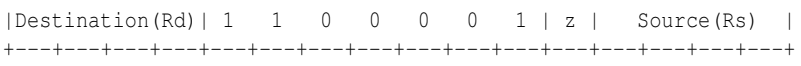
Instruction format: `ADDS.z Rs,Rd`



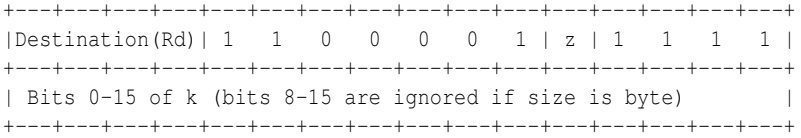
Instruction format: `ADDS.z [Rs],Rd`



Instruction format: `ADDS.z [Rs+],Rd`



Instruction format: `ADDS.z k,Rd`



Size (z)	Description
0	Byte source operand
1	Word source operand

2.3.11 ADDU - Add with Zero Extend

Assembler syntax:

```
ADDU.z Rs,Rd
ADDU.z [Rs],Rd
ADDU.z [Rs+],Rd
ADDU.z k,Rd
```

Size: Source size is byte or word. Operation size is dword.

Description: The source data is zero extended from z to dword, and then added to the destination register.

Operation:

```
Rd += (unsigned z)s;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

Instruction format: `ADDU.z Rs,Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 0   1   0   0   0   0   0 | z |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `ADDU.z [Rs],Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   0   0   0   0   0   0 | z |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `ADDU.z [Rs+],Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   1   0   0   0   0   0 | z |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `ADDU.z k,Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   1   0   0   0   0   0 | z | 1   1   1   1 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Bits 0-15 of k (bits 8-15 are ignored if size is byte)        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Size (z)	Description
0	Byte source operand
1	Word source operand

2.3.12 AND - Logical AND

Assembler syntax:

```
AND.m Rs,Rd
AND.m [Rs],Rd
AND.m [Rs+],Rd
AND.m k,Rd
```

Size: Byte, word, or dword

Description: A logical AND is performed between the source operand and the destination register. The size of the operation is m. The rest of the destination register is not affected.

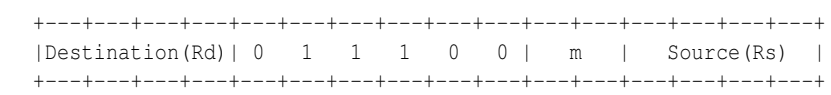
Operation:

```
(m)Rd &= (m)s;
```

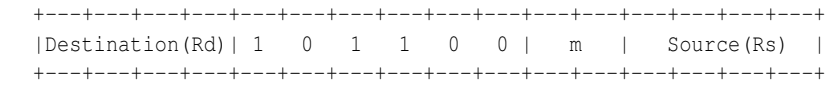
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

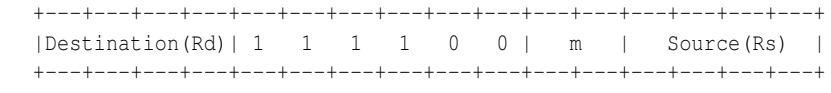
Instruction format: `AND.m Rs,Rd`



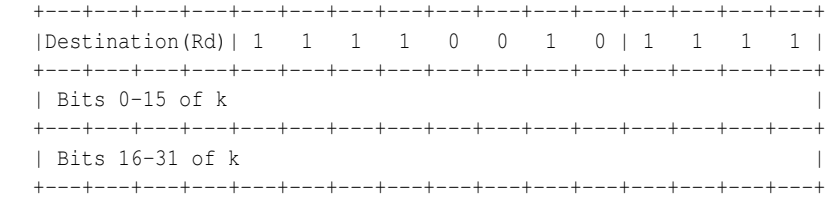
Instruction format: `AND.m [Rs],Rd`



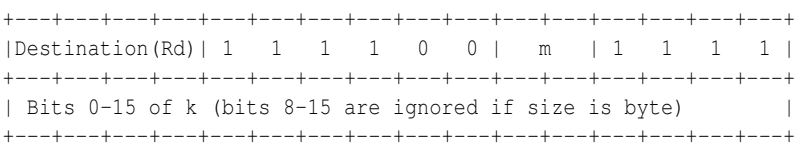
Instruction format: `AND.m [Rs+],Rd`



Instruction format: `AND.d k,Rd`



Instruction format: `AND.b k,Rd and AND.w k,Rd`



Mode (m)	Description
00	Byte
01	Word
10	Dword

2.3.13 ANDQ - Logical AND Quick

Assembler syntax:

```
ANDQ i,Rd
```

Size: Source data is 6-bit. Operation size is dword.

Description: A logical AND is performed between a 6-bit immediate value, sign extended to dword, and the destination register.

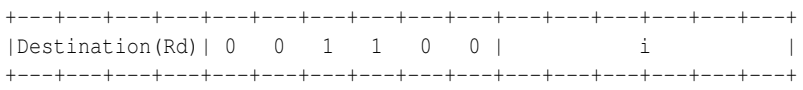
Operation:

```
Rd &= i;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: ANDQ i,Rd



2.3.14 ASR - Arithmetic Shift Right

Assembler syntax:

```
ASR.m Rs,Rd
```

Size: Byte, word or dword

Description: The destination register is right shifted the number of steps specified by the 6 least significant bits of the source register. The shift is performed with sign extend. The size of the operation is m. The rest of the destination register is not affected.

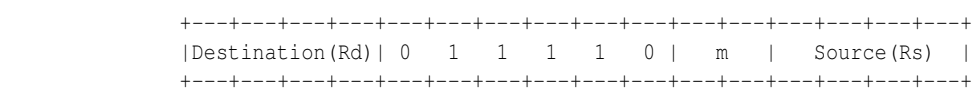
Operation:

```
(m)Rd >>= (Rs & 63);
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: `ASR.m Rs,Rd`



Mode (m)	Description
00	Byte
01	Word
10	Dword

2.3.15 ASRQ - Arithmetic Shift Right Quick

Assembler syntax:

```
ASRQ c,Rd
```

Size: Dword

Description: The destination register is right shifted the number of steps specified by the 5-bit immediate value. The shift is performed with sign extend. The size of the operation is dword.

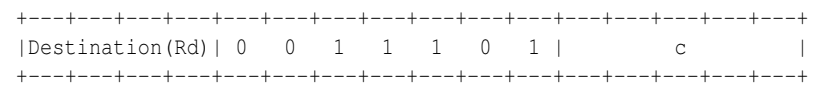
Operation:

```
Rd >>= c;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: ASRQ c,Rd



2.3.16 AX - Arithmetic Extension

Assembler syntax:

```
AX
```

Size: NA

Description: AX is the arithmetic extension prefix and an alias for SETF X. Set X flag. AX is also used to indicate that the following write operation is conditional.

2.3.17 BA - Branch Always

Assembler syntax:

```
BA ao
```

Size: Dword

Description: Dword sized BA ao is an alias for BAS ao,P0. For word and byte sized versions see section 2.3.20 below.

2.3.18 BAS - Branch And Save

Assembler syntax:

```
BAS ao,Pd
```

Size: Dword

Description: Jumps to a PC relative address.

Special register Pd is loaded with the address of the instruction following BAS plus two. The instruction following the BAS is executed before the jump takes effect (i.e. the BAS instruction has one delay slot). The content of the dword sized operand is then added to PC. The value of PC used for the address calculation is the address of the BAS instruction itself.

Legal instructions for the delay slot are all instructions except:

- Bcc
- JAS,BAS,JASC,BASC,JUMP,HALT,FIDXI,FTAGI
- Immediate Addressing other than Quick Immediate

The BAS instruction is used for both regular and subroutine branches. Regular branches are made by using BZ (P0) as the destination special register. Branches to subroutines are made by using SRP as the destination special register. Other destination special registers may be used as follows:

Special Register	Use
VR, WZ, DZ	Same behavior as BZ.
PID	Undefined behavior, should not be used.
SRS	Only eight bits, should not be used.
EXS, EDA, ERP	May be used when no exceptions are expected.
NRP	May be used when no NMI exceptions are expected.
EBP	May be used when no exceptions or NMI are used.
CCS	Not useful.
USP	May be used when USP is not used as a User Mode stack pointer.
MOF	May be used.
SPC	Should not be used as it may break debugging via the guru mode.

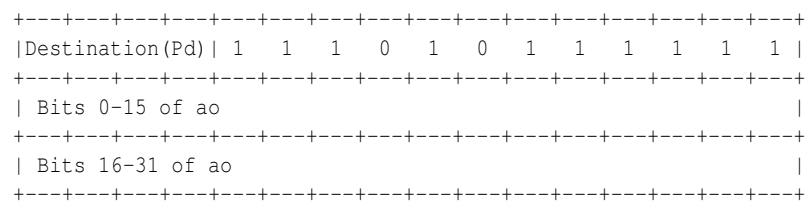
Operation:

```
Pd = PC + 8;
PC += ao;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: `BAS ao,Pd`



2.3.19 BASC - Branch And Save with Context Information

Assembler syntax:

```
BASC ao,Pd
```

Size: Dword

Description: Jumps to a PC relative address.

BASC is the same as BAS except that Pd is loaded with the address of the instruction following the BASC instruction plus 6. This leaves a 32 bit slot for storing context information after the delay slot.

```
addr a   : [ BASC aa,Pd    ]
     a+2 : [ jump target.. ]
     a+4 : [ address       ]
     a+6 : [ op or nop     ]
     a+8 : [ space for..   ]
     a+10: [ context info  ]
     a+12: [ op            ] <- addr in Pd
```

The value of PC used for the address calculation is the address of the branch instruction itself.

The BASC instruction is a delayed branch instruction, with one delay slot. Legal instructions for the delay slot are all instructions except:

- Bcc
- JAS,BAS,JASC,BASC,JUMP,HALT,FIDXI,FTAGI
- Immediate Addressing other than Quick Immediate

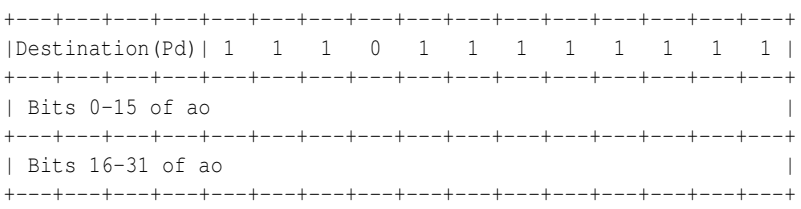
Operation:

```
Pd = PC + 12;
PC = ao;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: `BASC ao,Pd`



2.3.20 Bcc - Branch Conditionally

Assembler syntax:

```
Bcc o
Bcc bo
```

Size: Byte or word

Description: Conditionally jumps to a PC relative address. If the condition cc is true, the offset is added to PC.

In the byte version, bit 0 of the offset is the sign bit, and bits (7..1) in the offset represent bits (7..1) in the actual address increment/decrement. In this way the branch range will be (-256..+254 bytes). When bit 0 == 1 the branch is backwards.

In the word version, the immediate offset is added to the current PC to get the target address. Bit 0 in the word sized offset shall always be zero.

There is also a 32 bit version of BA (which is an alias for BAS ao,P0). See section 2.2.2.7 for details.

The instruction following the Bcc is executed before the jump takes effect (i.e. the Bcc instruction has one delay slot). Legal instructions for the delay slot are all instructions except:

- Bcc
- JAS,BAS,JASC,BASC,JUMP,HALT,FIDXI,FTAGI
- Immediate Addressing other than Quick Immediate

The value of PC used for the address calculation is the address of the branch instruction itself.

Operation: Bcc o :

```
if (cc) {
   PC += o;
}
```

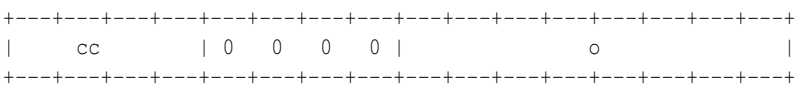
Bcc bo :

```
if (cc) {
   PC += bo;
}
```

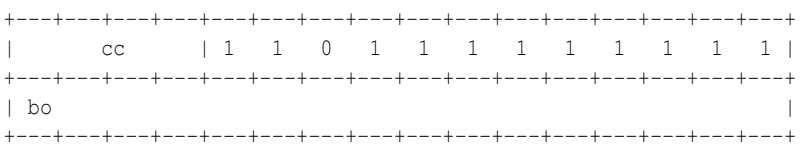
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: `Bcc o`



Instruction format: `Bcc bo`



2.3.21 BOUND - Adjust Index to Bound

Assembler syntax:

```
BOUND.m Rs,Rd
BOUND.m k,Rd
```

Size: Source is byte, word or dword. Operation is dword.

Description: BOUND is a bounding instruction. For example, it adjusts branch indexes in switch statements. If the unsigned contents of the dword index (destination) register is greater than the bound (source) data, the bound data (zero extended to dword) is loaded to the index (destination) register. Otherwise, the index (destination) register is unaffected.

This operation can also be used to calculate the unsigned minimum of two operands. The destination is loaded with the minimum of the source and destination operands.

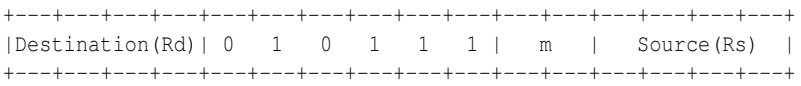
Operation:

```
if ((unsigned)Rd > (unsigned m)s) {
   Rd = (unsigned m)s;
}
```

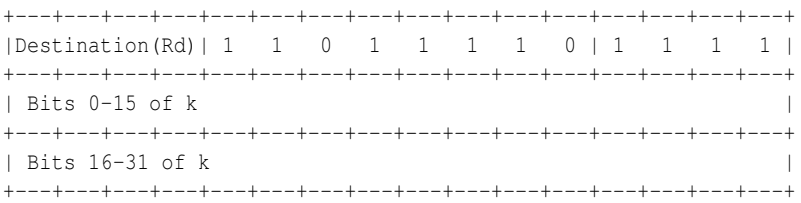
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

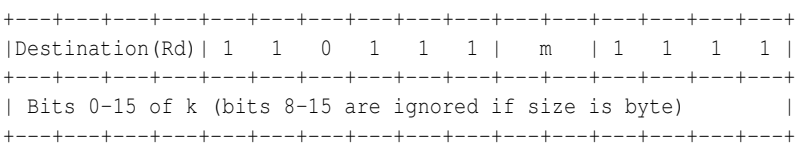
Instruction format: BOUND.m Rs,Rd



Instruction format: BOUND.d k,Rd



Instruction format: BOUND.b k,Rd and BOUND.w k,Rd



Mode (m)	Description
00	Byte
01	Word
10	Dword

2.3.22 BREAK - Software Breakpoint

Assembler syntax:

```
BREAK n
```

Size: NA

Description: This instruction causes a software exception. The operand n selects which vector index to use.

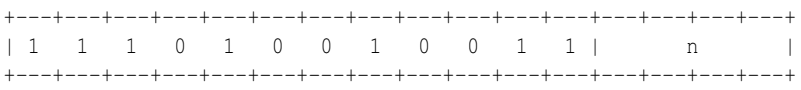
Operation:

```
Cause exception using vector n+0x10
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: `BREAK n`



2.3.23 BSR - Branch to Subroutine

Assembler syntax:

```
BSR ao
```

Size: Dword

Description: BSR ao is an alias for BAS ao,SRP.

2.3.24 BSRC - Branch to Subroutine with Context Information

Assembler syntax:

```
BSRC ao
```

Size: Dword

Description: BSRC ao is an alias for BASC ao,SRP.

2.3.25 BTST - Bit Test

Assembler syntax:

```
BTST Rs,Rd
```

Size: Dword

Description: The N flag is set according to the selected bit in the destination register. The Z flag is set if the selected bit and all bits to the right of it are zero. The bit number is selected by the 5 least significant bits of the source register. The destination register is not affected.

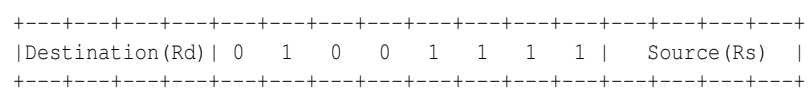
Operation:

```
N = Bit number (Rs & 31) of Rd;
Z = ((Bit numbers 0 to (Rs & 31) of Rd) == 0);
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: `BTST Rs,Rd`



2.3.26 BTSTQ - Bit Test Quick

Assembler syntax:

```
BTSTQ c,Rd
```

Size: Dword

Description: The N flag is set according to the selected bit in the destination register. The Z flag is set if the selected bit and all bits to the right of it are zero. The bit number is selected by the 5-bit immediate value. The destination register is not affected.

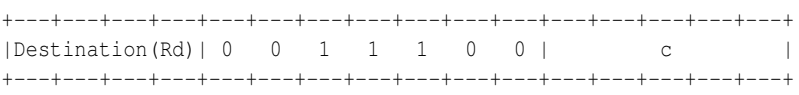
Operation:

```
N = Bit number c of Rd;
Z = ((Bit numbers 0 to c of Rd) == 0);
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: `BTSTQ c,Rd`



2.3.27 CLEAR - Clear

Assembler syntax:

```
CLEAR.m Rd
CLEAR.m [Rd]
CLEAR.m [Rd+]
```

Size: Byte, word, or dword

Description: CLEAR.m d is an alias for MOVE Ps,d where Ps = P0,P4 or P8. The destination is cleared to all zeroes. The size of the operation is m.

Note The clear instruction is implemented as a MOVE from a special register destination. The size depends on the selected number.

2.3.28 CLEARF - Clear Flags

Assembler syntax:

```
CLEARF <list of flags>
```

Size: NA

Description: The specified flags are set to 0. The X flag is cleared even if it is not in the list. The U and I flags can not be changed when in User mode (U==1).

Operation:

```
Selected flags = 0;
/* U and I flags are not affected in User mode */
X = 0;
```

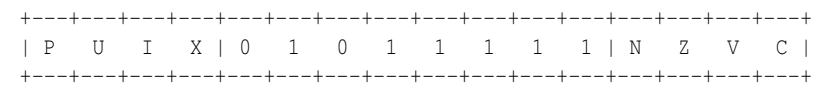
Flags affected: (Kernel mode)

```
S R P U I X N Z V C
- - * * * 0 * * * *
```

Flags affected: (User mode)

```
S R P U I X N Z V C
- - * - - 0 * * * *
```

Instruction format: CLEARF <list of flags>



2.3.29 CMP - Compare

Assembler syntax:

```
CMP.m Rs,Rd
CMP.m [Rs],Rd
CMP.m [Rs+],Rd
CMP.m k,Rd
```

Size: Byte, word, or dword

Description: The source data is subtracted from the destination register, and the flags are set accordingly. The size of the operation is m. The destination register is not updated.

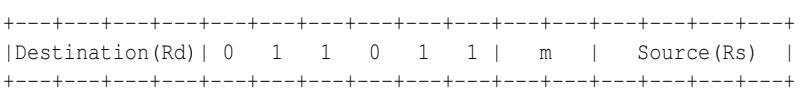
Operation:

```
(m)Rd - (m)s;
```

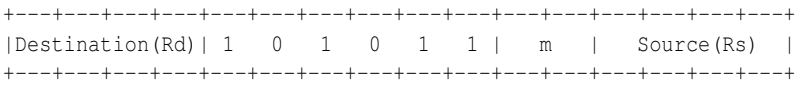
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

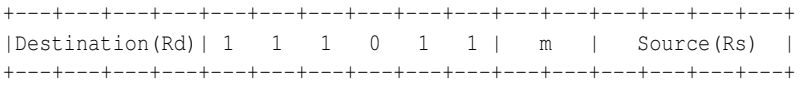
Instruction format: `CMP.m Rs,Rd`



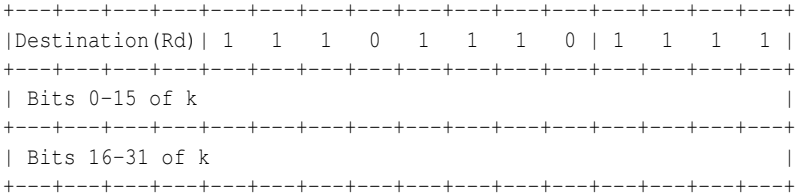
Instruction format: `CMP.m [Rs],Rd`



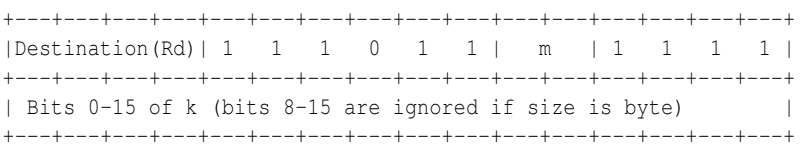
Instruction format: `CMP.m [Rs+],Rd`



Instruction format: `CMP.d k,Rd`



Instruction format: `CMP.b k,Rd` and `CMP.w k,Rd`



Mode (m)	Description
00	Byte
01	Word
10	Dword

2.3.30 CMPQ - Compare Quick

Assembler syntax:

```
CMPQ i,Rd
```

Size: Dword

Description: A 6-bit immediate value, sign extended to dword, is subtracted from the destination register, and the flags are set accordingly. The destination register is not updated.

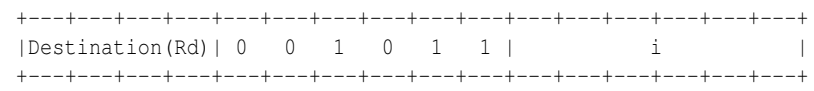
Operation:

```
Rd - i;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

Instruction format: `CMPQ i,Rd`



2.3.31 CMPS - Compare with Sign Extend

Assembler syntax:

```
CMPS.z [Rs],Rd
CMPS.z [Rs+],Rd
CMPS.z k,Rd
```

Size: Source size is byte or word. Operation size is dword.

Description: The source data, sign extended to dword, is subtracted from the destination register, and the flags are set accordingly. The destination register is not updated.

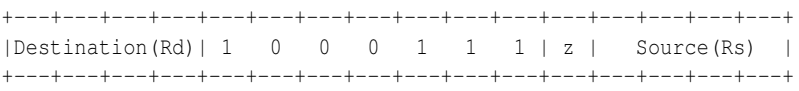
Operation:

```
Rd - (z)si;
```

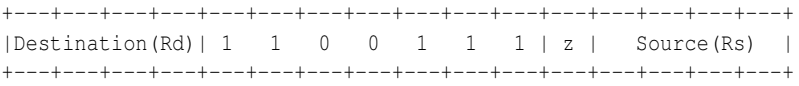
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

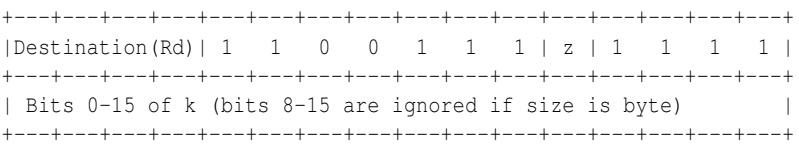
Instruction format: `CMPS.z [Rs],Rd`



Instruction format: `CMPS.z [Rs+],Rd`



Instruction format: `CMPS.z k,Rd`



Size (z)	Description
0	Byte source operand
1	Word source operand

2.3.32 CMPU - Compare with Zero Extend

Assembler syntax:

```
CMPU.z [Rs],Rd
CMPU.z [Rs+],Rd
CMPU.z k,Rd
```

Size: Source size is byte or word. Operation size is dword.

Description: The source data, zero extended to dword, is subtracted from the destination register, and the flags are set accordingly. The destination register is not updated.

Operation:

```
Rd - (unsigned z)si;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

Instruction format: `CMPU.z [Rs],Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   0   0   0   1   1   0 | z |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `CMPU.z [Rs+],Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   1   0   0   1   1   0 | z |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `CMPU.z k,Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   1   0   0   1   1   0 | z | 1   1   1   1 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Bits 0-15 of k (bits 8-15 are ignored if size is byte)        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Size (z)	Description
0	Byte source operand
1	Word source operand

2.3.33 DI - Disable Interrupts

Assembler syntax:

```
DI
```

Size: N/A

Description: DI is an alias for CLEARF I. Disables interrupts. DI has no effect in User Mode.

2.3.34 DSTEP - Divide Step

Assembler syntax:

```
DSTEP Rs,Rd
```

Size: Dword

Description: This is a divide-step operation, which performs one iteration of an iterative divide operation. The destination operand is shifted one step to the left. If the shifted destination operand is unsigned greater or equal to the source operand, the source operand is subtracted from the shifted destination operand. The size of the operation is dword.

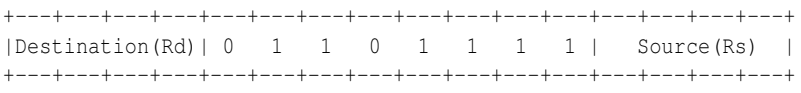
Operation:

```
Rd <<= 1;
if ((unsigned)Rd >= (unsigned)Rs) {
   Rd -= Rs;
}
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: DSTEP Rs,Rd



2.3.35 EI - Enable Interrupt

Assembler syntax:

```
EI
```

Size: N/A

Description: EI is an alias for SETF I. Enable interrupts. EI has no effect in User mode.

2.3.36 FIDXD - Flush Data Cache Line by Index

Assembler syntax:

```
FIDXD [Rs]
```

Size: N/A

Description: Flushes (invalidates and writes if dirty) the cache line with the specified index from the data cache. The cache line is specified as:

```
Rs[20:5] = cache line
Rs[4:0]  = cache bank
```

The number of bits actually used for cache line and cache bank depends on the (fixed) data cache size. All unused bits of Rs should be set to zero.

This instruction is only available in kernel mode and may be used to initialize the data cache at system start-up. FIDXD is silently ignored in user mode.

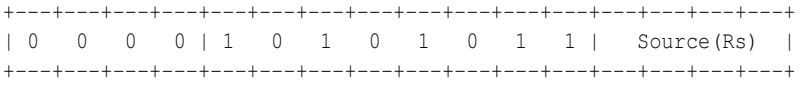
Operation:

```
Flush data cache line Rs[20:5] in bank Rs[4:0]
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: FIDXD [Rs]



2.3.37 FIDXI - Flush Instruction Cache Line by Index

Assembler syntax:

```
FIDXI [Rs]
```

Size: N/A

Description: Flushes (invalidates) the cache line with the specified index from the instruction cache. The cache line is specified as:

```
Rs[20:5] = cache line
Rs[4:1]  = cache bank
```

The number of bits actually used for cache line and cache bank depends on the (fixed) instruction cache size. All unused bits of Rs should be set to zero.

This instruction is only available in kernel mode and may be used to initialize the instruction cache at system start-up. FIDXI is silently ignored in user mode.

Operation:

```
Flush instruction cache line Rs[20:5] in bank Rs[4:1]
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: `FIDXI [Rs]`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 0   0   0   0 | 1   1   0   1   0   0   1   1 |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

2.3.38 FTAGD - Flush Data Cache Line by Address

Assembler syntax:

```
FTAGD [Rs]
```

Size: N/A

Description: Flushes (invalidates and writes if dirty) the cache line with the specified address from the data cache. If the specified address is not cached nothing is done. Bit 31 and bits 0-4 in the specified address are ignored.

This instruction is only available in kernel mode. It is silently ignored in User mode.

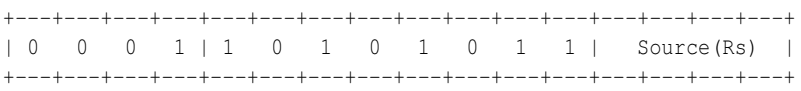
Operation:

```
Flush data cache line containing [Rs], if any
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: `FTAGD [Rs]`



2.3.39 FTAGI - Flush Instruction Cache Line by Address

Assembler syntax:

```
FTAGI [Rs]
```

Size: N/A

Description: Flushes (invalidates) the cache line with the specified address from the instruction cache. If the specified address isn't cached nothing is done. Bit 31 and bits 0-4 in the specified address are ignored.

This instruction is only available in kernel mode. It is silently ignored in User mode.

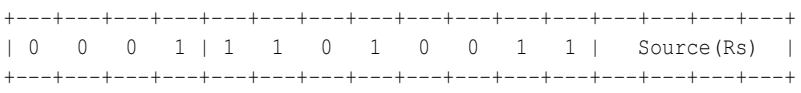
Operation:

```
Flush instruction cache line containing [Rs], if any
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: `FTAGI [Rs]`



2.3.40 HALT - Stop and Wait for Exceptions

Assembler syntax:

```
HALT
```

Size: N/A

Description: Stop execution and wait for exceptions. Continue execution with the instruction following the HALT after an exception has been served. This instruction is only allowed in kernel mode. In user mode it is ignored. HALT instructions may not be in a delay slot.

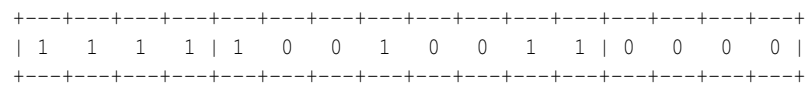
Operation:

```
while(!exceptions);
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: HALT



2.3.41 JAS - Jump and Save

Assembler syntax:

```
JAS Rs,Pd
JAS aa,Pd
```

Size: Dword

Description: Jumps to an absolute address.

Special register Pd is loaded with the address of the instruction following JAS plus two. The instruction following the JAS is executed before the jump takes effect (i.e. the JAS instruction has one delay slot). PC is then loaded with the content of the dword sized source operand.

Legal instructions for the delay slot are all instructions except:

- Bcc
- JAS,BAS,JASC,BASC,JUMP,HALT,FIDXI,FTAGI
- Immediate Addressing other than Quick Immediate

Modifying Rs in the delay slot of JAS Rs,Pd is harmless and does not affect the target address of the jump.

The JAS instruction is used for both regular jumps and subroutine jumps. Regular jumps are made by using BZ (P0) as the destination special register. Jumps to subroutines are made by using SRP as the destination special register. Other destination special registers may be used as follows:

Special register	Use
VR, WZ, DZ	Same behavior as BZ.
PID	Undefined behavior, should not be used.
SRS	Only eight bits, should not be used.
EXS, EDA, ERP	May be used when no exceptions are expected.
NRP	May be used when no NMI exceptions are expected.
EBP	May be used when no exceptions or NMI are used.
CCS	Not useful.
USP	May be used when USP is not used as a user mode stack pointer.
MOF	May be used.
SPC	Should not be used as it may break debugging via the guru mode.

Operation: JAS Rs,Pd :

```
Pd = PC + 4;
PC = Rs;
```

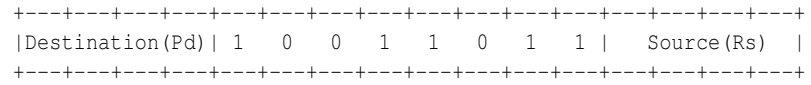
JAS aa,Pd :

```
Pd = PC + 8;
PC = aa;
```

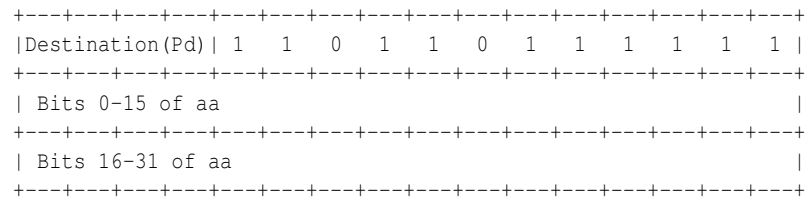
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: JAS Rs,Pd



Instruction format: JAS aa,Pd



2.3.42 JASC - Jump and Save with Context Information

Assembler syntax:

```
JASC Rs,Pd
JASC aa,Pd
```

Size: Dword

Description: Jumps to an absolute address.

JASC similar to JAS that Pd is loaded with the address of the instruction following the JAS plus 6. This leaves a 32 bit slot for storing context information after the delay slot.

```
addr a   : [ JASC Rs,Pd   ]
     a+2 : [ op or nop    ]
     a+4 : [ space for..  ]
     a+6 : [ context info ]
     a+8 : [ op           ] <- addr in Pd

addr a   : [ JASC aa,Pd    ]
     a+2 : [ jump target.. ]
     a+4 : [ address       ]
     a+6 : [ op or nop     ]
     a+8 : [ space for..   ]
     a+10: [ context info  ]
     a+12: [ op            ] <- addr in Pd
```

Operation: JASC Rs,Pd :

```
Pd = PC + 8;
PC = Rs;
```

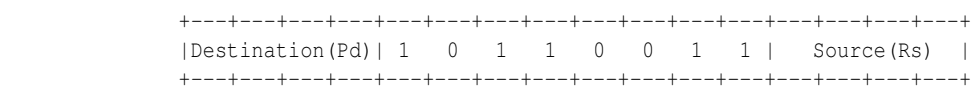
JASC aa,Pd :

```
Pd = PC + 12;
PC = aa;
```

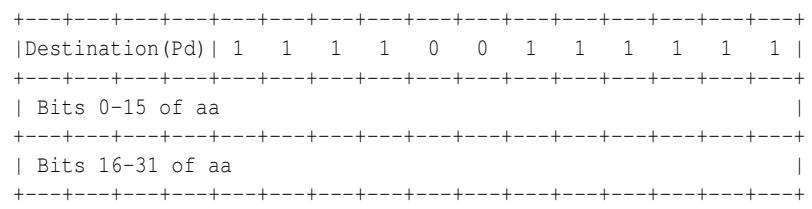
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: JASC Rs,Pd



Instruction format: `JASC aa,Pd`



2.3.43 JSR - Jump to Subroutine

Assembler syntax:

```
JSR aa
JSR Rs
```

Size: Dword

Description: JSR is an alias for JAS aa,SRP. JSR Rs is an alias for JAS Rs,SRP.

2.3.44 JSRC - Jump to Subroutine with Context Information

Assembler syntax:

```
JSRC aa
JSRC Rs
```

Size: Dword

Description: JSRC aa is an alias for JASC aa,SRP. JSRC Rs is an alias for JASC Rs,SRP.

2.3.45 JUMP - Jump to Absolute Address

Assembler syntax:

```
JUMP aa
JUMP Rs
```

Size: Dword

Description: JUMP aa an alias for JAS aa,P0. JUMP Rs is an alias for JAS Rs,P0.

2.3.46 JUMP - Jump to Special Register

Assembler syntax:

```
JUMP Ps
```

Size: Dword

Description: Jump to special register. The program counter is loaded with the value of the dword sized source operand. The instruction following the JUMP Ps is executed before the jump takes effect (i.e. the JUMP Ps instruction has one delay slot).

The JUMP Ps instruction is used for returning from subroutines and exceptions. JUMP SRP returns from a subroutine, JUMP ERP returns from an exception and JUMP NRP returns from an NMI exception. Observe that JUMP ERP normally should have an RFE instruction in the delay slot, and JUMP NRP should have an RFN instruction in the delay slot.

Legal instructions for the delay slot are all instructions except:

- Bcc
- JAS,BAS,JASC,BASC,JUMP,HALT,FIDXI,FTAGI
- Immediate addressing other than Quick Immediate

Modifying Ps in the delay slot of JUMP Ps is harmless and does not affect the target address of the jump.

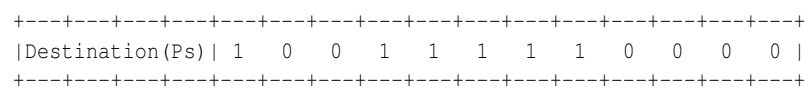
Operation:

```
PC = Ps;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: JUMP Ps



2.3.47 LAPC - Load PC Relative Address

Assembler syntax:

```
LAPC address,Rd
```

Size: Dword

Description: Assigns Rd the sum of the PC and the signed dword k in the immediate operand of the instruction. The assembler operand syntax describes an absolute address, the resulting PC + k.

This instruction is mainly used for PC relative addressing of data and code. Can also be used to get the current value of PC.

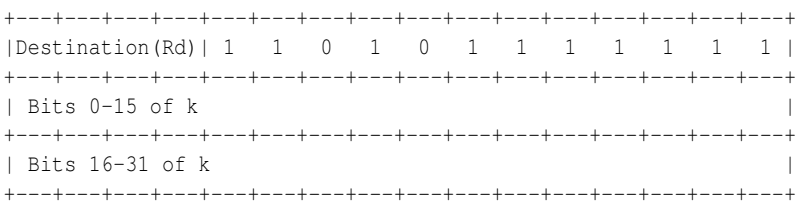
Operation:

```
Rd = PC + k;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: LAPC address,Rd



2.3.48 LAPCQ - Load PC Relative Address Quick

Assembler syntax:

```
LAPCQ address,Rd
```

Size: N/A

Description: Assigns Rd the sum of the PC and unsigned 4-bit immediate constant qo in the instruction source field, multiplied by two. The assembler operand syntax describes an absolute address, the resulting PC + 2*qo.

This instruction is mainly used for PC relative addressing of data and code. Can also be used to get the current value of PC.

· **Example:**

```
  LAPCQ .,Rd ;; Rd = PC
```

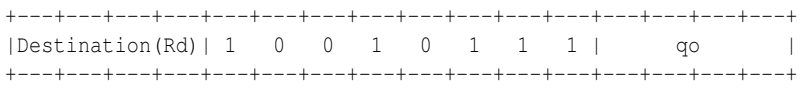
Operation:

```
Rd = PC + 2*qo;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: LAPCQ address,Rd



2.3.49 LSL - Logical Shift Left

Assembler syntax:

```
LSL.m Rs,Rd
```

Size: Byte, word, or dword

Description: The destination register is left shifted the number of steps specified by the 6 least significant bits of the source register. The size of the operation is m. The rest of the destination register is not affected.

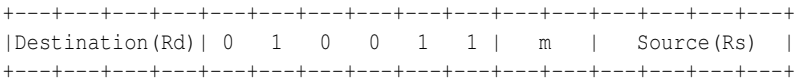
Operation:

```
(m)Rd <<= (Rs & 63);
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: `LSL.m Rs,Rd`



Mode (m)	Description
00	Byte
01	Word
10	Dword

2.3.50 LSLQ - Logical Shift Left Quick

Assembler syntax:

```
LSLQ c,Rd
```

Size: Dword

Description: The destination register is left shifted the number of steps specified by the 5-bit immediate value. The size of the operation is dword.

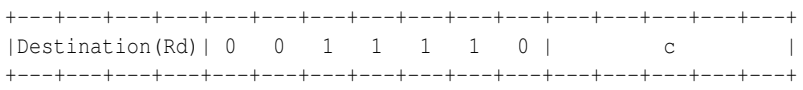
Operation:

```
Rd <<= c;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: LSLQ c,Rd



2.3.51 LSR - Logical Shift Right

Assembler syntax:

```
LSR.m Rs,Rd
```

Size: Byte, word or dword

Description: The destination register is right shifted the number of steps specified by the 6 least significant bits of the source register. The shift is performed with zero extend. The size of the operation is m. The rest of the destination register is not affected.

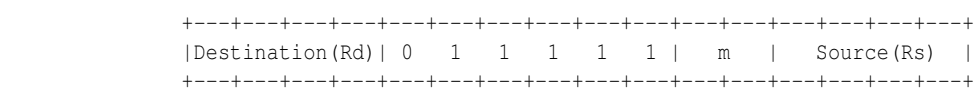
Operation:

```
(unsigned m)Rd >>= (Rs & 63);
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: `LSR.m Rs,Rd`



Mode (m)	Description
00	Byte
01	Word
10	Dword

2.3.52 LSRQ - Logical Shift Right Quick

Assembler syntax:

```
LSRQ c,Rd
```

Size: Dword

Description: The destination register is right shifted the number of steps specified by the 5-bit immediate value. The shift is performed with zero extend. The size of the operation is dword.

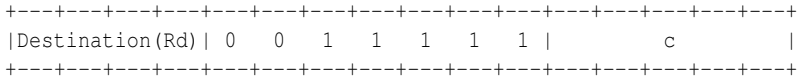
Operation:

```
(unsigned)Rd >>= c;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: LSRQ c,Rd



2.3.53 LZ - Leading Zeros

Assembler syntax:

```
LZ Rs,Rd
```

Size: Dword

Description: The destination register is loaded with the number of leading zeroes of the contents of the source register. The size of the operation is dword.

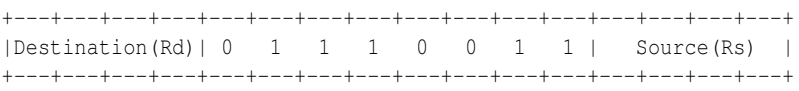
Operation:

```
Rd = 32;
while (((unsigned)Rs >> (32 - Rd)) != 0) {
   Rd--;
}
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 0 * - -
```

Instruction format: `LZ Rs,Rd`



2.3.54 MCP - Multiply Carry Propagation

Assembler syntax:

```
MCP Ps,Rd
```

Size:

Description: This operation is used during iterative large number multiply operation. It adds Ps to Rd with carry and uses the R-flag as both carry in and carry out.

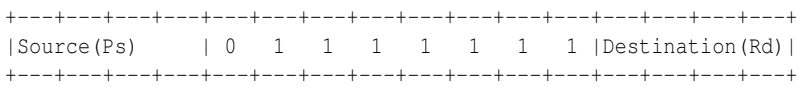
Operation:

```
{R-flag,Rd} = Rd + Ps + R-flag;
```

Flags affected:

```
S R P U I X N Z V C
- * - - - 0 * * * -
```

Instruction format: `MCP Ps,Rd`



2.3.55 MOVE - Move to General Register

Assembler syntax:

```
MOVE.m Rs,Rd
MOVE.m [Rs],Rd
MOVE.m [Rs+],Rd
MOVE.m k,Rd
```

Size: Byte, word or dword

Description: Move data from source to the destination register. The size of the operation is m. The rest of the destination register is not affected.

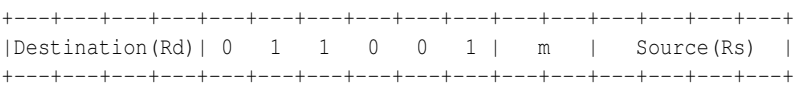
Operation:

```
(m)Rd = (m)s;
```

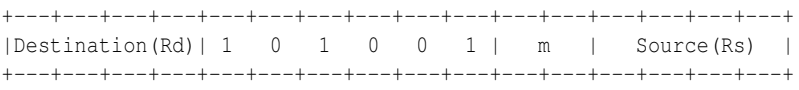
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

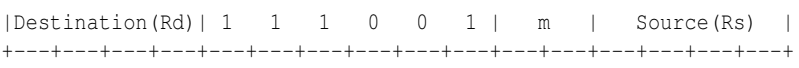
Instruction format: `MOVE.m Rs,Rd`



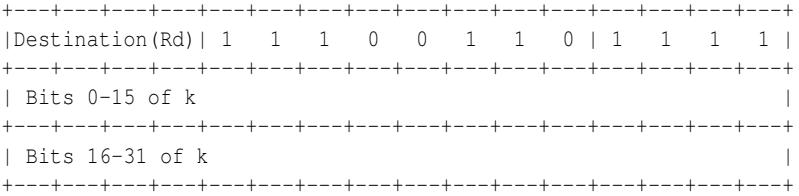
Instruction format: `MOVE.m [Rs],Rd`



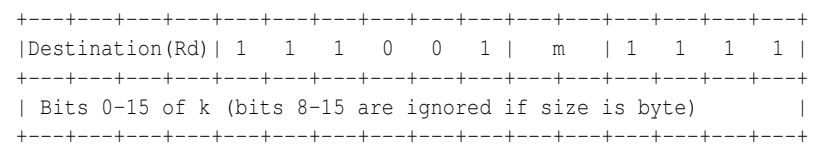
Instruction format: `MOVE.m [Rs+],Rd`



Instruction format: `MOVE.d k,Rd`



Instruction format: `MOVE.b k,Rd and MOVE.w k,Rd`



Mode (m)	Description
00	Byte
01	Word
10	Dword

2.3.56 MOVE - Move from General Register to Memory

Assembler syntax:

```
MOVE.m  Rs,[Rd]
MOVE.m  Rs,[Rd+]
```

Size: Byte, word or dword

Description: Move data from the source register to the destination. The size of the operation is m.

Operation:

```
*(m*)Rd = (m)Rs;
```

Flags affected: X flag cleared:

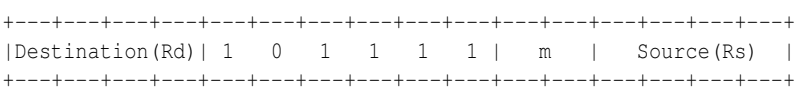
```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Flags affected: X flag set:

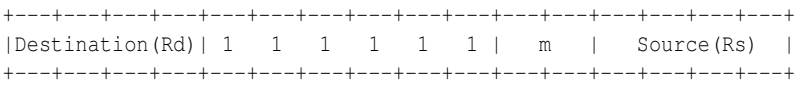
```
S R P U I X N Z V C
- - - - - 0 - - - *
```

The C flag is only affected if the X flag was set before the `MOVE CCS,d` instruction (conditional write).

Instruction format: `MOVE.m Rs,[Rd]`



Instruction format: `MOVE.m Rs,[Rd+]`



Mode (m)	Description
00	Byte
01	Word
10	Dword

2.3.57 MOVE - Move to Special Register

Assembler syntax:

```
MOVE Rs,Pd
MOVE [Rs],Pd
MOVE [Rs+],Pd
MOVE k,Pd
```

Size: Byte, word or dword depending on the size of of register Pd

Description: Move data from source to the destination special register. The size of the operation is the same as the size of the special register involved. Moves to constant registers are ignored. Constant registers are:

BZ, VR, WZ and DZ

In User Mode, moves to protected registers are ignored and may cause moves from special registers to yield invalid results the three following cycles. Protected registers are:

PID, SRS, EBP and SPC

Operation:

```
Pd = s;
```

Flags affected: (Pd != CCS)

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Flags affected: (Pd = CCS, Kernel Mode)

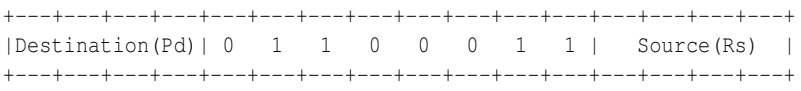
```
S R P U I X N Z V C
* * * * * * * * * *
```

Flags affected: (Pd = CCS, User Mode)

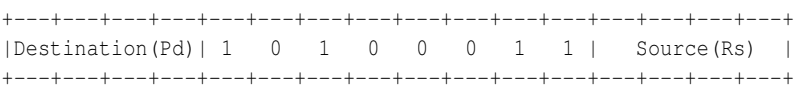
```
S R P U I X N Z V C
- * * - - * * * * *
```

The X flag is cleared after the instruction. If the X flag was set before a MOVE CCS,d instruction, the destination will have the bit corresponding to the X flag set.

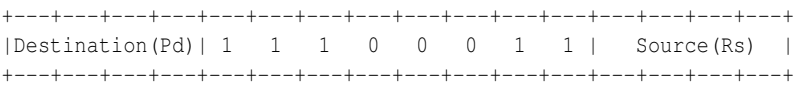
Instruction format: `MOVE Rs,Pd`



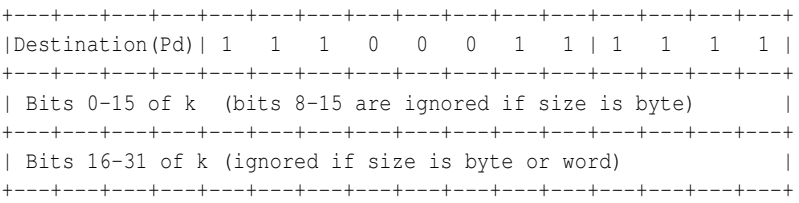
Instruction format: `MOVE [Rs],Pd`



Instruction format: `MOVE [Rs+],Pd`



Instruction format: `MOVE k,Pd`



2.3.58 MOVE - Move from Special Register to General Register

Assembler syntax:

```
MOVE Ps,Rd
```

Size: Byte, word or dword depending on the size of of register Ps

Description: Move data from the source special register to the destination general register. The size of the operation is the same as the size of the special register involved. The rest of the destination register is not affected.

Operation:

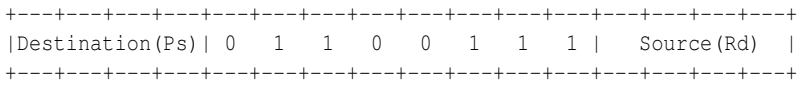
```
Rd = Ps;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

The X flag is cleared after the instruction. If the X flag was set before a MOVE CCS,Rd instruction, the destination will have the bit corresponding to the X flag set.

Instruction format: `MOVE Ps,Rd`



2.3.59 MOVE - Move from Special Register to Memory

Assembler syntax:

```
MOVE Ps,[Rd]
MOVE Ps,[Rd+]
```

Size: Byte, word or dword depending on the size of of register Ps

Description: Move data from the source special register to the destination. The size of the operation is the same as the size of the special register involved.

Operation:

```
(size)d = Ps;
```

Flags affected: (X flag was cleared)

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

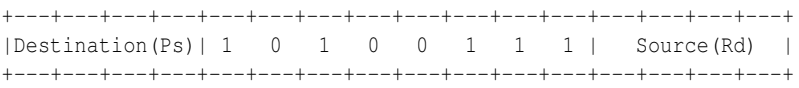
Flags affected: (X flag was set)

```
S R P U I X N Z V C
- - - - - 0 - - - *
```

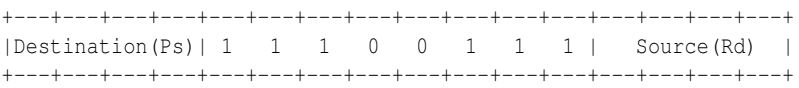
The C flag is only affected if the X flag was set before the MOVE CCS,d instruction (conditional write).

The X flag is cleared after the instruction. If the X flag was set before a MOVE CCS,d instruction, the destination will have the bit corresponding to the X flag set.

Instruction format: `MOVE Ps,[Rd]`



Instruction format: `MOVE Ps,[Rd+]`



2.3.60 MOVE - Move to Support Function Register

Assembler syntax:

```
MOVE Rs,Sd
```

Size: Byte, word or dword

Description: Move data from the source register Rs to the support function register Sd of the current support register bank. The current bank is selected by the SRS register. This instruction is only available in kernel mode, it is silently ignored in user mode.

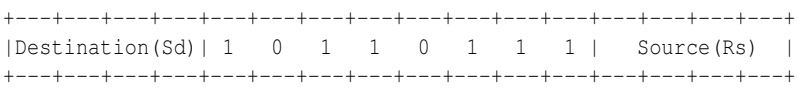
Operation:

```
Sd = Rs;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: MOVE Rs,Sd



2.3.61 MOVE - Move from Support Function Register

Assembler syntax:

```
MOVE Ss,Rd
```

Size: Byte, word or dword

Description: Move data from the support function register Sd of the current support register bank to the source register Rs. The current bank is selected by the SRS register.

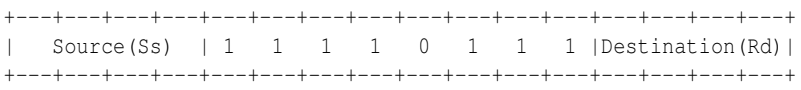
Operation:

```
Rd = Ss;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: `MOVE Ss,Rd`



2.3.62 MOVEM - Move Multiple Registers to Memory

Assembler syntax:

```
MOVEM Rs,[Rd]
MOVEM Rs,[Rd+]
```

Size: Dword

Description: The contents of registers R0 to Rs are stored to memory, starting at the memory location pointed to by Rd. The size of each register transfer is dword. R0 is stored at the lowest address: [Rd], and Rs is stored at the highest address:

[Rd + 4 * (<number of stored registers> - 1)].

If autoincrement addressing mode is specified, Rd is updated to

(Rd + 4 * <number of stored registers>).

Setting the X flag does not cause this instruction to perform a conditional write, as it does with other instructions that write to memory.

Operation:

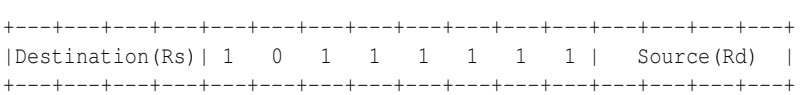
```
for (n = 0; n <= numberof(Rs); n++) {
   Rd[n] = Rn;
}
```

numberof(Rs) is the register number of Rs, n is an integer, and Rn the general register with register number n.

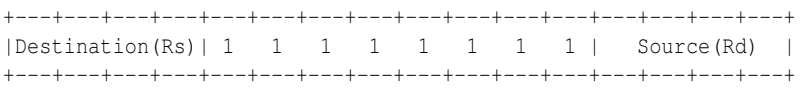
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: `MOVEM Rs,[Rd]`



Instruction format: `MOVEM Rs,[Rd+]`



2.3.63 MOVEM - Move Multiple register from Memory

Assembler syntax:

```
MOVEM [Rs],Rd
MOVEM [Rs+],Rd
```

Size: Dword

Description: The registers R0 to Rd are loaded from memory, starting at the memory location pointed to by Rs. The size of each register transfer is dword. R0 is loaded from the lowest address: [Rs], and Rd is loaded from the highest address:

[Rs + 4 * (<number of loaded registers> - 1)].

If autoincrement addressing mode is specified, Rs is updated to

(Rs + 4 * <number of loaded registers>).

Operation:

```
for (n = 0; n <= numberof(Rd); n++) {
   Rn = Rs[n];
}
```

numberof(Rd) is the register number of Rd, n is an integer, and Rn the general register with register number n.

Observe that the behavior is undefined unless numberof(Rs) is greater than numberof(Rd).

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: `MOVEM [Rs],Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   0   1   1   1   0   1   1 |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `MOVEM [Rs+],Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   1   1   1   1   0   1   1 |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

2.3.64 MOVEQ - Move Quick

Assembler syntax:

```
MOVEQ i,Rd
```

Size: Source data is 6-bit. Operation size is dword.

Description: The destination register is loaded with a 6-bit immediate value, sign extended to dword.

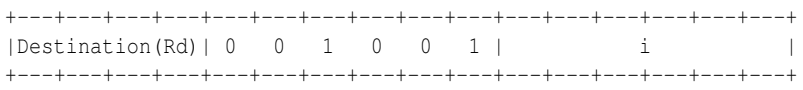
Operation:

```
Rd = i;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: MOVEQ i,Rd



2.3.65 MOVS - Move to General Register with Sign Extend

Assembler syntax:

```
MOVS.z Rs,Rd
MOVS.z [Rs],Rd
MOVS.z [Rs+],Rd
MOVS.z k,Rd
```

Size: Source size is byte or word. Operation size is dword.

Description: Move data from source to the destination register. The source data is sign extended from z to dword.

Operation:

```
Rd = (z)s;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: `MOVS.z Rs,Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 0   1   0   0   0   1   1 | z |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `MOVS.z [Rs],Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   0   0   0   0   1   1 | z |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `MOVS.z [Rs+],Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   1   0   0   0   1   1 | z |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `MOVS.z k,Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   1   0   0   0   1   1 | z | 1   1   1   1 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Bits 0-15 of k (bits 8-15 are ignored if size is byte)        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Size (z)	Description
0	Byte source operand
1	Word source operand

2.3.66 MOVU - Move to General Register with Zero Extend

Assembler syntax:

```
MOVU.z Rs,Rd
MOVU.z [Rs],Rd
MOVU.z [Rs+],Rd
MOVU.z k,Rd
```

Size: Source size is byte or word. Operation size is dword.

Description: Move data from source to the destination register. The source data is zero extended from z to dword.

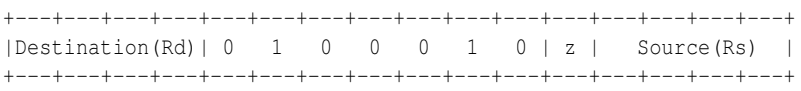
Operation:

```
Rd = (unsigned z) s;
```

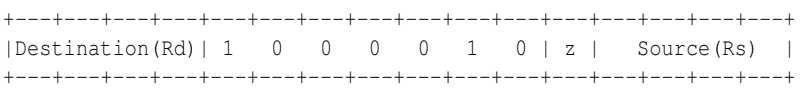
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 0 * - -
```

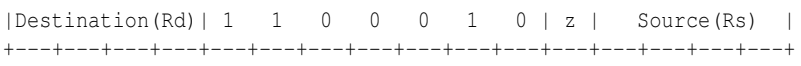
Instruction format: `MOVU.z Rs,Rd`



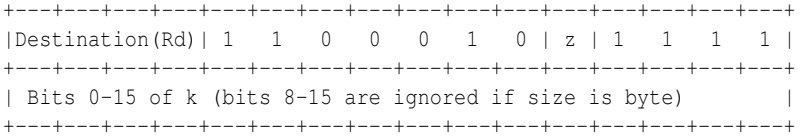
Instruction format: `MOVU.z [Rs],Rd`



Instruction format: `MOVU.z [Rs+],Rd`



Instruction format: `MOVU.z k,Rd`



Size (z)	Description
0	Byte source operand
1	Word source operand

2.3.67 MULS - Signed Multiply

Assembler syntax:

```
MULS.m Rs,Rd
```

Size: The operands are byte, word or dword. The result is 64 bits.

Description: Both operands are sign extended from the size (m) to dword, and the extended operands are multiplied, generating a 64-bit result. The lower 32 bits of the result are written to Rd, and the upper 32 bits are written to the multiply overflow register (MOF).

N and Z flags are set depending on the 64 bit result. The V flag is set if the result is more than 32 bits.

```
V-flag = ((Rd >= 0) && (MOF != 0)) ||
         ((Rd < 0) && (MOF != -1))
```

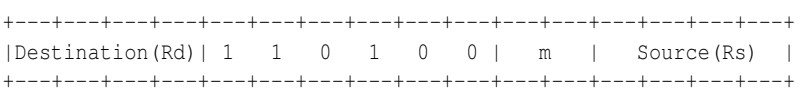
Operation:

```
MOF = ((m)Rs * (m)Rd) >> 32;
Rd = (dword)((m)Rs * (m)Rd);
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * -
```

Instruction format: `MULS.m Rs,Rd`



m	Description
00	Byte
01	Word
10	Dword

2.3.68 MULU - Unsigned Multiply

Assembler syntax:

```
MULU.m Rs,Rd
```

Size: Byte, word or dword. The result is 64 bits.

Description: Both operands are zero extended from the size (m) to dword, and the extended operands are multiplied, generating a 64-bit result.

The lower 32 bits of the result are written to Rd, and the upper 32 bits are written to the multiply overflow register (MOF).

N and Z flags are set depending on the 64 bit result. The V flag is set if the result is more than 32 bits.

```
V-flag = (MOF != 0)
```

Operation:

```
MOF = ((unsigned m)Rs * (unsigned m)Rd) >> 32;
Rd = (dword)((unsigned m)Rs * (unsigned m)Rd);
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * -
```

Instruction format: `MULU.m Rs,Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   0   0   1   0   0 |   m   |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

m	Description
00	Byte
01	Word
10	Dword

2.3.69 NEG - Negate

Assembler syntax:

```
NEG.m Rs,Rd
```

Size: Byte, word or dword

Description: The contents of the source register is negated, and stored in the destination register. The size of the operation is m. The rest of the destination register is not affected.

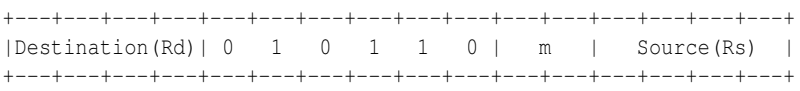
Operation:

```
(m)Rd = -(m)Rs;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

Instruction format: `NEG.m Rs,Rd`



m	Description
00	Byte
01	Word
10	Dword

2.3.70 NOP - No Operation

Assembler syntax:

```
NOP
```

Size: NA

Description: NOP is an alias for SETF.

2.3.71 NOT - Logical Complement

Assembler syntax:

```
NOT Rd
```

Size: Dword

Description: NOT Rd is an alias for SWAPN Rd. The contents of the destination register is inverted. The size of the operation is dword.

2.3.72 OR - Logical OR

Assembler syntax:

```
OR.m Rs,Rd
OR.m [Rs],Rd
OR.m [Rs+],Rd
OR.m k,Rd
```

Size: Byte, word or dword

Description: A logical OR is performed between the source operand and the destination register. The size of the operation is m. The rest of the destination register is not affected.

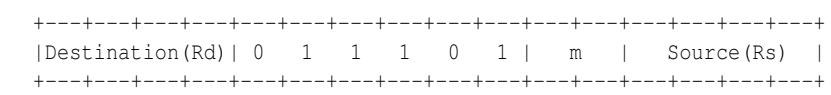
Operation:

```
Rd |= s;
```

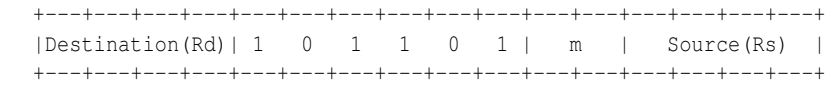
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

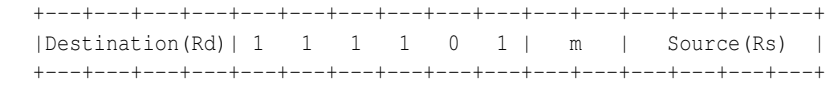
Instruction format: `OR.m Rs,Rd`



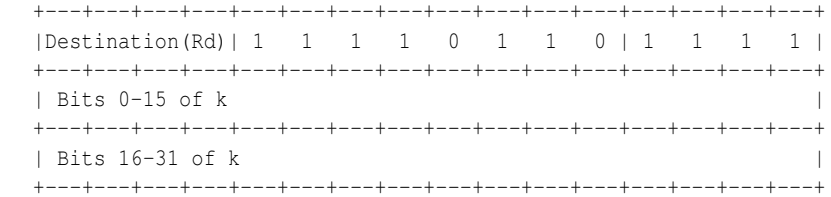
Instruction format: `OR.m [Rs],Rd`



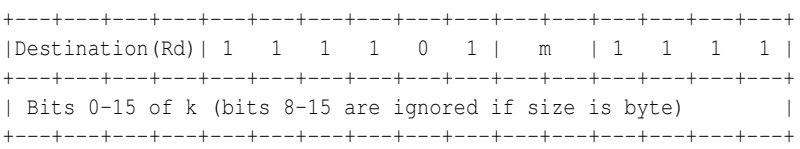
Instruction format: `OR.m [Rs+],Rd`



Instruction format: `OR.d k,Rd`



Instruction format: `OR.b k,Rd` and `OR.w k,Rd`



m	Description
00	Byte
01	Word
10	Dword

2.3.73 ORQ - Logical OR Quick

Assembler syntax:

```
ORQ i,Rd
```

Size: Source data is 6-bit. Operation size is dword.

Description: A logical OR is performed between a 6-bit immediate value, sign extended to dword, and the destination register.

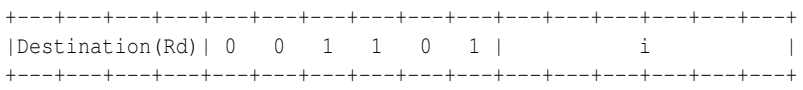
Operation:

```
Rd |= i;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: `ORQ i,Rd`



2.3.74 RET - Return From Subroutine

Assembler syntax:

```
RET
```

Description: RET is an alias for JUMP SRP.

2.3.75 RETE - Return From Exception

Assembler syntax:

```
RETE
```

Description: RETE is an alias for JUMP ERP.

2.3.76 RETN - Return from NMI Exception

Assembler syntax:

```
RETN
```

Description: RETN is an alias for JUMP NRP.

2.3.77 RFE - Restore from Exception

Assembler syntax:

```
RFE
```

Size: N/A

Description: RFE restores the saved flags by shifting the CCS. The operation performed on CCS is shown in Operation below.

Observe that the P flag is always set by RFE unless the R flag is set. This instruction is intended to be placed in the delay slot of a RETE or RETN instruction:

```
<exception routine>
...
RETE
RFE
```

Operation:

```
       {S,R,P,U,I,X,N,Z,V,C} = {S1,R1,R?P1:1,U1,I1,X1,N1,Z1,V1,C1};
{S1,R1,U1,I1,X1,N1,Z1,V1,C1} = {S2,R2,U2,I2,X2,N2,Z2,V2,C2};
{S2,R2,U2,I2,X2,N2,Z2,V2,C2} = 0;
```

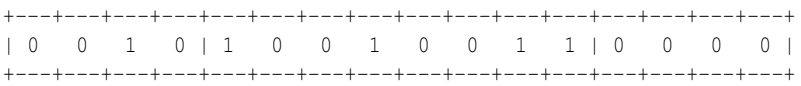
Flags affected: (Kernel mode)

```
S R P U I X N Z V C
* * * * * * * * * *
```

Flags affected: (User mode)

```
S R P U I X N Z V C
- * * - - * * * * *
```

Instruction format: RFE



2.3.78 RFG - Restore from Guru Mode Exception

Assembler syntax:

```
RFG
```

Size: N/A

Description: Restores the status saved when guru mode was entered (CCS, SRS and NRP). The hidden guru mode flag (g-flag) is also cleared. This instruction is intended to be placed in the delay slot of an RETN instruction:

```
<guru exception routine>
...
RETN
RFG
```

RFG has no effect in User and Kernel modes.

Operation:

```
CCS = G_CCS;
NRP = G_NRP;
SRS = G_SRS;
```

Flags affected: (Kernel and User Mode)

```
S R P U I X N Z V C
- - - - - - - - - -
```

Flags affected: (Guru Mode)

```
S R P U I X N Z V C
* * * * * * * * * *
```

Instruction format: RFG

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 0   1   0   0 | 1   0   0   1   0   0   1   1 | 0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

2.3.79 RFN - Restore from NMI Exception

Assembler syntax:

```
RFN
```

Size: N/A

Description: RFN restores the saved flags by shifting the CCS, then sets the M flag. The operation performed on CCS is shown in Operation below.

Observe that the P flag is always set by RFN unless the R flag is set. This instruction is intended to be placed in the delay slot of a RETE or RETN instruction:

```
<NMI exception routine>
...
RETN
RFN
```

Operation:

```
       {S,R,P,U,I,X,N,Z,V,C} = {S1,R1,R?P1:1,U1,I1,X1,N1,Z1,V1,C1};
{S1,R1,U1,I1,X1,N1,Z1,V1,C1} = {S2,R2:U2,I2,X2,N2,Z2,V2,C2};
{S2,R2,U2,I2,X2,N2,Z2,V2,C2} = 0;
                           M = 1;
```

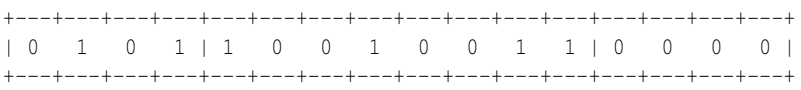
Flags affected: (Kernel mode)

```
S R P U I X N Z V C
* * * * * * * * * *
```

Flags affected: (User mode)

```
S R P U I X N Z V C
- * * - - * * * * *
```

Instruction format: RFN



2.3.80 Scc - Set Conditional

Assembler syntax:

```
Scc Rd
```

Size: Dword

Description: The destination register is loaded with 1 if the condition cc is true, and with 0 otherwise. The size of the operation is dword.

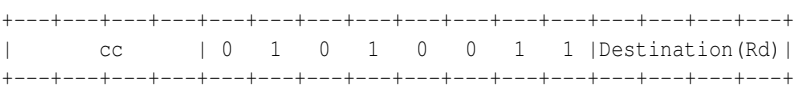
Operation:

```
if (cc) {
   Rd = 1;
} else {
   Rd = 0;
}
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 - - - -
```

Instruction format: `Scc Rd`



2.3.81 SETF - Set Flags

Assembler syntax:

```
SETF <list of flags>
```

Size: N/A

Description: The specified flags are set to 1. If the X flag is not in the list, it will be cleared. SETF may be used to enter User mode from Kernel mode by setting the U flag. The U and I flags can not be changed when in User mode (U==1).

Operation:

```
X = 0;
Selected flags = 1; /* U and I flags are not affected in User mode */
```

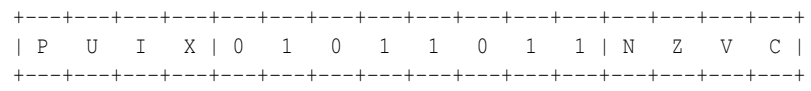
Flags affected: (Kernel mode)

```
S R P U I X N Z V C
- - * * * * * * * *
```

Flags affected: (User mode)

```
S R P U I X N Z V C
- - * - - * * * * *
```

Instruction format: `SETF <list of flags>`



2.3.82 SFE - Save for Exception

Assembler syntax:

```
SFE
```

Size: N/A

Description: Save flags by shifting the CCS. This operation is normally performed automatically when an exception occurs. This instruction can be used to fake calls to exception routines from kernel mode. If in User Mode the S, U and I flags are unaffected by this instruction.

Fake a call to an exception routine:

```
JAS exception_routine, ERP
SFE
```

Observe that this faked call only works from kernel mode.

Operation:

```
{S2,R2,P2,U2,I2,X2,N2,Z2,V2,C2} = {S1,R1,P1,U1,I1,X1,N1,Z1,V1,C1};
{S1,R1,P1,U1,I1,X1,N1,Z1,V1,C1} = {S,R,P,U,I,X,N,Z,V,C};
          {S,R,P,U,I,X,N,Z,V,C} = 0;
```

Flags affected: (Kernel mode)

```
S R P U I X N Z V C
0 0 0 0 0 0 0 0 0 0
```

Flags affected: (User mode)

```
S R P U I X N Z V C
- 0 0 - - 0 0 0 0 0
```

Instruction format: SFE

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 0   0   1   1 | 1   0   0   1   0   0   1   1 | 0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

2.3.83 SUB - Subtract

Assembler syntax:

```
SUB.m Rs,Rd
SUB.m [Rs],Rd
SUB.m [Rs+],Rd
SUB.m k,Rd
```

Size: Byte, word or dword

Description: The source data is subtracted from the destination register. The size of the operation is m. The rest of the destination register is not affected.

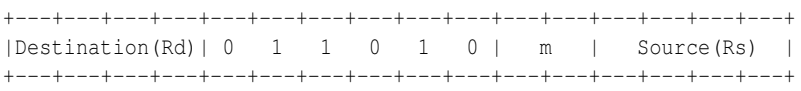
Operation:

```
(m)Rd -= (m)s;
```

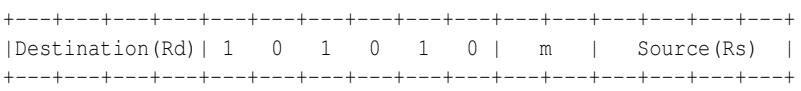
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

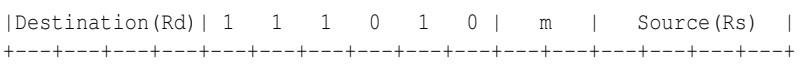
Instruction format: `SUB.m Rs,Rd`



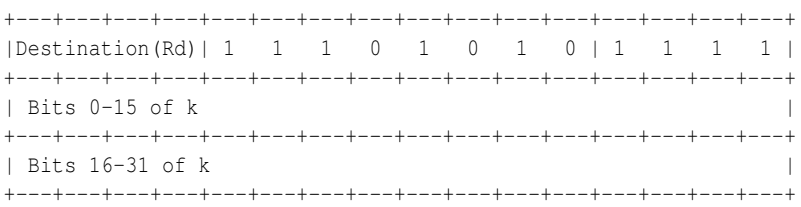
Instruction format: `SUB.m [Rs],Rd`



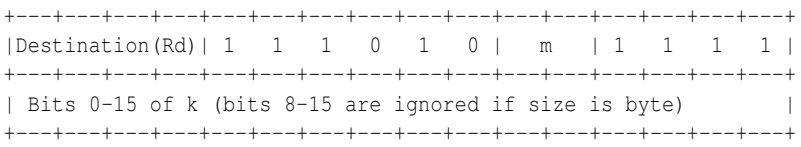
Instruction format: `SUB.m [Rs+],Rd`



Instruction format: `SUB.d k,Rd`



Instruction format: `SUB.b k,Rd and SUB.w k,Rd`



2.3.84 SUBQ - Subtract Quick

Assembler syntax:

```
SUBQ j,Rd
```

Size: Source data is 6-bit. Operation size is dword.

Description: A 6-bit immediate value, zero extended to dword, is subtracted from the destination register.

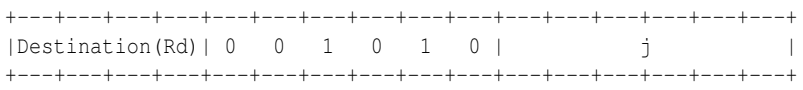
Operation:

```
Rd -= j;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

Instruction format: SUBQ j,Rd



2.3.85 SUBS - Subtract with Sign Extend

Assembler syntax:

```
SUBS.z Rs,Rd
SUBS.z [Rs],Rd
SUBS.z [Rs+],Rd
SUBS.z k,Rd
```

Size: Source size is byte or word. Operation size is dword.

Description: The source data is sign extended from z to dword, and then subtracted from the destination register.

Operation:

```
Rd -= (z)s;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

Instruction format: `SUBS.z [Rs+],Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 0   1   0   0   1   0   1 | z |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `SUBS.z [Rs],Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   0   0   0   1   0   1 | z |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `SUBS.z [Rs+],Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   1   0   0   1   0   1 | z |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Instruction format: `SUBS.z k,Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 1   1   0   0   1   0   1 | z | 1   1   1   1 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Bits 0-15 of k (bits 8-15 are ignored if size is byte)        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Size (z)	Description
0	Byte source operand
1	Word source operand

2.3.86 SUBU - Subtract with Zero Extend

Assembler syntax:

```
SUBU.z Rs,Rd
SUBU.z [Rs],Rd
SUBU.z [Rs+],Rd
SUBU.z k,Rd
```

Size: Source size is byte or word. Operation size is dword.

Description: The source data is zero extended from z to dword, and then subtracted from the destination register.

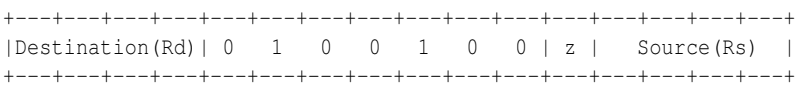
Operation:

```
Rd -= (unsigned z)s;
```

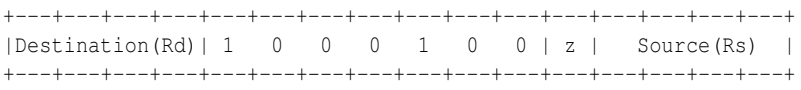
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * * *
```

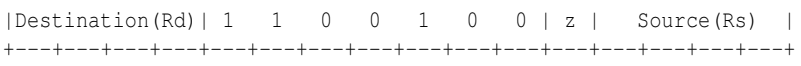
Instruction format: `SUBU.z Rs,Rd`



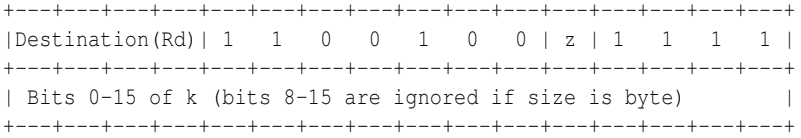
Instruction format: `SUBU.z [Rs],Rd`



Instruction format: `SUBU.z [Rs+],Rd`



Instruction format: `SUBU.z k,Rd`



Size (z)	Description
0	Byte source operand
1	Word source operand

2.3.87 SWAP - Swap Bits

Assembler syntax:

```
SWAP<option> Rd
```

Size: Dword

Description: The bits in the destination register are reorganized according to the specified option(s). The following options apply:

Flag	Description
N	Invert all bits in the operand.
W	Swap the words of the operand.
B	Swap the two bytes within each word of the operand.
R	Reverse the bit order within each byte of the operand.

Any combination of the four options are allowed. If more than one option is specified, they must be given in the order NWBR. The size of the operation is dword.

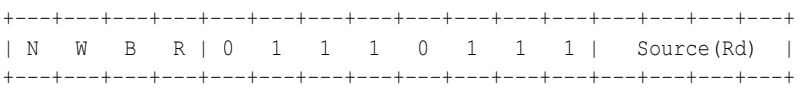
Operation:

```
if (option N) {
   Rd = ~Rd;
}
if (option W) {
   Rd = (Rd << 16) | ((Rd >> 16) & 0xffff);
}
if (option B) {
   Rd = ((Rd << 8) & 0xff00ff00) |
   ((Rd >> 8) & 0x00ff00ff);
}
if (option R) {
   Rd = ((Rd << 7) & 0x80808080) |
   ((Rd << 5) & 0x40404040) |
   ((Rd << 3) & 0x20202020) |
   ((Rd << 1) & 0x10101010) |
   ((Rd >> 1) & 0x08080808) |
   ((Rd >> 3) & 0x04040404) |
   ((Rd >> 5) & 0x02020202) |
   ((Rd >> 7) & 0x01010101);
}
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: SWAP<option> Rd



2.3.88 TEST - Compare with Zero

Assembler syntax:

```
TEST.m [Rs]
TEST.m [Rs+]
```

Size: Byte, word, or dword

Description: The source data is compared with 0, and the flags are set accordingly.

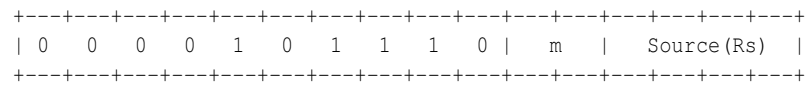
Operation:

```
(m)s - 0;
```

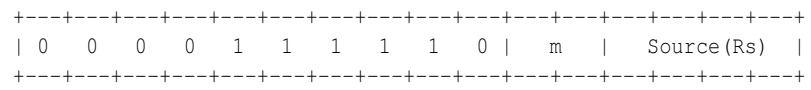
Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * 0 0
```

Instruction format: `TEST.m [Rs]`



Instruction format: `TEST.m [Rs+]`



m	Description
00	Byte
01	Word
10	Dword

2.3.89 XOR - Logical Exclusive OR

Assembler syntax:

```
XOR Rs,Rd
```

Size: Dword

Description: A logical XOR is performed between the contents of the source register and the destination register. The size of the operation is dword.

Operation:

```
Rd ^= Rs;
```

Flags affected:

```
S R P U I X N Z V C
- - - - - 0 * * - -
```

Instruction format: `XOR Rs,Rd`

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Destination(Rd)| 0   1   1   1   1   0   1   1 |   Source(Rs)  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

2.4 CRIS CPU Cycle behavior

2.4.1 References

[CACHE] Cache module datasheet, chapter 3

2.4.2 Pipeline Overview

In order to reduce the number of cycles needed to execute an instruction and allow for higher clock frequencies a pipelined architecture is often used in modern CPU architectures. This allows the resources to be split into several discrete portions and shared between several instructions simultaneously.

The CRIS v32 CPU uses a 5-stage pipelined architecture together with a 2-stage multiply unit allowing five instructions to execute simultaneously in the pipeline and one instruction to be completed in every clock cycle.

The different stages of the pipeline are shown in figure 2.13 below:

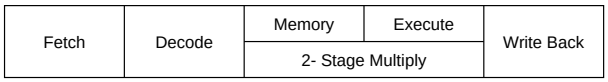


Figure 2.13: *CPU pipeline stages*

In the fetch stage instructions are fetched from the instruction cache into a small prefetch buffer in the CPU. The prefetch buffer is used to hide the latency of the cache

when crossing a cache line boundary. One full instruction is delivered for execution into the decode stage every clock cycle.

The decode stage is responsible for decoding the instruction and fetching the source register operands of the instruction from the register bank. In case of any memory operands a memory operation is prepared for the memory stage.

The memory stage handles fetching of source operands from memory and storing of destination operands to memory. The memory stage also includes the first part of a 2-stage multiplier unit. The memory stage is fully incorporated in the pipeline allowing one memory or multiply operation to be executed in each clock cycle without any extra penalty cycles.

In the execute stage the source operands are fed into an ALU (Arithmetic Logic Unit) where the selected operation of the instruction is performed. The execute stage also includes the second stage of the multiplier unit. The execute stage delivers the result of the operation in the form of the output from the ALU or multiplier unit together with new CCS settings.

The result from the execute stage is used by the write back stage which is responsible for storing the result back in the register bank.

2.4.2.1 Prefetch Unit

A 256-bit data bus is used towards the instruction cache allowing for load of up to 32 bytes of instruction words in each clock cycle. A prefetch unit is used to hide the instruction cache load latency when execution crosses a cache line boundary. The prefetch unit uses a 48-byte prefetch buffer and when less than 10 bytes are available the buffer is automatically filled up with the next sequential 32 bytes and rotated by the CPU. The prefetch unit assures that no extra stall cycles are inferred when fetching sequential program code as long as no instruction cache misses occur.

The prefetch buffer is always cleared at any jump or branch instruction causing the line to be fetched from the instruction cache again.

2.4.2.2 Branch Prediction Unit

The flags in the CCS register are not known until the instruction has passed the execute stage in the pipeline. When a conditional branch instruction is to be executed the execution would have to be stalled until the preceding instruction had passed the execute stage unless this was handled by the pipeline.

The CRIS v32 CPU uses a dynamic branch prediction unit to guess whether a conditional branch is to be taken or not. The predicted branch direction is executed speculatively and when the previous instruction passes the execute stage the prediction is checked with the actual flag settings. If the prediction is found to be correct no extra stall cycles are inferred but if the prediction was wrong the current execution has to be aborted and two extra stall cycles are needed to start executing the other direction of the branch.

The prediction unit uses a two-bit prediction scheme that requires that the predictor misses twice before being changed. This improves the predictor performance on

branches that strongly favours taken or not taken. The branch prediction unit uses a direct mapped 256-entry prediction buffer that is addressed by the lower eight bits of the program counter (PC). The prediction accuracy is normally well over 90% but branches that are switching direction often will of course degrade the predictor performance. The direct mapped buffer structure will also in rare cases degrade performance when two conditional branch instruction are mapped to the same entry in the prediction buffer and thus using the same two-bit predictor.

2.4.2.3 Memory Unit

A 256-bit data bus is used towards the data cache allowing for load or store of up to 32 bytes of data in each clock cycle. The data cache load and store latency is incorporated in the pipeline and does not infer any extra penalty cycles in the normal case.

Besides normal cache stalls there are cases where the data cache will stall the cpu pipeline. A load in any of the two cycles following directly after a store will stall the cpu for two cycles. A store in any of the two cycles following directly after a store to a clean cache line will also stall the cpu for two cycles. More information can be found in the cache documentation [CACHE].

There are no alignment restrictions on the data that causes any extra stall cycles, but if the data is larger than one cache line or crosses one or more cache line boundaries the access will be split into several sequential accesses and the CPU is stalled meanwhile.

2.4.2.4 Pipelined Multiplier Unit

For multiply instructions a 2-stage pipelined multiplier that overlaps the memory and execute stages is used. One multiplication can be started in the memory stage every clock cycle and the result is delivered in the execute stage one cycle later. Sequential multiplications are fully pipelined where the result of the first multiplication is delivered in the execute stage simultaneously as the first part of the second multiplication is computed in the memory stage.

2.4.3 Pipeline Hazards

In general, one instruction is completed every clock cycle. In some cases, however, dependencies between subsequent instructions as well as alignment of data and/or instructions may cause the execution of an instruction to use extra cycles. These exceptions are described in the sections below:

- Addresses
- Multiplication
- Jump with Register Operand
- Unaligned Data Accesses
- MOVEM
- Jump Targets

2.4.3.1 Addresses

If an instruction accessing memory, i.e. [Rs] or [Rs+], uses a register (Rs) that is modified by the preceding instruction, an extra cycle is needed to execute the instruction accessing memory.

This extra cycle is needed because the memory pipeline stage is placed before the execution stage. As a result, the value of a modified register is not available until the end of the execution stage while the instruction accessing memory needs the value at the start of the memory stage.

The following sequence needs one extra cycle:

```
ADD.d R1,R2
OR.d  [R2],R3
```

	Fetch	Decode	Memory	Execute	Write Back
cycle n		OR.d [R2],R3	ADD.d R1,R2		
cycle n+1		OR.d [R2],R3	<stall>	ADD.d R1,R2	
cycle n+2			OR.d [R2],R3	<stall>	ADD.d R1,R2

Observe that autoincrement is performed in the memory stage, so there is no need for extra cycles when the autoincremented register is used as an address in the next cycle. The following sequence does not require extra cycles:

```
ADD.w [R1+], R2
ADD.w [R1+], R2
ADD.w [R1+], R2
ADD.w [R1+], R2
```

2.4.3.2 Multiplication

As multiplication starts in the memory pipeline stage, the same dependency as described in section 2.4.3.1 applies. If any of the two multiplication operands are modified by the preceding instruction, with the exception of autoincrement, an extra cycle is needed to execute the multiplication.

2.4.3.3 Jump with Register Operand

Jumps need the target address in the decode stage. For immediate operands this is not a problem since immediate data is always available in the decode stage. For register operands, however, the register used has to have a valid value in the decode stage. This means that if the preceding instruction modifies the register used by the jump instruction, one extra cycle is needed. This applies to both regular and special registers.

This sequence needs one extra cycle:

```
ADD.d R1,R2
JUMP R2
```

	Fetch	Decode	Memory	Execute	Write Back
cycle n		JUMP R2	ADD.d R1,R2		
cycle n+1		JUMP R2	<stall>	ADD.d R1,R2	
cycle n+2			JUMP R2	<stall>	ADD.d R1,R2

This sequence needs one extra cycle:

```
MOVE R1,SRP
JUMP SRP
```

	Fetch	Decode	Memory	Execute	Write Back
cycle n		JUMP SRP	MOVE R1,SRP		
cycle n+1		JUMP SRP	<stall>	MOVE R1,SRP	
cycle n+2			JUMP SRP	<stall>	MOVE R1,SRP

2.4.3.4 Unaligned Data Accesses

Accesses spanning over a 32-byte cache line boundary require an extra cycle. The following accesses will cause an extra cycle:

- word access to address A, where (A & 0x1f) == 0x1f
- dword access to address A, where (A & 0x1f) >= 0x1d

2.4.3.5 Restarting After Data Cache Stalls

Two sequential memory operation instructions require an extra cycle for the second instruction if the first instruction causes a cache miss.

2.4.3.6 MOVEM

The MOVEM instruction may take between 1 and 3 cycles depending on the address and how many registers that are saved or restored. The following rules apply:

N Number of registers

A Address

- 1 cycle if A >> 5 == ((A + 4 * N) - 1) >> 5
- 2 cycles if A >> 5 == (((A + 4 * N) - 1) >> 5) - 1

- 3 cycles if A >> 5 == (((A + 4 * N) - 1) >> 5) - 2

The MOVEM Rs,[Rd/Rd+] instructions depend on Rd as well as R0..Rs, i.e. potentially a lot of registers. This makes forwarding of register values impractical. The requirement is that all registers saved must have valid values in the write back stage of the pipeline. In practice this means that if any of the two preceding instructions modifies any of the registers saved by MOVEM, extra cycles are inserted.

This sequence needs two extra cycles:

```
ADD.d R1,R3
MOVEM r7, [R10+]
```

	Fetch	Decode	Memory	Execute	WB
cycle n		MOVEM r7,[R10+]	ADD.d R1,R3		
cycle n+1		MOVEM r7,[R10+]	<stall>	ADD.d R1,R3	
cycle n+2		MOVEM r7,[R10+]	<stall>	<stall>	ADD.d R1,R3
cycle n+3			MOVEM r7,[R10+]	<stall>	<stall>

This sequence needs one extra cycle:

```
ADD.d R1,R3
ADD.d R1,R12
MOVEM R7, [R10+]
```

	Fetch	Decode	Memory	Execute	WB
cycle n		MOVEM R7,[R10+]	ADD.d R1,R12	ADD.d R1,R3	
cycle n+1		MOVEM R7,[R10+]	<stall>	ADD.d R1,R12	ADD.d R1,R3
cycle n+2			MOVEM R7,[R10+]	<stall>	ADD.d R1,R12

The destination registers loaded by the MOVEM [Rs/Rs+],Rd instructions are not practical to forward to e.g. the memory or execution stage. Registers loaded by MOVEM can thus not be used by other stages in the pipeline until after they have been written to the register bank. Extra cycles are inserted if registers loaded by MOVEM are used by any of the three instructions following the MOVEM.

This sequence requires three extra cycles:

```
MOVEM [R10],R7
SUBQ  1,r1
```

	Fetch	Decode	Memory	Execute	WB
cycle n		SUBQ 1,r1	MOVEM [R10],R7		
cycle n+1		SUBQ 1,r1	<stall>	MOVEM [R10],R7	

cycle n+2		SUBQ 1,r1	<stall>	<stall>	MOVEM [R10],R7
cycle n+3		SUBQ 1,r1	<stall>	<stall>	<stall>
cycle n+4			SUBQ 1,r1	<stall>	<stall>

This sequence requires two extra cycles:

```
MOVEM [R10],R7
SUBQ  1,r8
ADD.d r1,r12
```

	Fetch	Decode	Memory	Execute	WB
cycle n		ADD.d r1,r12	SUBQ 1,R8	MOVEM [R10],R7	
cycle n+1		ADD.d r1,r12	<stall>	SUBQ 1,R8	MOVEM [R10],R7
cycle n+2		ADD.d r1,r12	<stall>	<stall>	SUBQ 1,R8
cycle n+3			ADD.d r1,r12	<stall>	<stall>

This sequence also requires two extra cycles, for any S=b,w,d:

```
MOVEM  [R10],R7
MOVE.d r8,r1
SUBQ   1,R1
```

	Fetch	Decode	Memory	Execute	WB
cycle n		SUBQ 1,R1	MOVE.d r8,r1	MOVEM [R10],R7	
cycle n+1		SUBQ 1,R1	<stall>	MOVE.d r8,r1	MOVEM [R10],R7
cycle n+2		SUBQ 1,R1	<stall>	<stall>	MOVE.d r8,r1
cycle n+3			SUBQ 1,R1	<stall>	<stall>

However, this sequence requires no extra cycles:

```
MOVEM  [r10],r7
MOVE.d r8,r1
SUBQ   1,r8
```

This sequence requires one extra cycle:

```
MOVEM [R10],R7
ADD.d r8,r9
SUBQ  1,R9
SUBQ  1,R1
```

	Fetch	Decode	Memory	Execute	WB
cycle n		SUBQ 1,R1	SUBQ 1,R9	ADD.d r8,r9	MOVEM [R10],R7
cycle n+1		SUBQ 1,R1	<stall>	SUBQ 1,R9	ADD.d r8,r9
cycle n+2			SUBQ 1,R1	<stall>	SUBQ 1,R9

2.4.3.7 Jump Targets

If the instruction, at the address to which a JUMP instruction jumps, spans a cache line boundary, an extra cycle is needed to execute that instruction. This can only happen for instructions with word or dword immediate operands since two cache lines have to be fetched to get a complete instruction.

An extra cycle is needed for the following cases (A is the address of the instruction jumped to):

- instruction with word immediate, when A & 0x1f == 0x1e
- instruction with dword immediate, when A & 0x1f >= 0x1c

  These sequences need one extra cycle:

```
        JUMP 0x1e
        nop
        ...
0x1e:   ADD.w 34, R3          ;; 2 cycles
        ...


        ba 0x1c
        nop
        ...
0x1c:   ADD.d 1000000, R3     ;; 2 cycles
        ...
```

  No extra cycles are needed for corresponding sequential cache-line-straddling code sequences.

2.4.4 Self modifying code in the pipeline

It is generally not a problem to use self modifying code in the CRIS CPU. The instruction cache is part of the on-chip cache coherence mechanism. Cache lines that exist in the instruction cache will be automatically invalidated when they are written in the data cache.

The 48-byte prefetch buffer is not part of the cache coherence mechanism. Therefore, self modifying code that modifies code within 48 bytes of the current PC will not take effect in the pipeline until the prefetch buffer is updated as well. The simplest way of updating the prefetch buffer is to issue any type of jump or branch instruction. When jumping to self modified code, at least two nops must be inserted between the modifying of code and a following jump or branch to the same modified code:

```
; change immediate value
move.d instr + 2, r1
clear.d [r1]
nop
nop
jsr instr
nop
```

2.5 Assembly language syntax

2.5.1 Assembly language syntax

The syntax for the assembly language is covered by the [GAS] GNU assembler manual, specifically the chapter named ”CRIS Dependent Features”. This document is available from the GNU project (http://sourceware.org/binutils), and is also installed as part of the Axis compiler tools package known as cris-dist.

2.6 CRIS v32 Compiler specifics

2.6.1 GCC Compiler options

This section is an edited extract of the GCC documentation [GCC] (See section 2.1.1), where compiler -m options for different target processors are described.

These specifications are subject to change with future revisions of the CRIS v32 port of GCC.

For the reader familiar with the CRIS pre-v32 GCC port, the definitions are the same for all applicable parts.

The following -m options are defined for the CRIS architecture family:

```
-mcpu=CPU_MODEL
-march=CPU_MODEL
```

The two options above produce code that runs on CPU_MODEL. Values for CPU_MODEL are etrax4, etrax100, etrax100lx, and vN, where N is in the range from 0..32 are recognized. When vN is specified, N denotes the version-register contents of the targeted CPU model. The default is equivalent to -march=v32 when targeting the CRIS v32.

```
-mconst-align
-mdata-align
-mstack-align
-m16bit
-m32bit
```

```
-m8bit
-mno-const-align
-mno-data-align
-mno-stack-align
```

Align constants, data and stack respectively, to 16-bit (two bytes) data boundary by alignment directives, or by rounding up the size of the stack frame. Only individual variables are affected; the (unaligned) ABI is unaffected. Saying -m16bit is equivalent to all of -mconst-align, -mdata-align, and -mstack-align. This is the default when the base (v0) instruction set is specified. Saying -m32bit means rounding them up to a 32-bit data boundary. This is the default for the v8 instruction set and up, including CRIS v32. Specifying -m8bit means do not align anything. The no- option variant disables alignment of that entity.

```
-mmax-stack-frame=SIZE
```

Warn when the stack frame exceeds SIZE bytes.

```
-mprologue-epilogue
-mno-prologue-epilogue
```

Do (do not) output a prologue and epilogue for any function. For code compiled with the -mno-prologue-epilogue option, it is necessary to add a function prologue and epilogue through asm statements. The default is equivalent to -mprologue-epilogue.

2.6.2 C Preprocessor macros

The CRIS GCC port sets the following preprocessor macros:

```
__cris__
__CRIS__
__GNU_CRIS__
```

These three macros are always defined to 1.

```
__arch_X
```

The __arch_X macro is defined to 1 for the options -mcpu=X and -march=X (where the variable X is the value entered for CPU_MODEL). See section 2.6.1 for an explanation of these options. For CRIS v32, the macro __arch_v32 is defined to 1.

```
__CRIS_arch_version
```

The __CRIS_arch_version is defined to the numeric value N corresponding to the option -march=vN. For CRIS v32, the macro __CRIS_arch_version is defined to 32.

`__tune_X`

The __tune_X macro is set for the option -mtune=X in the same way as the macro __arch_X is for -march=X.

The underlining at the beginning and end of the macros above represents two underline characters.

2.6.3 The ABI

2.6.3.1 Introduction

This is a description of the Application Binary Interface (ABI), the binary-level conventions for the CRIS architecture family as implemented in the GCC port. An application binary interface defines some conventions common to all compiled programs. Among the conventions that an ABI establishes are register usage, calling conventions and layout of data. These specifications are subject to change with future revisions of the CRIS ABI. For the reader familiar with the pre-v32 CRIS ABI, the conventions in the CRIS v32 version are the same, with the addition of the call-clobbered register ACR for CRIS v32, and increased variation of the location of the return address in the stack frame. (See sections 2.6.3.4 and 2.6.3.5.)

2.6.3.2 Fundamental C data types

The following list shows how C and C++ data types correspond to CRIS data types. See table 2.7:

- A signed, unsigned, or plain char is a signed or unsigned byte (or 8-bit integer). The plain type ”char” is signed.
- A signed or unsigned short int is a signed or unsigned word (or 16-bit integer).
- A signed or unsigned int and long is a signed or unsigned dword (or 32-bit integer).
- Pointers to any type are represented as 32-bit integer entities.
- Enumerated types in C and C++, enum, are represented as integer objects, 32-bit dwords.
- The floating point type float is represented as 32-bit IEEE-754 floating point numbers.
- The types double and long double are represented as a 64-bit IEEE-754 floating point number, with the lower part of the mantissa in the dword at the lower address.

Figure 2.14: *32-bit floating point number*

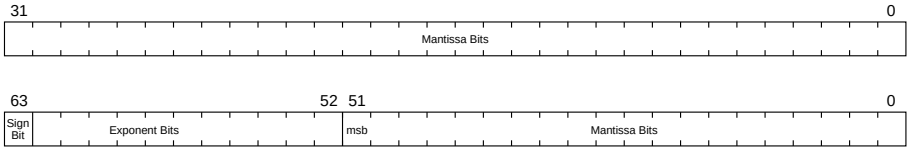


Figure 2.15: *64-bit floating point number*

2.6.3.3 C Object memory layout

The memory layout of a structure has each member at increasing addresses, without any alignment padding in between members. The size of the structure is, therefore, the sum of each of the sizes of the elements (with the exception of zero bitfields, which align to the next byte boundary).

- **Example:** of the structure layout of the CRIS ABI

```
struct example
{
   char c;            /* 1 Byte,  offset 0 */
   short s;           /* 2 Bytes, offset 1 */
   int i;             /* 4 Bytes, offset 3 */
   long l;            /* 4 Bytes, offset 7 */
   float f;           /* 4 Bytes, offset 11 */
   double d;          /* 8 Bytes, offset 15 */
   long double ld;    /* 8 Bytes, offset 23 */
   char s[6];         /* 6 Bytes, offset 31 */
};
```

  The size of struct example is 37 bytes.

Bitfields span over any byte, word or dword boundaries. The first declared field is in the lowest bits of the lowest address of the bitfield.

Compiler options specify whether objects have byte, word or dword alignment. Code must not assume that objects are laid out at stricter alignments than bytes. Compiler options specify the actual alignment. For example, -m8bit specifies that objects are always byte-aligned, while the default is 16-bit alignment. Note that options specifying a processor-version also implicitly control the alignment of objects. No target options affect structure layout or size.

2.6.3.4 C Calling conventions

Arguments shorter than or equal to 64 bits are passed by value. Integral types smaller than 32 bits are promoted to the corresponding 32-bit types by the same rules as in ISO C 1998-1999. Entities larger than 64 bits or declared to have varying size (regardless of the size at the time of the call) are passed by reference by passing a pointer to a read-only value. This means that the callee has to copy that value if it wants to modify it.

The first parameters to a function are passed (by value or reference) in registers R10..R13, starting with the first parameter in R10. If R13 is in turn for a 64-bit parameter, it is passed partially in R13 (the least significant 32 bits) and partially on stack (the most significant 32 bits). Parameters passed on stack are located at offset zero from SP upon entry to the called function (not including any space allocated for the return address).

A return value shorter than or equal to 64 bits is returned with the least significant 32 bits in register R10, and (if applicable) with the most significant 32 bits in R11. For structure return values, the caller allocates an area on the stack and passes the address of that area in R9 to the called function. The *this* pointer in C++ is passed as an invisible first argument in R10 (i.e. the first argument to a non-static member function ends up in register R11 and so on).

Registers R9..R13, SRP, ACR and MOF (except any return values in register R10 and R11) are assumed to contain garbage upon return from the function. Registers R0..R8 must have the same contents upon return from, as before the call to the function.

2.6.3.5 Stack frame layout

As can be seen below, the stack does not have a static layout except for the order of its components. It may, in fact, be collapsed and empty (not even contain a return address). For simplicity it is assumed in figure 2.16 below that all parameters are 32 bits or smaller. A function must not depend on the actual stack frame layout of its caller or callee for proper execution, only on the location of incoming and outgoing parameters.

Very few functions need a frame pointer. When a frame pointer is needed, the called register R8 is used. The frame pointer value is derived from the stack pointer value at the beginning of the function.

For functions with a variable number of parameters, the called function itself is responsible for storing any necessary portion of registers R10..R13 as indicated in figure 2.16. The calling function passes parameters exactly as to a function with a fixed number of parameters. The va_list type is a pointer to an array of parameters or (for each case the calling convention above indicates pass-by-reference) pointers to parameters.

The location of the return address varies depending on the function. If the function does not call any other functions (i.e. it is a ”leaf function”) and register SRP is not used by the function itself, the return address is kept there until the return. Other common cases include keeping the return address in another register or in the dedicated stack position in the figure 2.16.

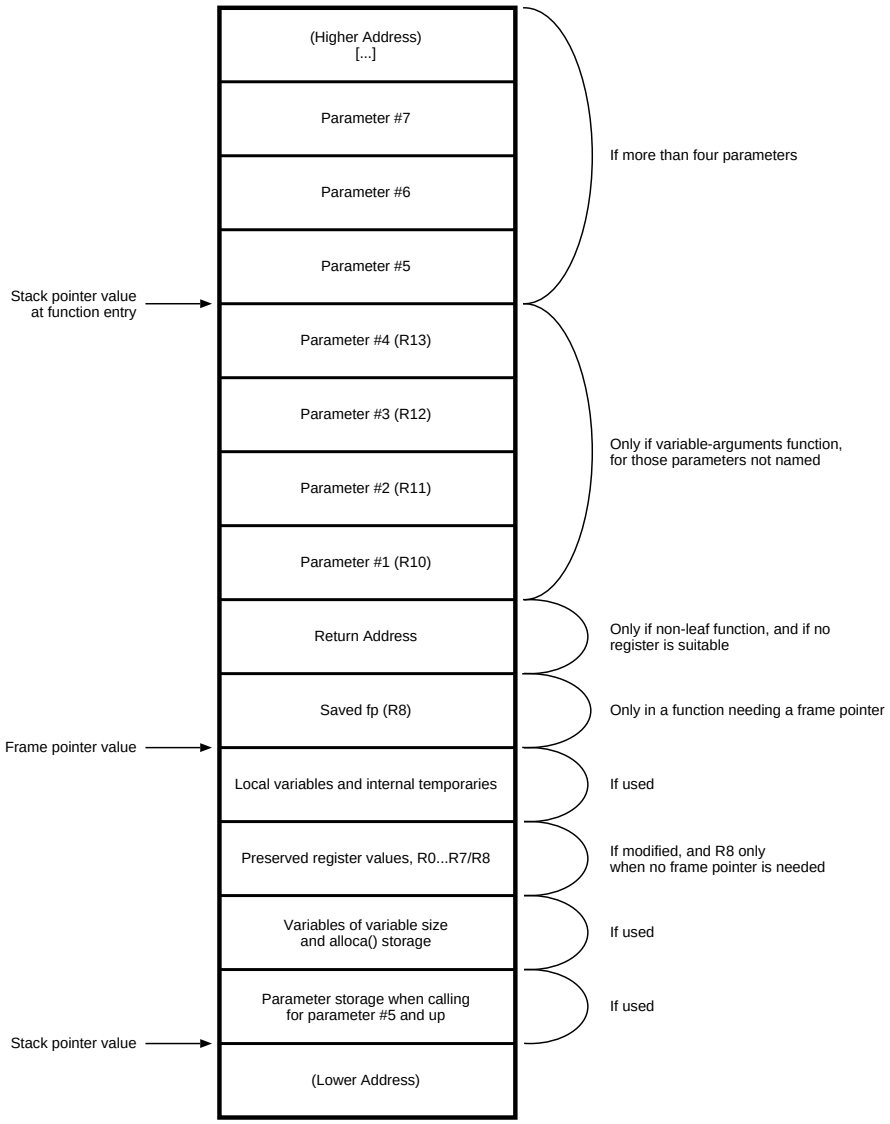


Figure 2.16: *Stack frame layout*

2.7 ETRAX FS and ETRAX 100LX CPU comparison

2.7.1 Introduction

Some properties of the instruction set in the CRIS v10 CPU (present in ETRAX 100LX) were not suitable for efficient pipelining (most notably the addressing mode prefixes). Binary compatibility has thus been dropped to allow adjustments to the instruction set to make it more pipeline friendly. Below is a summary of the differences between the CRIS v10 and CRIS v32 CPUs.

2.7.2 Registers

2.7.2.1 General registers

General register R15 is no longer PC. It is replaced by ACR (Address Calculation Register) which is used by the instructions replacing the Addressing Mode prefixes. There are two differences between R15 (ACR) and the other general registers:

1. R15 can not be used as address in the Autoincrement Mode. Specifying [R15+] still means [PC+].
2. Instructions that replace the addressing mode prefixes use ACR as an implicit destination.

2.7.2.2 Special registers

2.7.2.2.1 Removed special registers

The following ETRAX 100LX registers have been removed from the ETRAX FS CPU:

Register	Description
16-bit CCR (former P5)	16 bit Condition Code Register.
BAR (former P12)	Breakpoint Address Register.
BRP (former P14)	Breakpoint Return Pointer.

Table 2.78: *Removed ETRAX 100LX special registers*

2.7.2.2.2 New special registers

The following registers, which do not exist in the ETRAX 100LX , have been added to the ETRAX FS CPU:

Register	Description
PID (P2, Process/Page ID)	PID is an 32-bit register that is used, for example, by the TLB to know whether or not a mapping is valid for the current process.
SRS (P3, Support Register Select)	SRS is an 8-bit register used to select the active support register bank.
EXS (P5, Exception Status)	EXS is a 32-bit register for holding status information about the current Exception.
EDA (P6, Exception Data Address)	EDA is a 32-bit register for holding the data address used when an exception triggers (if any).
NRP (P12, NMI Return Pointer)	NRP is a 32-bit register used to store the return address for NMI exceptions.

Table 2.79: *New ETRAX FS special registers*

2.7.2.2.3 Renamed or modified special registers

The following ETRAX 100LX registers have been renamed or modified in the ETRAX FS CPU:

Register	Description
IBR (P9)	IBR has been renamed EBP (Exception Base Pointer).
IRP (P10)	IRP has been renamed ERP (Exception Return Pointer).
DCCR (P13)	DCCR has been renamed CCS and the 16-bit CCR has been removed. The bits in the CCS have changed as well.
USP (former P15)	USP has been moved to P14.

Table 2.80: *Renamed or modified ETRAX 100LX special registers*

2.7.2.3 Support function registers

The possibility of having banks of support function registers has been added to the ETRAX FS CPU. Registers on the current bank, selected by the SRS register, can be accessed by moving to or from a general register. There are up to 16 registers with 32 bits on each bank.

2.7.3 Addressing modes (Prefixes)

The address mode prefixes have been removed and replaced by stand alone address calculation instructions. The prefixes are unsuitable for a pipelined architecture.

The instructions that replaces the prefixes always assign the result of the address calculation to ACR (R15). ACR can then be used by subsequent instructions.

2.7.4 Instructions

2.7.4.1 Removed instructions

The following ETRAX 100LX instructions have been removed:

Instructions	Description
Address Mode Prefixes	BDAP, BIAP, DIP.
MSTEP	Multiply STEP. MULS and MULU can be used instead.
JMPU	Transfer to User mode is now made through setting the U flag in the CCS.
JIR, JIRC, JBRC	Jump to Interrupt Routine, Jump to Interrupt Routine with Context and Jump to Bus fault Routine with Context.
RBF, SBFS	Return from Bus Fault, Save Bus Fault Status.
MOVE Ps/Rs,PC	All move instructions with PC as destination operand are removed.
BOUND with memory operands.	BOUND with register and immediate operands is still present.

Table 2.81: *Removed ETRAX 100LX instructions*

2.7.4.2 Modified instructions

The following ETRAX 100LX instructions have been modified in the ETRAX FS CPU:

Instructions	Description
JUMP, JSR, JSRC	Addressing modes support has been reduced to register and immediate [PC+] only. A delay slot is also added to all jump instructions. The JSRC instruction is moved to a separate opcode due to the changed meaning of the op2 field in the JAS instruction. All jumps with immediate [PC+] are also moved to new opcodes.
MOVEM	The order in which the registers are saved has been changed. R0 is saved to or read from the lowest address.
NOP	NOP is redefined to SETF without parameters. The ADDI R0.b,R15 sequence used before is now a legal instruction.
RET, RETB, RETI	The return instructions all use the new JUMP Ps instruction since MOVE Ps,PC no longer exists.
All but ADD, SUB, CMP, TEST and specific flag-changing instructions	V and C flags are no longer cleared.

Table 2.82: *Modified ETRAX 100LX instructions*

2.7.4.3 New instructions

The following instructions, which do not exist for the ETRAX 100LX , have been added to the ETRAX FS CPU:

Instructions	Description
JUMP Ps	Jump to special register.
BA ao	32-bit relative jumps added.
BRS ao	
BSRC ao	
LAPC k,Rd	PC relative address calculation instructions.
LAPCQ qo,Rd	
RFE	Restore From Exception. Shift the CCS register right to restore flags when returning from an exception.
RFG	Restore From Guru Exception. Restore CCS and registers used when returning from a Guru mode Exception.
RFN	Restore From NMI Exception. Shift the CCS register right to restore flags when returning from an exception. Set M flag.
SFE	Save For Exception. Shift the CCS register left to save the flags when faking an exception.
HALT	Stop and wait for exception. Continue after exception has been server.
ADDC	Add with carry.
MCP	Multiply Carry Propagation. MCP is for efficient large number multiply.
FIDXI	Cache flush instructions.
FTAGI	
FIDXD	
FTAGD	

Table 2.83: *New ETRAX FS instructions*

2.7.4.4 Address mode prefix replacements

The following address mode prefixes have been replaced:

Instructions	Description
ADDOQ o,Rs,ACR	Replaces the quick immediate BDAP prefix.
ADDO.m [Rs],Rd,ACR	
ADDO.m [Rs+],Rd,ACR	Replaces the indirect BDAP prefix.
ADDI Rs.m,Rd,ACR	Replaces the BIAP prefix.

Table 2.84: *New ETRAX FS instructions*

2.7.5 Exception handling

The interrupt and bus fault mechanisms have been unified to one single exception mechanism. The only exception handled differently is NMI since this may occur at any time, even in the very start of a normal exception service routine.

One single vector table, pointed out by EBP, is used for all exceptions. When an exception occurs, the flags in the CCS are automatically saved by shifting the CCS left. Interrupts are automatically turned off and instructions that modify or read from the special registers or the flags no longer temporarily inhibit interrupts. When returning from an exception service routine the flags in the CCS should be restored by the RFE instruction.

Chapter 3

Cache

3.1 References

Reference	Description
[MACROS]	CRIS v32 support function register access macros, http://developer.axis.com
[DEFS]	CRIS v32 support function register constants and data types, http://developer.axis.com

Table 3.1: *References*

3.2 Overview

A cache is a fast local memory connected between a CPU and main memory. Caches are used to hold frequently used data close to the CPU in order to hide main memory access time. Local data that can be found in a cache can be accessed in one cycle while main memory access time is in the order of several tenths of CPU cycles.

The CPU uses two separate caches, one 16-kbyte instruction cache and one 16-kbyte data cache. For more information about the CPU see the CRIS v32 documentation, chapter 2. Both caches have the same behavior and this description can be applied to both the instruction and the data cache. The instruction cache is a read-only cache, so sections discussing the CPU write behavior are not applicable to the instruction cache.

The cache is physically addressed. The CPU address is, therefore, always translated by the MMU before being used in the cache. For more information about translating virtual addresses into physical addresses see the MMU documentation, chapter 7.

Each cache is connected to the CPU via a separate 256-bit local CPU bus that allows read or write one single byte up to a maximum of 32 bytes per cycle. If the CPU tries to access data that can not be found in a cache memory, the cache will halt the CPU and start to fetch the data from main memory. This is done via a separate 256-bit shared memory bus. When data is available, the CPU will resume the access.

The caches are designed to work in a shared memory system where several units have

access to main memory. Therefore, each cache might have local copies of portions of the main memory that are shared between several units. When other units access these shared portions, a cache coherence mechanism will automatically handle any upcoming consistency problems and make sure that main memory and local copies in any cache always are coherent. The cache coherence mechanism also applies between the local instruction and data caches within one single CPU.

3.3 Functional description

3.3.1 Cache organization

The 16 kbytes of each cache are divided into 512 cache lines. The cache line size is 32 bytes, which is the quantity a cache works on against main memory. For each cache line there is a corresponding entry in a local tag memory, indicating the address and state of the data currently present in the cache line.

Each cache is a 2-way set associative memory meaning that the cache is divided into two banks. Data from a specific memory address can only be placed into two alternative locations in the cache, one in each bank. Each of the two banks are 8 kbytes large, which means that the specific addresses 'a' and 'a + n * 8 kbyte' will have the same two possible positions in the cache.

When the CPU writes to a cache, data is only written in the cache and not to main memory. This is commonly known as a copy-back cache mechanism. The modified data is not written to main memory until it is replaced by some other data, or there is a request for the same data from another unit in the shared memory system.

3.3.2 Cache coherence

The cache coherence protocol is a mechanism for making sure that all accesses to memory read or write the proper data in the presence of one or more caches. The protocol used for these caches is called MESI. This is an acronym for Modified, Exclusive, Shared and Invalid and reflects the four states that each cache line might be in.

The states can be explained as:

Modified Valid-and-written-to-more-than-once-by-this-CPU

Exclusive Valid-and-not-present-in-any-other-caches

Shared Valid-and-never-written-to-by-any-CPU

Invalid Not-valid

The state transitions are automatically handled by each cache and are totally invisible for the programmer. State transitions will normally not affect the performance, but there are situations where sharing of data between several units in the shared memory system may affect performance.

Only one cache in the system is allowed to have a specific cache line in the modified state at the same time, which is assured by the MESI protocol itself. However, if another unit requests the same cache line, the modified line must first be written to main memory before the other unit can access the line. If this mechanism is triggered too often this will seriously degrade performance.

3.3.3 Cache hits

A cache hit is when the CPU tries to access data that can be found in a cache. There is normally no penalty for the CPU during read or write hits in a cache. Sequential reads and writes will be pipelined to one access per cycle.

If the CPU tries to flush or read from a cache in either of the two cycles following directly after a write to the same cache, the CPU will be stalled for two cycles caused by internal pipelining of the writes in the cache.

A cache line that the CPU has never written to is commonly known as a clean cache line. A clean cache line is marked as shared in the MESI protocol. In order to keep all caches coherent a write to a shared line will cause a write-through operation and update main memory as well. The write-through operation will assure that shared (clean) copies of the same cache line in any other caches will get invalidated. The CPU will be stalled until the write-through operation has finished. The cache line will then enter the exclusive state and future writes to the same cache line will not be stalled.

A cache line that is present in one single cache and still coherent with main memory is marked as exclusive. A write operation to an exclusive line will cause a state update to modified in the cache to indicate that the cache line is no longer coherent with main memory. Any request in either of the two cycles following a write to an exclusive cache line will be stalled for two cycles caused by the internal cache line state update in the tag memory.

3.3.4 Cache misses

A cache miss is when the CPU tries to access data that can not be found in a cache. On a cache read miss or write miss, the CPU is stalled and the mechanism of replacing one cache line is started. The cache has the possibility to replace a line in any of the two banks. This selection is made by a Least Recently Used (LRU) algorithm.

If the cache line that is selected to be replaced is clean, the new cache line is fetched immediately from main memory. If not, the current data present in the cache line is first written to main memory before the new line is read into the cache.

3.3.5 Non-cached accesses

Accesses with address bit 31 set will bypass the cache and directly access main memory. There is of course a performance penalty for doing non-cached accesses. This is caused by shared memory bus arbitration and slow main memory access time.

Care must be taken when doing non-cached accesses of data that are present in any cache, or the cache and main memory might not be consistent anymore. Non-cached

accesses will not trigger the cache coherence mechanism so any data that has been modified inside any cache in the system should never be referred to in a non-cached manner. A non-cached read of modified data will cause the wrong value to be read by the CPU while a non-cached write will cause the cache and main memory to be inconsistent.

Therefore, data that is referred to in a non-cached manner must first be assured to be nonexistent in all caches in order to avoid these kinds of inconsistency problems. This can be done by special flush operations.

3.3.6 Conditional write operation

The cache supports conditional write operations to make sure that the CPU is able to do atomic read-modify-write cycles to portions of main memory that are shared between several units in the system.

A conditional write that results in a cache hit will succeed and result in a normal write in the cache. A conditional write that causes a cache miss will be ignored and thus fail, and the missing data will not be fetched from main memory.

A conditional write to an non-cached region, i.e., address bit 31 set, is always treated as a hit and will thus never fail. Non-cached accesses will not trigger the cache coherence mechanism and is therefore useless for atomic read-modify-write operations.

By using the conditional write operation, an atomic read-write operation in main memory can be achieved by the following pseudo-code sequence:

```
retry:  read
        modify
        conditional write
        retry if write failed
```

The first read operation assures that data is present in the local cache. After data has been modified, it must be written by a conditional write. If any other unit has written to the same data during the read-modify sequence, data will no longer be present in the local cache. In this case, the following conditional write will fail and the whole sequence must be restarted. If data is still in the local cache during the conditional write, this means that no other unit has written to this data and the sequence is assured to be atomic.

For a description of the conditional write mechanism in the CPU, see 2.1.14.

3.3.7 Flush operations

In some cases it is desirable to make sure that certain data is not present in a local cache. This can be done by two different types of flush operations, flush index and flush tag, controlled by the CPU. Both flush operations can be used to set a specific cache line to the invalid state.

3.3.7.1 Flush index

Flush index is used to unconditionally flush any line in one of the banks in the cache. If the line is modified, data will be written to main memory before the line is flushed. The address used during a flush index operation is divided into two parts, index and bank. The rest of the address is unused.

address[12:5] - select which index in tag memory to flush

address[4] - select which bank to flush

Flush index can be used to unconditionally flush all cache lines. In order to flush all cache lines, the CPU must loop through all possible indexes in both banks.

Flush index is also used to initialize the tag memory into a well defined state after system reset with the cache disabled. In this mode the tag contents are ignored and any modified data will not be written to main memory.

3.3.7.2 Flush tag

Flush tag is used to flush a certain cache line in the cache. The address used during a flush tag operation is used to compare with the tag memory contents in order to only flush the line if it exists in the cache. If the line is modified, data will be written to main memory before the line is flushed. Address bit 31 is ignored during a flush tag operation, and there is no difference between a cached and non-cached flush tag operation. When the cache is disabled, the flush tag operation is ignored and will not have any effect on the cache.

One case where the flush tag operation is useful is when the CPU has produced cached modified data to be used in, for example, a DMA transfer with very strict latency requirements. If the data is modified in the cache when the DMA is started, a lot of coherence traffic will be generated in order to update main memory. In this case it might be beneficial to flush the affected cache lines in advance in order to write the modified data to main memory.

Another case where flush tag operations are useful is when parts of main memory are updated by mechanisms other than by writes through the normal main memory system. One example is that some types of flash memories are written by a separate serial protocol. To avoid inconsistency problems after writes, the updated region should be flushed from all caches.

3.3.7.3 Flushing other caches

If there are other caches connected to the same on-chip shared memory system these might in some cases need to be flushed as well.

It is possible to force all other caches to write any modified data to main memory by issuing a normal read operation on the specific address.

In order to make sure that no caches have any modified data, the normal read should be followed by a flush tag operation in the local cache on the specific address.

3.3.8 Enable/disable the cache

The cache operation mode is controlled by the CPU support function register rw_gc_cfg as defined in 25.9. General information about support function registers can be found in the CPU documentation, chapter 2. To access the registers, fields and register constants from a C program, a set of macros and data types are defined in [MACROS] and [DEFS].

After reset, the caches are disabled by default. In this mode all CPU accesses are treated as non-cached accesses (irrespective of address bit 31). This is not visible for normal program code in ETRAX FS. The internal boot ROM will initialize and enable the caches before jumping to normal program code.

3.3.8.1 Initialization

Before a cache can be enabled, the tag memory must be initialized to a well defined state. This is done by looping through all indexes in both banks by the flush index operation while the cache is disabled. The internal boot ROM in ETRAX FS handles this initialization before enabling the caches.

3.3.8.2 Disabling the cache

Care must be taken if a cache is to be disabled during normal operation. All cache lines must then be flushed before the cache is disabled in order to assure that any cached modified data will be consistent with main memory. The program that handles the flushing and disabling of the cache must of course not modify any cached data itself.

3.4 Software examples

The functions fidxi(), fidxd(), ftagi() and ftagd() are used in the examples below to issue flush index and flush tag operations to the instruction and data caches. In the CPU this is done by issuing the FIDXI, FIDXD, FTAGI and FTAGD instructions. More information about these instructions can be found in the CPU documentation, chapter 2.

3.4.1 Initialize instruction cache (while disabled)

This code shows the part of the internal boot ROM in ETRAX FS that is used for initializing the instruction cache.

```
#define BANK_SIZE 8192
#define LINE_SIZE 32
void icache_init() {
  int idx;
  // Address bit 4 is used to select which bank to flush. The
```

```
  // loop variable is therefore increased by half of the cache
  // line size in each round to flush all lines in both banks
  for( idx = 0; idx < BANK_SIZE; idx += (LINE_SIZE / 2) ) {
     fidxi( idx );
  }
}
```

3.4.2 Flush whole data cache

```
#define BANK_SIZE 8192
#define LINE_SIZE 32
void dcache_flush_all() {
  int idx;
  // Address bit 4 is used to select which bank to flush. The
  // loop variable is therefore increased by half of the cache
  // line size in each round to flush all lines in both banks
  for( idx = 0; idx < BANK_SIZE; idx += (LINE_SIZE / 2) ) {
     fidxd( idx );
  }
}
```

3.4.3 Flush specific address region in data cache

```
#define LINE_SIZE 32
void dcache_flush_region( unsigned from, unsigned to ) {
  unsigned a;
  // Increase loop variable by cache line size to loop through
  // all cache lines in the region.
  for( a = from; a < to; a += LINE_SIZE ) {
    ftagd( a );
  }
}
```

3.4.4 Flush specific address in data cache

```
void dcache_flush_addr( unsigned a ) {
  ftagd( a );
}
```

Chapter 4

Bus interface

4.1 References

Reference	Description
[BOOTROM]	Chapter 6
[BIF_REGS]	Chapter 25.4
[BIF_MACROS]	http://developer.axis.com
[EXTDMA_REGS]	Chapter 25.5
[EXTDMA_MACROS]	http://developer.axis.com
[MEMARB]	Chapter 14
[SLAVE_REGS]	Chapter 25.6
[SLAVE_MACROS]	http://developer.axis.com
[EXT_REGS]	Chapter 25.7
[EXT_MACROS]	http://developer.axis.com
[STRMUX]	Chapter 10
[DMA]	Chapter 5
[PINMAP]	Chapter 16

Table 4.1: *References*

4.2 Overview

The bus interface implements the interface between on-chip functions and an external memory bus. The bus interface can operate in two main modes:

1. master mode

2. slave mode

4.2.1 Master mode

In master mode, the on-chip functions control the external memory bus via two different methods: Through the central memory arbiter, and through the internal DMA via the external DMA block.

In master mode the bus interface controls external memory modules. The supported memory modules are SRAM types (SRAM/flash/peripheral units), SDRAM and NAND flashes.

4.2.2 Slave mode

In slave mode, the on-chip functions are controlled from the external bus. In this mode, the chip looks like an I/O device to the external bus master. The internal functions are controlled via two different methods: Through the central memory arbiter, and through the internal DMA via the external DMA block.

4.3 Functional description

4.3.1 General

The bus interface has a 32-bit data bus, a 25-bit address bus, and 12 internally decoded chip select outputs. The bus interface also supports 8 Synchronous DRAM (SDRAM) banks without external logic.

The bus interface works with bursts of data. These bursts can be 1-8 bus cycles with 32-bit data bus width, and 1-16 bus cycles with 16-bit data bus width.

The general functionality of the bus interface is controlled by a set of mode registers, see 25.4. Register, field and register constant names in sections 4.3.3 - 4.3.11 refer to this set of registers. The Bus arbitration, external DMA and slave mode operation have separate sets of mode registers, described in the respective sections.

4.3.2 Data bus

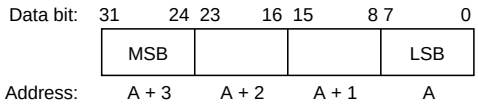
The data bus shown in figure 4.1 is 32 bits wide, but also supports 16-bit wide memories. The data bus is organized with the least significant byte at the lowest address (”little endian”).

4.3.3 Address and chip selects

The external address bus consists of 25 pins, **a25** - **a1**. The internal address bits 30 - 26 are decoded in the bus interface to generate the 12 different memory chip select outputs and the selection of SDRAM banks.

Address bit 31 is ignored by the bus interface since it is used to select whether the cache should be used or bypassed by CPU accesses.

32-bit Mode



16-bit Mode

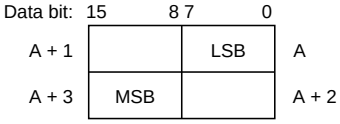


Figure 4.1: *Data bus width*

The names of the different chip selects are **cse0_n**, **cse1_n**, **csr0_n**, **csr1_n**, **csp0_n**, **csp1_n**, **csp2_n**, **csp3_n**, **csp4_n**, **csp5_n**, **csp6_n** and **css_n**.

A memory map is given in the table below.

Address range	MByte	Name	Usage
00000000-03FFFFFF	64	**cse0_n**	EPROM/flashPROM select 0.
04000000-07FFFFFF	64	**cse1_n**	EPROM/flashPROM select 1.
08000000-0BFFFFFF	64	**csr0_n**	SRAM select 0.
0C000000-0FFFFFFF	64	**csr1_n**	SRAM select 1.
10000000-13FFFFFF	64	**csp0_n**	Peripheral select 0.
14000000-17FFFFFF	64	**csp1_n**	Peripheral select 1.
18000000-1BFFFFFF	64	**csp2_n**	Peripheral select 2.
1C000000-1FFFFFFF	64	**csp3_n**	Peripheral select 3.
20000000-23FFFFFF	64	**csp4_n**	Peripheral select 4.
24000000-27FFFFFF	64	**csp5_n**	Peripheral select 5.
28000000-2BFFFFFF	64	**csp6_n**	Peripheral select 6.
2C000000-2FFFFFFF	64	**css_n**	Slave chip select.
30000000-3FFFFFFF	256	-	Not used by the bus interface.[1]
40000000-7FFFFFFF	1024	-	SDRAM interface select.
80000000-AFFFFFFF	768	-	00000000-2FFFFFFF but non-cached.
B0000000-BFFFFFFF	256	-	Not used by the bus interface. [1]
C0000000-FFFFFFFF	1024	-	40000000-7FFFFFFF but non-cached.

Table 4.2: *Memory map*

[1] 30000000-3FFFFFFF and B0000000-BFFFFFFF are used for mode registers and on-chip memory. Accesses to these will not be routed through the bus interface.

The memory banks are separated in five different groups:

Group	Chip selects
Group 1	**cse0_n**, **cse1_n**
Group 2	**csr0_n**, **csr1_n**
Group 3	**csp0_n**, **csp1_n**, **csp2_n**, **csp3_n**
Group 4	**csp4_n**, **csp5_n**, **csp6_n**, **css_n**
Group 5	SDRAM banks

Table 4.3: *Memory bank groups*

Configuration of wait states, bus width etc. can be made separately for each group, but all banks in the same group have the same configurations.

4.3.3.1 Gated chip select

In chip select group 3 and group 4 the individual chip select signals **csp0_n - csp6_n** can be gated internally with either the **wr0_n** or the **rd_n** signal. This is controlled in the registers rw_grp3_cfg and rw_grp4_cfg.

In figure 4.2, cspx_n can be any of the signals **csp0_n - csp6_n**.

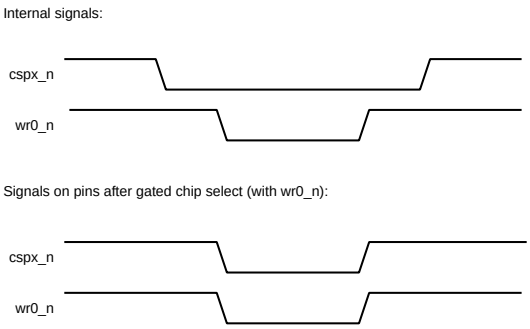


Figure 4.2: *Gated chip select*

4.3.4 Internal priority in master mode

In master mode, the bus interface arbitrates between normal memory cycles, external DMA cycles and SDRAM refresh cycles.

SDRAM refresh cycles have the highest priority. External DMA (as a whole) and normal memory accesses have the same priority. If both are requested, they will alternate.

External DMA will break up SRAM/flash/peripheral bursts, but it will not break up SDRAM bursts. If the external DMA is configured for burst accesses, its bursts will not be broken up by the normal memory cycles or SDRAM refresh cycles.

4.3.5 Bus width

In master mode, the bus width of each group of memory banks, see table 4.3, can be configured to either a 32-bit or 16-bit bus width. The bus width is controlled in registers rw_grp1_cfg, rw_grp2_cfg, rw_grp3_cfg and rw_grp4_cfg.

The bus width of group 1 is initialized from the boot ROM at system reset. This is described in 6.

The external DMA bus width is configurable for 8, 16 or 32 bits. This is true for both master and slave mode.

In non-DMA slave mode, the bus width is always 32 bits.

4.3.6 Bus states

The SRAM/flash and peripheral interface bus sequence is divided into different bus states. The bus states Ta, Td and Tz are always present in a burst. The other bus states can be set with wait states in rw_grp1_cfg, rw_grp2_cfg, rw_grp3_cfg and rw_grp4_cfg. The wait states are described in sections 4.3.6.1 to 4.3.6.6. The bus states are listed below:

T$_a$ Activate state. The **rd_n** and **wr0_n - wr3_n** signals are asserted in this state.

T$_d$ Data state. Data input is sampled at the end of this state.

T$_z$ Data bus turn-off state. This state is inserted between bursts, to allow the bus interface and external units to turn off their outputs before the data bus is driven by another source. This state may overlap with a Taw state, a Tew state or a Tewb state.

T$_{ew}$ Early wait state. This bus state is inserted before Ta and may overlap with a Tz state or a Tzw state.

T$_{lw}$ Late wait state. This bus state is inserted between Ta and Td.

T$_{zw}$ Turn-off wait state. This bus state is inserted after Td. Tzw may overlap with a Taw state, a Tew state or a Tewb state.

T$_{dw}$ Data setup wait state. The Tdw state holds data valid before Ta. This bus state is inserted between Tew and Ta or before Ta if no Tew is present.

T$_{aw}$ Address recovery wait state. The Taw state holds the address valid after the Td state. This bus state is inserted after Td and may overlap Tzw and Tz but not Tew or Tewb.

T$_{ewb}$ Early wait state burst. This bus state is inserted before Tew in the first data cycle in a burst or before Ta in the first data cycle in a burst if no Tew is present. Tewb may overlap with Tz state or Tzw state.

In the figures below, `cs_n` denotes any of the chip selects **cse0_n**, **cse1_n**, **csr0_n**, **csr1_n**, **csp0_n - csp6_n** or **css_n**, and `wr_n` denotes any of the write signals **wr0_n - wr3_n**.

Diagram 4.3 is an example of a write cycle showing the different bus states.

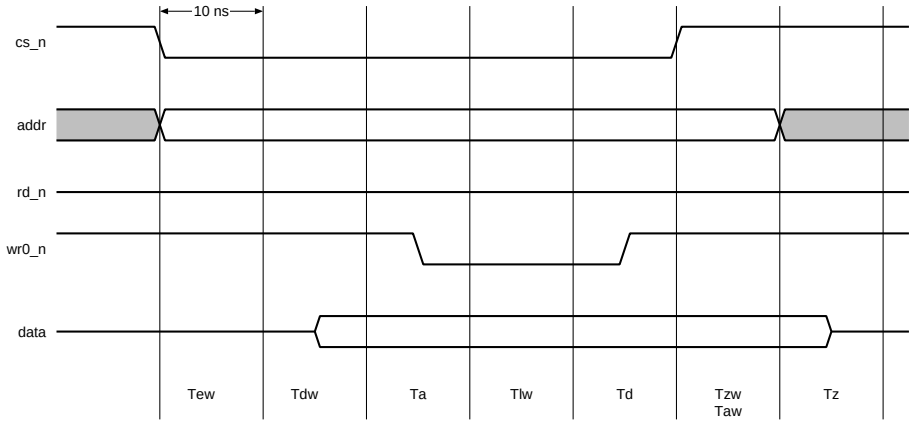


Figure 4.3: *Write cycle*

4.3.6.1 Early wait state (ew)

The early wait state is inserted before the activate state. The time added with early wait state is ew * 10 ns. The range for ew is 0-7.

4.3.6.2 Late wait state (lw)

The late wait state is inserted between the activate state and data state. The time added with late wait state is lw * 10 ns. The range for lw is 0-63.

4.3.6.3 Turn-off wait state (zw)

The turn-off wait state is inserted after the data state. The time added with turn-off wait state is zw * 10 ns. The range for zw is 0-7.

4.3.6.4 Address recovery wait state (aw)

The address recovery wait state is inserted after the data state. The time added with address recovery wait state is aw * 10 ns. The range for aw is 0-3.

4.3.6.5 Data setup wait state (dw)

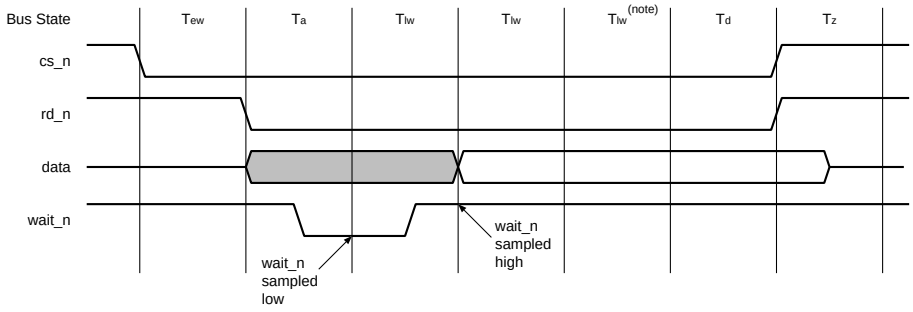
The data setup wait state is inserted between the early wait state and and the activate state or before the activate state if no early wait state is present. The time added with data setup wait state is dw * 10 ns. The range for dw is 0-3.

4.3.6.6 Early wait state burst (ewb)

The early wait state burst is inserted before the first early wait state in the first data cycle in a burst or before the first activate state in the first data cycle in a burst if no early wait state is present. The time added with early wait state burst is ewb * 10 ns. The range for ewb is 0-3.

4.3.6.7 External wait input

The external **wait_n** pin can be used by external devices to insert extra late wait states. The **wait_n** pin is sampled 30 ns before the end of the bus cycle. It can only be used if ew+lw >= 3. The bus interface adds late wait states until **wait_n** is deasserted. See figure 4.4 below.

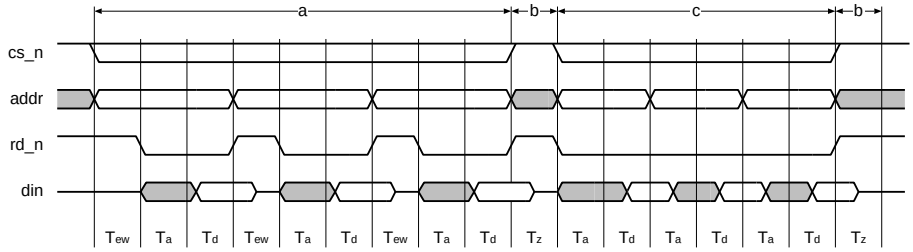


Note: Added late wait state due to wait_n asserted.

Figure 4.4: *External wait input example*

4.3.7 SRAM/Flash/peripheral timing

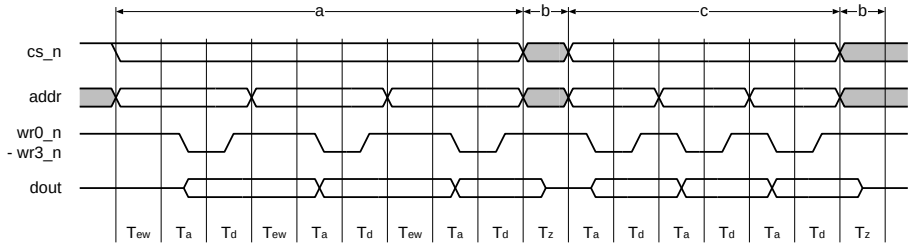
The bus cycles for SRAM/Flash/peripheral are controlled with wait states described in 4.3.6. See the timing diagram 4.5 for a read burst with and without early wait states:



(a) Burst with one early wait state.
(b) Turn-off cycle.
(c) Burst with zero early wait state.

Figure 4.5: *Read burst cycle*

See the timing diagram 4.6 for a write burst with and without early wait states.



(a) Burst with one early wait state.
(b) Turn-off cycle.
(c) Burst with zero early wait state.

Figure 4.6: *Write burst cycle*

4.3.8 Read and write modes

The read and write modes are used for units on the SRAM/flash and peripheral interfaces. The modes are controlled in register rw_grp1_cfg, rw_grp2_cfg, rw_grp3_cfg and rw_grp4_cfg.

4.3.8.1 Common write enable and bytewise write enable modes

The SRAM/flash/peripheral interface operates in two different modes: common write enable mode (cwe) and bytewise write enable mode (bwe).

Bytewise write enable uses four write enable signals in 32-bit mode and two write enable signals in 16-bit mode. This is the normal mode and the external pins are named after this mode.

Common write enable uses one common write enable (`cwe_n`) signal and four byte enable signals (`be0_n` - `be3_n`) in 32-bit mode and two byte enable signals (`be0_n` - `be1_n`) in 16-bit mode.

In common write enable mode, the byte enable signals and common write enable signal are mapped to the output pins according to the table below. The first column lists the pin name, the second column shows the signal to pin mapping for 32-bit mode, and the third column shows the mapping for 16-bit mode.

Pin name	(32-bit mode) cwe	(16-bit mode) cwe
wr0_n	`be0_n`	`be0_n`
wr1_n	`be1_n`	`be1_n`
wr2_n	`be2_n`	not used
wr3_n	`cwe_n` (common write enable)	`cwe_n` (common write enable)
a1	`be3_n`	`a1`
rd_n	`rd_n`	`rd_n`

Table 4.4: *Signal to pin mapping in common write enable mode*

The signals are described below.

rd_n Read strobe, common to all four bytes of the data bus. This signal is not active during SDRAM access.

wr0_n-wr3_n Write strobes, one for each byte in the data bus.

be0_n-be3_n Byte enable strobes, one for each byte in the data bus.

cwe_n Common write enable strobe, common to all four bytes of the data bus.

a1 Address bit 1, not used as address in 32-bit mode.

be0_n - be3_n have the same timing as the address signals. cwe_n has the same timing as the **wr0_n - wr3_n** signals.

4.3.8.2 Normal and extended write modes

In normal mode the **wr0_n - wr3_n** or cwe_n signals go high 5 ns before the end of the bus cycle. If set to extended write mode the **wr0_n - wr3_n** or cwe_n signals go high at the end of the bus cycle.

4.3.8.3 Normal and early read complete modes

In normal mode the shortest read pulse width is 20 ns. In early read complete mode the rd_n signal goes high one clock cycle before the end of the read cycle. The shortest read pulse width in early read complete mode is 10 ns.

See figure 4.7 for an example of a read cycle with early read complete, and no wait states.

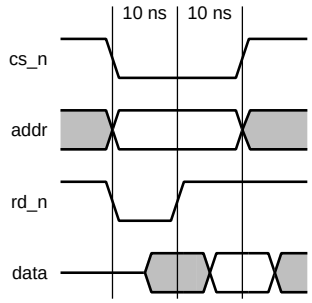


Figure 4.7: *Early read complete*

4.3.9 NAND flash

A NAND flash can be connected to the bus interface with its re_n and we_n signals connected to any two of the chip select signals **csp0_n** to **csp6_n**. The other control signals to NAND flash are connected to general I/O pins.

Figure 4.8 below, is example of a NAND flash connected to the bus interface.

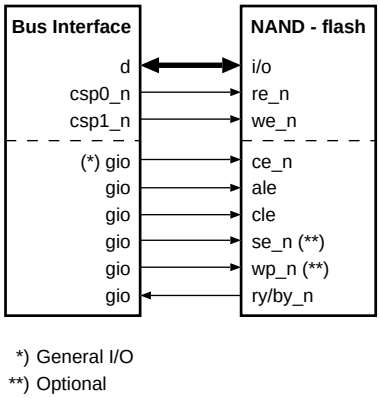


Figure 4.8: *NAND flash to bus interface connection*

The signals connected to general I/O are controlled via software. The read/write is controlled via reading/writing to the different memory areas associated with the respective chip select signals.

- **Example:** READ

```
10000000-13FFFFFF    64   csp0_n  Peripheral select 0 -> READ
14000000-17FFFFFF    64   csp1_n  Peripheral select 1 -> WRITE
```

1. ce_n -> low
2. cle -> high
3. read data CMD (write to 0x14000000)
4. cle -> low
5. ale -> high
6. write ADDR (3 writes to 0x14000000)
7. ale -> low
8. wait by_n
9. read data (0x10000000)
10. ce_n -> high

- **Example:** WRITE

```
10000000-13FFFFFF    64   csp0_n  Peripheral select 0 -> READ
14000000-17FFFFFF    64   csp1_n  Peripheral select 1 -> WRITE
```

1. ce_n -> low
2. cle -> high

3. input data CMD (write to 0x14000000)
4. cle -> low
5. ale -> high
6. write ADDR (3 writes to 0x14000000)
7. ale -> low
8. wait by_n
9. write data (0x14000000)
10. cle -> high
11. page program CMD (write to 0x14000000)
12. cle -> low
13. ce_n -> high

4.3.10 SDRAM interface

The SDRAM interface is configured with three registers rw_sdram_cfg_grp0, rw_sdram_cfg_grp1 and rw_sdram_timing. Commands to the SDRAM memory modules during initialization is controlled by rw_sdram_cmd.

The SDRAM interface clock frequency is 100 MHz.

4.3.10.1 Connecting the SDRAM

The SDRAM interface signals are mapped on the following bus interface pins.

Bus interface pins	SDRAM signals
a1 - a16	row, col address 16-bit mode
a2 - a16	row, col address 32-bit mode
a17	ba0 bank address
a18	ba1 bank address
a19 - a22	dqm0 - dqm3
a23	we_n
a24	cas_n
a25	ras_n
d0 - d31	d0 - d31
csd0_n - csd1_n	csd0_n - csd1_n
wr0_n	dqm4 (wide module mode)
wr1_n	dqm5 (wide module mode)
wr2_n	dqm6 (wide module mode)
a1	dqm7 (wide module mode)

Table 4.5: *SDRAM interface signal to bus interface pin mapping*

These examples use 2x16-bit SDRAMs. There is no difference if 8x4-bit, 4x8-bit or 1x32-bit is used.

The SDRAM banks are combined into two groups where each group is controlled by a

separate chip select signal. See figure 4.9.

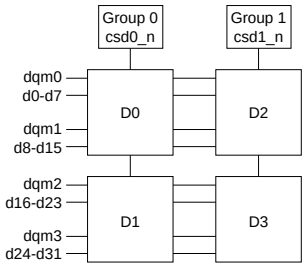


Figure 4.9: *2 rows x 16-bit SDRAMs*

When using 64-bit wide SDRAM modules, one chip select is assigned to each group and controls all 64-bits. The eight dqm signals are used to select the different bytes within the word. Upper and lower 32 bits are tied together. In other words, bit 0 and bit 32 (bit 1 and bit 33, bit 2 and bit 34, etc.) are tied together, and one at a time they drive the data bus. See figure 4.10.

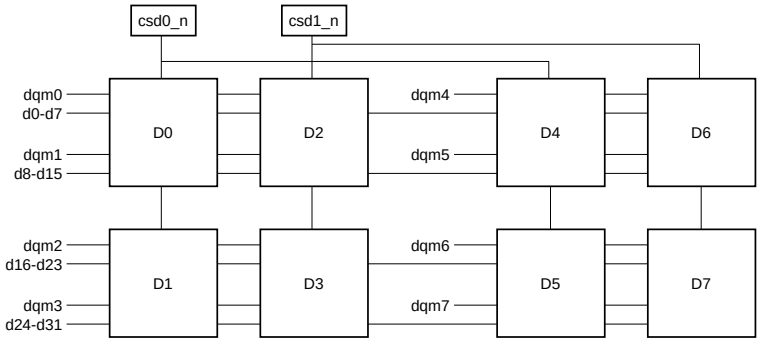


Figure 4.10: *2 rows 64-bit wide modules with 16-bit SDRAMs*

4.3.10.1.1 Address shift in 16-bit mode

It is possible to shift the address bits up one step in 16-bit mode. In 16-bit shifted address mode the internal `a16` address bit is not used. 16-bit mode is described in section 4.3.10.3.1.

Internal address signal	External pin in normal 16-bit mode	External pin in shifted 16-bit mode
`a1`	**a1**	**a2**
`a2`	**a2**	**a3**
...	...	...
`a15`	**a15**	**a16**
`a16`	**a16**	-

Table 4.6: *Address shift in 16-bit mode*

4.3.10.2 SDRAM timing parameters

SDRAM timing parameters are set in the register rw_sdram_timing. These timing parameters are common for both SDRAM groups.

The SDRAM timing parameters are measured in SDRAM bus cycles where one cycle is 10 ns.

The following configurations for SDRAM timing parameters are available for the SDRAM interface.

Refresh interval (ref) The selectable interval for SDRAM refresh is 15620 ns, 7800 ns and disable.

Row cycle time (rc) The refresh cycle time is rc + 6.

RAS precharge delay (rp) The time between precharging a bank and activating a row in the same bank. Number of delay cycles will be rp cycles.

RAS to CAS delay (rcd) After a row in a specific bank is activated, read and write operations can only be initiated on this activated bank after the minimum rcd cycles has elapsed. Number of delay cycles will be rcd cycles.

CAS latency (cl) The time from a read command until the data arrives. Number of latency cycles will be cl cycles.

Power save select (ps) Enter power save mode. SDRAM banks will enter power save mode immediately after each auto refresh cycle, and will stay in the power save mode until an SDRAM access or a new auto refresh cycle occurs.

Power down exit delay (pde) When the CKE signal goes high after a power down there is a penalty of one or two clock cycles before a new command can be issued. Number of delay cycles will be pde + 1.

4.3.10.3 SDRAM configuration

This configuration of RAS/CAS address size, wide module mode and bus width for the SDRAM interface is controlled via the registers rw_sdram_cfg_grp0 and rw_sdram_cfg_grp1, one for each group.

SDRAM banks are combined into two groups where each group can use either two-bank or four-bank chips. The following configurations are common for all banks:

Width Width selects either a 16-bit or 32-bit SDRAM bus width.

Group Select Mode The group select mode determines how to select a group of SDRAM banks. When using only one group, the group select mode value must be set to either 0 or 1.

Row Address Shift During activate bank commands, the row portion and the bank select bits of the address are shifted down to the lower address outputs to which the SDRAM address pins are connected.

Column Address Range The column address range determines how many address bits that are used in the column address.

Bank Type Mode Bank type mode selects either the two-bank or four-bank mode.

Bank Decode Mode The bank decode mode determines which address bit that selects between bank 0 and 1 in a group. In the four-bank mode, bank 2 and 3 will be selected by the next higher order address bit.

Wide Module Mode This mode supports 64-bit wide SDRAM modules where all 64-bits are controlled by the same chip select signal. dqm0 to dqm7 are used to control the individual bytes within both groups of SDRAM banks.

· Example: Address output in 32-bit mode during RAS and CAS cycles (Figure 4.11.

In this example the setting of the fields in rw_sdram_cfg_grp0 are:

Field	Value
bank_sel	bit22
ca	10
type	bank4
bw	bw32
sh	2

Table 4.7: *Example values*

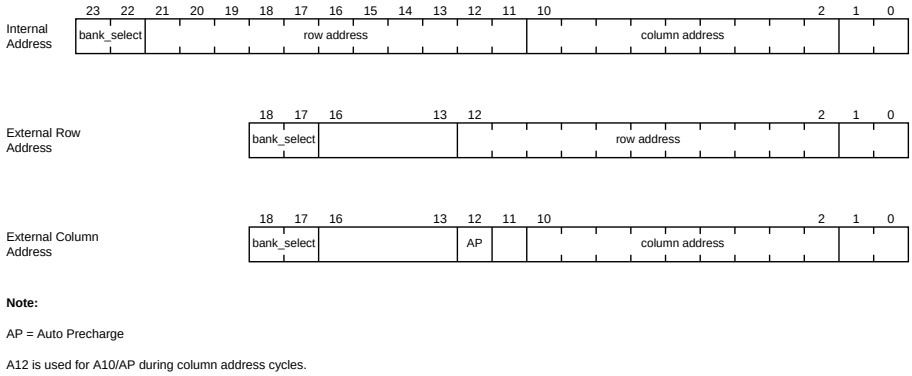


Figure 4.11: *SDRAM address output example*

4.3.10.3.1 16-bit mode

In 16-bit mode the row address is shifted down to address pin **a1** and the column address start at **a1**. The bank address signals `ba0` and `ba1` are still on address pins **a17** and **a18**.

Auto precharge (A10/AP) bit is on address pin **a11** in column address cycles.

In 16-bit mode it is possible to shift the address pins up one bit as described in section 4.3.10.1.1. If this is done the Auto precharge (A10/AP) bit is on address pin **a12** in column address cycles.

The 16-bit mode is controlled in rw_sdram_cfg_grp0 and rw_sdram_cfg_grp1. The bw field selects 16-bit or 32-bit bus width, and the sh16 field selects normal or shifted 16-bit mode.

4.3.10.4 SDRAM Power up and initialization commands

SDRAM commands are issued through the register rw_sdram_cmd. The SDRAM commands are:

- NOP (nop)
- Precharge All (pre)
- Auto Refresh (ref)
- Mode Register Set (mrs)

These commands are used during SDRAM power up and initialization.

4.3.10.5 Power up and initialization

The SDRAM interface is disabled after reset. It is started by the first write to the rw_sdram_timing register.

The SDRAM memories have an internal configuration register that must be written to with the Mode Register Set command during initialization.

The explicit SDRAM commands should only be used during power up and initialization of the SDRAM banks. A typical SDRAM initialization sequence is:

1. Configure the banks by writing to rw_sdram_cfg_grp0 and rw_sdram_cfg_grp1.
2. Configure the SDRAM timing parameters and enable the master clock by writing to rw_sdram_timing. Set the ref to off.
3. Issue NOP command by writing to the cmd field in rw_sdram_cmd.
4. Wait for 200 μs.
5. Issue Precharge All command by writing to the cmd field in rw_sdram_cmd.
6. Wait 80ns.
7. Issue Auto Refresh command by writing to the cmd field in rw_sdram_cmd.
8. Wait 80ns. Repeat step 7-8 seven more times, totalling eight times.
9. Issue Mode Register Set command by writing to the cmd and mrs_data fields in rw_sdram_cmd. The burst length shall be set to one.
10. Wait 80ns.
11. Configure the refresh time parameter ref field in rw_sdram_timing.

4.3.10.6 SDRAM self refresh mode

The SDRAM can be entered into self refresh mode to save power when the system is idle. While the SDRAM is in self refresh mode, accesses to the SDRAM area are not allowed.

To enter self refresh mode, slf should be written to the cmd field in register rw_sdram_cmd. The SDRAM interface will then deassert the **sdcke** output and issue the self refresh command to the SDRAM. To exit the self refresh mode, slf should again be written to cmd in register rw_sdram_cmd. This will assert the **sdcke**, and the SDRAM resumes normal operation. See figure 4.12.

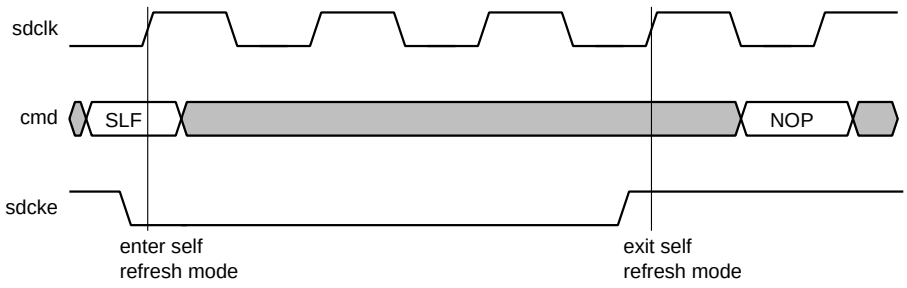


Figure 4.12: *SDRAM self refresh mode*

4.3.10.7 PLL bypass mode

When the chip PLL is disabled the SDRAM controller must still generate refresh cycles often enough to ensure that the 15.625 μs or 7800 ns refresh interval is met. This is controlled by setting cpd in rw_sdram_timing to yes when the PLL is bypassed. When turning the PLL on again, the missed refresh cycles during the PLL lock time can be compensated by keeping the cpd field set to yes for at least 25 μs after the PLL lock time has expired.

If the software does not need to access the SDRAM during PLL bypass mode, an alternative is to use the self refresh mode to maintain data retention.

4.3.11 SDRAM timing

In the diagrams 4.13, 4.14, 4.15 and 4.16 below, cmd denotes the signals `ras_n`, `cas_n` and `sdwe_n`, which are output on pins **a25** to **a23**.

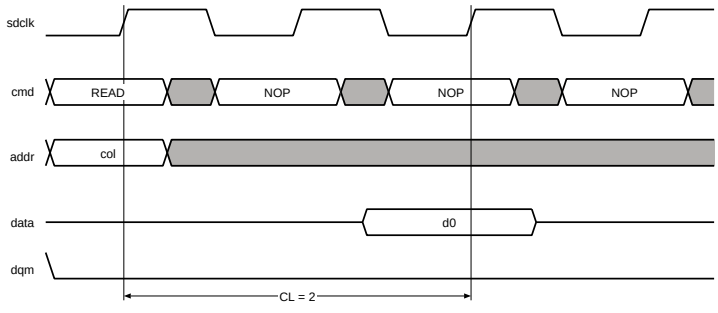


Figure 4.13: *Read burst, CAS latency 2*

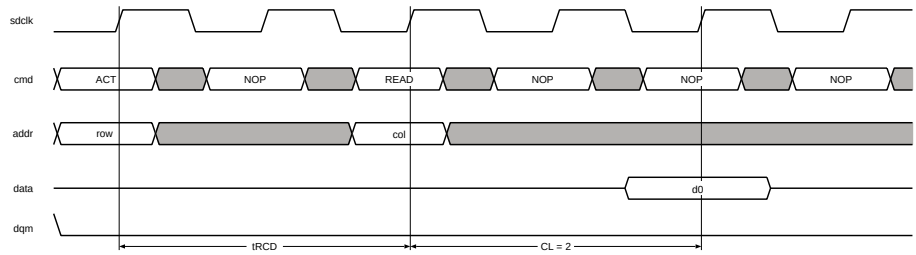


Figure 4.14: *Activating a bank and a read to that bank, CAS latency 2*

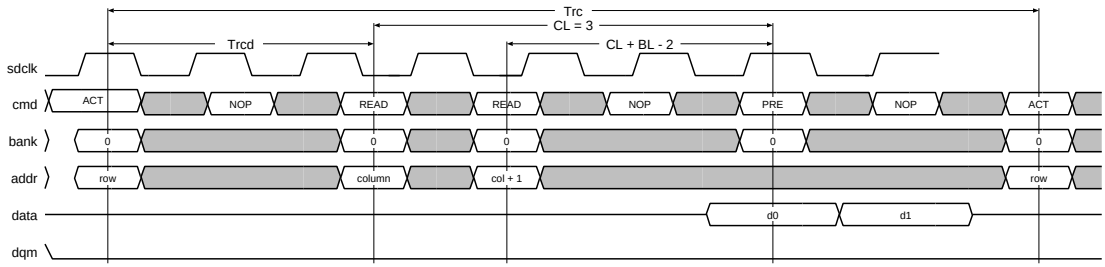


Figure 4.15: *Activating a bank, two reads to that bank then precharge that bank, CAS latency 3*

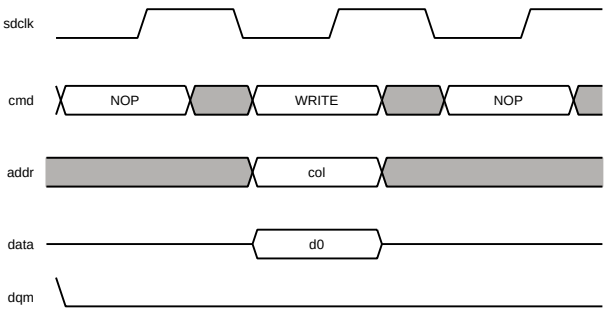


Figure 4.16: *SDRAM write*

4.3.12 Bus arbitration interface

4.3.12.1 Overview

The bus interface implements a bus arbitration protocol that allows several different units to arbitrate and gain access to one shared external bus.

The figure 4.17 below shows an example system with four units sharing the bus:

A unit in this system may be an instance of the bus interface or any other chip or sub-system that participates in the bus arbitration.

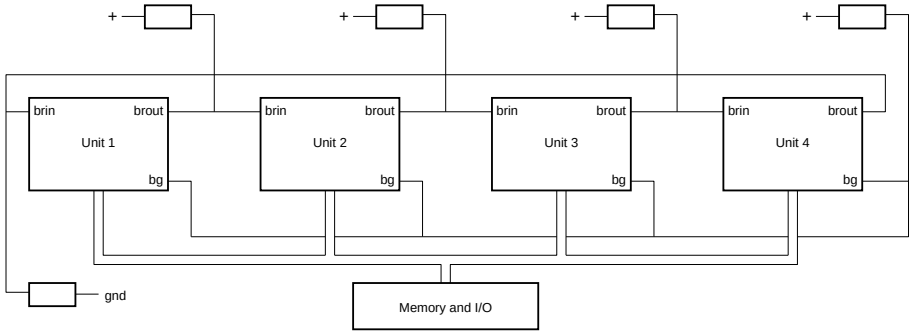


Figure 4.17: *Four units sharing the same system bus*

4.3.12.2 Bus arbitration interface mode registers

The mode registers for the bus arbitration and the slave mode interface are combined together and specified in 25.6.

There are six mode registers for the bus arbitration interface:

1. rw_arb_cfg: Configuration register.
2. r_arb_stat: Status register.
3. rw_intr_mask: Interrupt mask register.
4. rw_ack_intr: Interrupt acknowledge register.
5. r_intr: Interrupt register.
6. r_masked_intr: Masked interrupt register.

4.3.12.3 Arbitration signals

The bus interface has three three dedicated signals for the bus arbitration. These are:

1. **brin**: Bus request input.
2. **brout**: Bus request output.
3. **bg**: Bus grant input/output. This signal is driven by the current bus master. The **bg** signal will need a pull-up to keep it stable during the hand-over between different bus masters.

In a typical application, the **brout** pin of one chip is connected to the **brin** pin of the next chip circularly, and all **bg** pins are connected together. This will result in a round-robin priority scheme. Other priority schemes can be implemented if the arbitration pins are instead connected to an external arbiter.

The default polarity of the bus arbitration signals is active high, but the polarity of each signal can be configured separately through the rw_arb_cfg register. It is also possible to control the **brout** and **bg** outputs directly through the rw_arb_cfg register, and to read back the value on all three pins via the r_arb_stat register.

4.3.12.4 Arbitration protocol

This section specifies the arbitration protocol supported by the bus interface. The requirements in this section should be fulfilled by all units that participate in the bus arbitration.

4.3.12.4.1 Bus request

A slave asserts its **brout** when it wants to get access to the bus, or when it sees its **brin** active. Thus, a bus request anywhere in the chain will ripple all the way down to the current bus master. The slave must not activate its **brout** because of an internal bus request after it has detected **bg** active, but must always forward incoming requests from **brin** to **brout**.

4.3.12.4.2 Bus grant

The current bus master keeps its **brout** inactive and keeps **bg** inactive as long as it holds the bus. When the master releases the bus, it asserts **bg** for 3-20 ns and then sets it to high-z. Now the slave that has its **brin** inactive and its **brout** active takes over the bus and becomes the new bus master.

4.3.12.4.3 Start-up

At system reset, the units participating in the bus arbitration have to start up in a well defined order. During reset, all units put their **brout** and **bg** signals in high-z. One unit, the initial bus master, has a pull-down on its **brin**. The other units, the initial slaves, have pull-up resistors on their **brin** signals.

After reset, the initial bus master drives its **brout** and **bg** low. The bus release mechanisms should initially be disabled in all units, and have to be enabled by software. Once a unit has acquired the bus it will not release it unless the software has changed the bus release mode. This makes it possible to start the units one by one in a controlled way.

To decide whether to be the initial master or an initial slave, the unit has to look at both **brin** and **bg**. It becomes the initial master only if it sees **brin** low and **bg** high when reset is released.

4.3.12.4.4 Bus release timing

This timing specification relates to a master that releases the bus and becomes a slave. See figure 4.18.

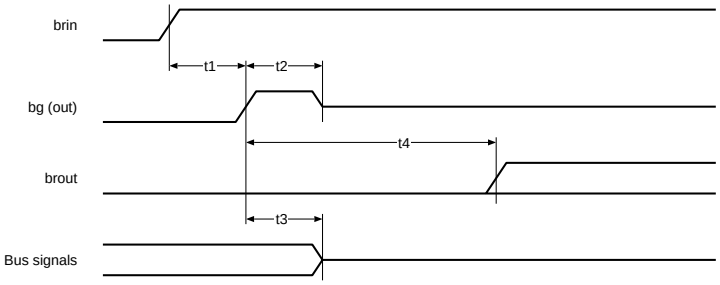


Figure 4.18: *Bus release timing*

Symbol	Explanation	Min	Max	Unit
t1	**brin** to **bg** delay.	0	-	ns
t2	**bg** turn off time.	3	20	ns
t3	Bus turn off time from **bg**.	-	20	ns
t4	**brout** inactive hold time from **bg**.	83	-	ns

Table 4.8: *Bus release timing*

4.3.12.4.5 Bus acquirement timing

This timing specification relates to a slave that acquires the bus and becomes the new master. See figure 4.19.

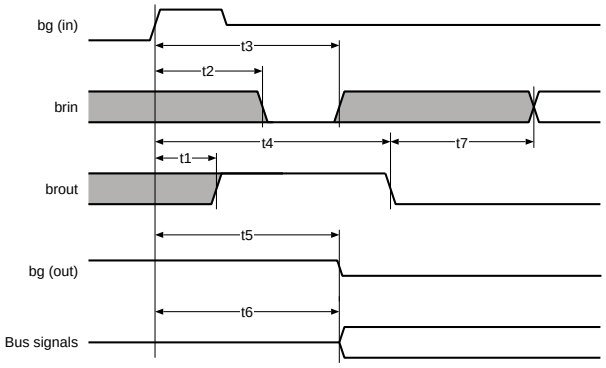


Figure 4.19: *Bus acquirement timing*

Symbol	Explanation	Min	Max	Unit
t1	**brout** valid from **bg** (latest time a slave may assert **brout** because of its own bus request. Forwarding of requests on **brin** may occur later).	-	30	ns
t2	**brin** valid from bg (earliest time a slave may sample **brin** to become the new master).	50-c[2]	-	ns
t3	**brin** hold time from **bg** (latest time a slave may sample brin to become the new master).	-	80	ns
t4	**brout** hold time from **bg**.	80	120	ns
t5	**bg** enable time from **bg**.	50-c [2]	100	ns
t6	Bus enable from **bg**.	50-c [2]	-	ns
t7	**brin** valid from **brout** inactive (earliest time the new master is allowed to take notice of a new bus request).	-	50	ns

Table 4.9: *Bus acquirement timing*

4.3.12.4.6 Request forward timing

Request forward timing, figure 4.20, relates to a slave that forwards a bus request from another slave.

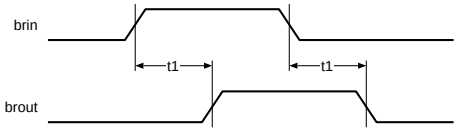


Figure 4.20: *Bus request forward timing*

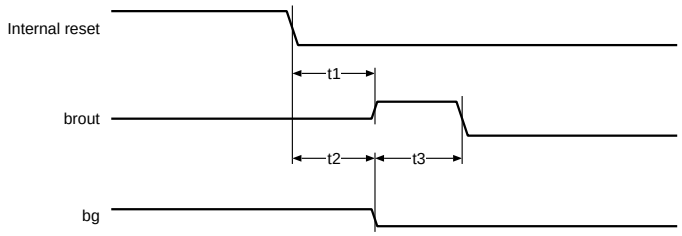
Symbol	Explanation	Min	Max	Unit
t1	**brin** to **brout** delay.	-	10	ns

Table 4.10: *Request forward timing*

[2]The parameter c is 0 by default,but the protocol allows it to be configurable between 0 and 30 ns. This makes it possible to optimize the bus hand-over timing depending on the real delays in the system. In a system with only two units, the slave will always see a valid (inactive) **brin**. Therefore, with only two units it works with c = 30 ns. In the bus interface, this can be configured by the settle_time field in the rw_arb_cfg register, see also 4.3.12.6 below.

4.3.12.4.7 Initial master start up timing

See figure 4.21.



Note: Internal reset is shown active high.

Figure 4.21: *Initial master start up timing*

Symbol	Explanation	Min	Max	Unit
t1	Internal reset to **brout** driven.[3]	0	-	ns
t2	Internal reset to **bg** low	0	-	ns
t3	**bg** low to **brout** low.[4]	30	-	ns

Table 4.11: *Initial master start up timing*

4.3.12.5 Bus release modes

The bus release behavior can be configured in the release field of the rw_arb_cfg register. The following alternatives are available:

- Don't release the bus (default after reset)
- Release the bus at the end of the current bus burst
- Release the bus when the bus interface becomes idle

4.3.12.6 Arbitration settle time and bus acquirement time

The arbitration protocol specified in 4.3.12.4 requires a minimum time from when the old master releases the bus until the new master acquires the bus, to allow the arbitration to settle.

The maximum allowed arbitration settle time in the system, and thus also the minimum time before a new master may acquire the bus, can be configured in the settle_time field of the rw_arb_cfg register. The configurable range is 20 - 50 ns. A shorter settle time will give faster switch over between bus masters, but will on the other hand limit the

[3] t1 is allowed to be longer than t2 + t3. In that case, **brout** will go directly from high-z to 0.

[4] This minimum time is there to ensure that no unit falsely identifies itself as the initial master because of different internal reset timing in the different units.

number of units that can participate in the bus arbitration. The initial value for the settle time is 50 ns.

The actual bus acquirement time will be in the range of 0-20 ns longer than the configured settle time, depending on the relative clock skew between the old and the new master.

4.3.12.7 Bus acquirement modes

There are two modes for when the bus interface tries to acquire the bus. The default mode is to only try to acquire the bus if there is a pending memory access to be performed.

The interface can be configured to arbitrate for the bus even if there is no memory access pending. This mode can be useful e.g. in systems where timing critical applications run from the internal RAM and the external bus may be occupied by another master for a long time.

The bus acquirement mode is configured in the acquire field of the rw_arb_cfg register.

4.3.12.8 SDRAM control in slave mode

The bus interface can be configured to either switch over SDRAM control between bus masters, or to maintain the control of the SDRAM while in slave mode. In the latter case, the slave will continue to generate the SDRAM clock during slave mode, and drive other SDRAM specific signals to their inactive state. For details on the different signals, see 4.4.1.

4.3.12.9 Bus release and acquirement detection

The mode field in r_arb_stat indicates whether the bus interface is in master or slave mode.

The bus arbitration also generates the bus_release and bus_acquire interrupts. The bus_release interrupt is generated whenever the bus is released to another master, and the bus_acquire interrupt is generated when the bus is re-acquired.

4.3.13 External DMA

4.3.13.1 General

The external DMA operates in a so-called ”pseudo DMA” fashion. This means that the memory accesses are separated from the I/O accesses of a DMA transfer, and the data is temporarily stored in the DMA FIFOs. There are two input channels and two output channels. Each external DMA channel is connected to an internal DMA channel. The relation between external and internal DMA channel numbers is described in 10.

The external DMA channels are:

ext_dma0 Output channel

ext_dma1 Input channel

ext_dma2 Output channel

ext_dma3 Input channel

4.3.13.2 External DMA bus width

The external DMA can operate with a bus width of 8, 16 or 32 bits. With 8-bit and 16-bit bus width, the least significant bits of the data bus are used.

With common write enable mode and 32-bit external DMA transfers in master mode, the group of memory banks used by the DMA cycles must be configured for 32-bit bus width. The configured bus width of the used memory bank group does not matter for 8-bit or 16-bit external DMA transfers, or when bytewise write enable mode is used.

4.3.13.3 External DMA burst length

Each external DMA channel can be individually configured for a bus burst length of one or eight bus cycles.

4.3.13.4 External DMA handshake signals

The bus interface has eight handshake signals that are shared between the external DMA and the slave mode interface. The signals can be configured to different functions depending on whether the chip is in master or slave mode.

When the pins are controlled by the external DMA, each pin can be configured as DMA request, DMA acknowledge, terminal count input or terminal count output. Each pin can be configured to any of the functions for any of the channels, see 25.5. For detailed timing of the signals refer to section 4.4.3.3.

dreq DMA request input. Each channel can be configured to take its DMA request from any of the eight handshake signals, or to take it from the configuration mode register for the channel. The polarity of the `dreq` can be configured as well. When `dreq` is active, the associated external DMA channel arbitrates for the external bus. The channel will continue to issue new bus bursts as long as `dreq` is active. The external unit can pause the DMA transfer by setting `dreq` inactive.

dack (master mode only) DMA acknowledge output. The `dack` output is activated at the beginning of the external DMA bus burst, and deactivated at the end of the burst. The polarity of the `dack` can be configured to active high or active low. DMA acknowledge may not be needed in all applications, since a chip select output can be used instead.

tc_in Terminal count input. If enabled, an active signal on `tc_in` causes the associated external DMA channel to stop. If the channel is in continuous transfer mode,

see section 4.3.13.8 below, tc_in will not stop the channel, but will generate an interrupt, and will result in end of packet in an input channel. The polarity of the tc_in can be configured to active high or active low.

An inactive to active transition on tc_in will take action immediately (after completion of the ongoing bus burst, if any). If tc_in is kept constantly active, it will take action after the first burst of each transfer. A special case occurs if the unit alternates between master mode and slave mode, and the pin selected for tc_in is not configured to maintain the same function in both modes. In this case, the terminal count signal in to the external DMA channel will be turned off while the chip is in the opposite bus mode to what the channel is configured for. The result will be that with a constantly active tc_in (or with tc_in_mode set to force), the change of bus masters in the system may cause tc_in to take effect even though no DMA transfer has occured.

tc_out Terminal count output. The polarity of the tc_out can be configured to active high or active low. The tc_out signal becomes active in four cases:

1. The tc_out will be active during a bus burst in which the transfer counter will expire. The tc_out is active during the whole burst.
2. The tc_out will become active at the start of the last bus cycle of a transfer that is terminated by an end of packet from the internal DMA (output channels only). The tc_out stays active until the end of the burst.
3. The tc_out becomes active when tc_in is activated. If tc_in is constantly active, tc_out becomes active at the first burst of each transfer. If tc_in occurs within a burst, tc_out will be active until the end of the burst. When tc_in is activated outside a burst, tc_out will be active for 20 ns.
4. The tc_out becomes active when the channel is stopped by software. The tc_out will remain active until the end of the ongoing bus burst. If there is no ongoing burst, tc_out will be active for 20 ns.

4.3.13.5 External DMA address

The I/O cycles of the DMA transfers are directed to a constant, configurable address. The wait states and cycle behavior are the same as for the normal memory accesses to the same address. Each of the channels has its own address register, rw_ch0_addr - rw_ch3_addr. Bits 31, 30, 1 and 0 of the address are not used, since external DMA always performs 32-bit aligned accesses to non-DRAM non-cached area.

The external DMA address is not used when the external DMA channel is used in slave mode.

4.3.13.6 Transfer counter

Each external DMA channel has a 16-bit transfer counter. The counter is initialized to the configured start value when the channel is started, and counts down with one for each external DMA cycle. An external DMA cycle is one, two or four bytes in size, depending on the configured bus width. Note that writing yes to run in rw_ch0_start will re-initialize the counter even if the channel is already running.

When the transfer counter counts down to zero, it stops the channel, generates an interrupt and, for input channels, signals end of packet to the internal DMA.

In the continuous transfer mode, the counter is re-initialized with the start value immediately after it counts down to zero, and the DMA transfers continue. The interrupt is still generated, and the end of packet is set for an input channel. While in continuous transfer mode, a `tc_in`, or for out channels an end of packet from the DMA channel, will also cause the transfer counter to be re-initialized.

The transfer counter can also be configured to not cause any actions at all (including not being re-initialized) when it reaches zero. The transfer counter will still be re-initialized when the channel is started, or when a `tc_in` is received or an end of packet is signalled from an out DMA channel in continuous transfer mode.

4.3.13.7 Bus burst behavior in 8-cycle burst mode

With 8-cycle burst length, the transfer counter may expire in the middle of a bus burst.

For in channels, end of packet will be set immediately, resulting in a packet length corresponding to the transfer counter start value. The bus burst will however run to completion before the channel is stopped.

The end_discard field in register rw_ch1_ctrl or rw_ch3_ctrl selects whether the excess data will be discarded or forwarded to the internal DMA. If the excess data is received, and the channel is in continuous transfer mode, the transfer counter will be re-initialized immediately. If the transfer counter expires a second time within the same burst, the data coming in after the second expiration will be lost. If the excess data is received and the channel is not in continuous transfer mode, the transfer counter will not be re-initialized, but will instead wrap around and continue. The transfer counter will only count cycles that result in data written to the internal DMA.

For out channels, the end_pad field in register rw_ch0_ctrl or rw_ch2_ctrl selects whether the remaining cycles in a burst, after the expiration of the transfer counter, will be read out from the internal DMA or will be padded with 0's. An end of packet from the internal DMA in the middle of a bus burst will always cause padding with 0 for the rest of the burst. If the channel is in continuous transfer mode, and excess data is read from the internal DMA, the transfer counter will be re-initialized immediately. If excess data is read from the internal DMA, and the channel is not in continuous transfer mode, the transfer counter will not be re-initialized, but will instead wrap around and continue. The transfer counter will only count cycles that result in reading data from the internal DMA.

For out channels, a burst will only be started if there is either enough data for a full burst available from the internal DMA, or the internal DMA signals ends of packet.

4.3.13.8 Start and stop of external DMA transfers

The channel will pause without stopping for the following reasons:

- dreq is inactive
- Internal DMA FIFO full (input channels) or FIFO empty (output channels)

Each channel may be started and stopped by the run field in the rw_ch0_start ... rw_ch3_start register. The channel will stop if the run field is cleared. Clearing the run field will also generate an interrupt and, for input channels, signal an end of packet to the internal DMA. The run field can be cleared by the software, or the external DMA interface can clear it for the following reasons:

- transfer counter expired
- tc_in is activated
- Internal DMA channel signals end of packet (output channels only)

Note that stopping the channel by software or by using tc_in may result in a tc_out signal that occurs after the last access rather than during it.

The rw_ch0_start ... rw_ch3_start register will go to the stopped state immediately when a stop condition occurs. This may happen before the last bus burst is completed. To see if the channel is really stopped, the run field of the r_ch0_stat ... r_ch3_stat register should be used instead.

4.3.13.9 Continuous transfer mode

Each external DMA channel can be individually configured to stop and clear the run bit of the rw_ch0_start ... rw_ch3_start register at end of transfer, or to just generate interrupt, tc_out and end of packet at end of transfer. End of transfer is defined as any of the following events:

- transfer counter expired
- tc_in is activated
- Internal DMA channel signals end of packet (output channels only)
- run field cleared by software

Clearing the run field by software will stop the channel even if it is in continuous transfer mode.

4.3.13.10 Interrupts

The external DMA interface can generate the following interrupts:

ext_dma0 Interrupt from external DMA channel 0.

ext_dma1 Interrupt from external DMA channel 1.

ext_dma2 Interrupt from external DMA channel 2.

ext_dma3 Interrupt from external DMA channel 3.

Each channel generates an interrupt when the transfer counter expires, when an end of packet is signalled, when a `tc_in` is received or when the software stops the channel.

If an interrupt event occurs during an ongoing DMA bus burst for the interrupting channel, the interrupt will be delayed until the burst is completed.

The interrupts are cleared by writing to the rw_ack_intr mode register. All interrupts can be masked through the rw_intr_mask register. The masked and non-masked interrupts can be read from the r_masked_intr and r_intr registers.

4.3.13.11 Priority between external DMA channels

The external DMA channels all have equal priority. A round-robin arbitration mechanism is used.

4.3.13.12 Rate control

The external DMA interface contains a rate control mechanism that gives the possibility to set a bandwidth limit for selected external DMA channels, and/or to stream out data with a defined rate. The external DMA has one rate control input common to all four channels. The rate control input is pin **pa7**.

If rate control is enabled for a channel, the channel will only perform a burst if the rate control input is high or if there has been a positive pulse on the rate control input since the last burst. The channel will still require an active `dreq` to run a burst, unless it is configured to ignore `dreq`.

4.3.14 Slave mode operation

4.3.14.1 Overview

The slave mode interface offers a method for an external bus master to access the internal RAM within the slave[5]. While in slave mode, the bus interface looks like an I/O device to the external bus master. The communication takes place over four channels, where each channel is reached through a set of five registers accessible over the external bus. See figure 4.22.

4.3.14.2 Slave channels

There are four unidirectional slave channels, two in and two out:

1. slave_ch0: Output channel
2. slave_ch1: Input channel
3. slave_ch2: Output channel

[5]Internal mode registers may also be accessed, provided that no other unit performs any mode register accesses concurrently. Other units that could access the mode registers are the internal CPU and the I/O processor.

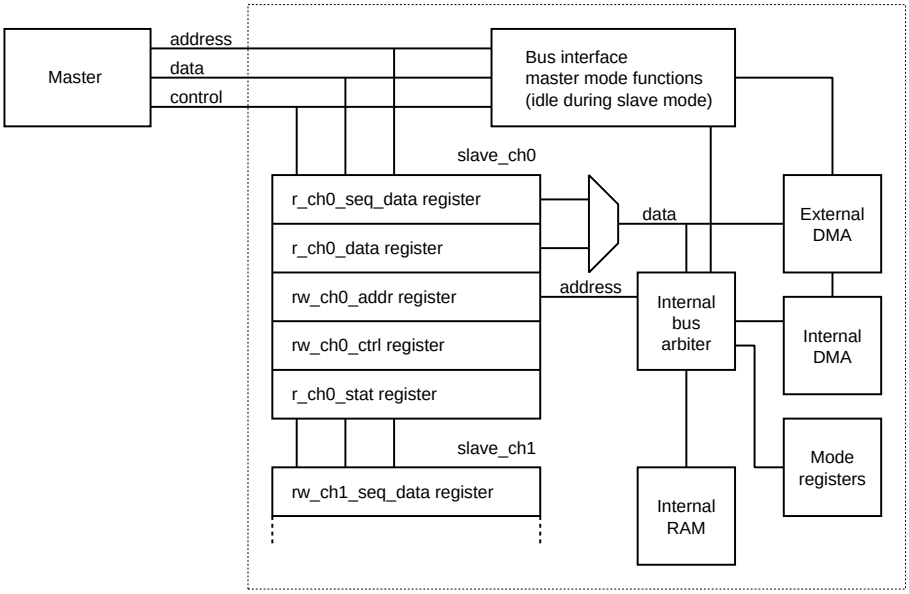


Figure 4.22: *Slave mode operation overview*

4. slave_ch3: Input channel

Each slave channel can operate in two different modes:

1. Address register mode
2. DMA mode

4.3.14.3 External slave mode registers

Each slave channel has five 32-bit registers that can be accessed by an external bus master. These registers are not accessible as mode registers from inside the slave. The external slave mode registers are described in more detail in 25.7.

The external slave mode registers, and the address map for them as seen from the master, are as follows:

```
slave_ch0:

offset + 0x00: r_ch0_seq_data
offset + 0x04: r_ch0_data
offset + 0x08: rw_ch0_addr
offset + 0x0c: rw_ch0_ctrl and r_ch0_stat

slave_ch1:
```

```
offset + 0x10: rw_ch1_seq_data
offset + 0x14: rw_ch1_data
offset + 0x18: rw_ch1_addr
offset + 0x1c: rw_ch1_ctrl and r_ch1_stat

slave_ch2:

offset + 0x20: r_ch2_seq_data
offset + 0x24: r_ch2_data
offset + 0x28: rw_ch2_addr
offset + 0x2c: rw_ch2_ctrl and r_ch2_stat

slave_ch3:

offset + 0x30: rw_ch3_seq_data
offset + 0x34: rw_ch3_data
offset + 0x38: rw_ch3_addr
offset + 0x3c: rw_ch3_ctrl and r_ch3_stat
```

Where `offset` is the start address for the external slave mode registers. The offset can either be set completely by address decoding external to the slave, or the slave can set bits 6 - 8 of the offset in an internal mode register, see section 4.3.14.9.

4.3.14.4 Internal slave mode registers

The internal mode registers for the bus arbitration and the slave mode interface are combined together and specified in 25.6.

There are six mode registers for the slave mode operation:

1. rw_slave_cfg: General configuration register.
2. r_slave_mode: Operation modes for the channels.
3. rw_ch0_cfg: Configuration register for slave_ch0.
4. rw_ch1_cfg: Configuration register for slave_ch1.
5. rw_ch2_cfg: Configuration register for slave_ch2.
6. rw_ch3_cfg: Configuration register for slave_ch3.

4.3.14.5 Slave chip selects

The slave mode registers are selected by the **css_n** signal, which is an input in slave mode. If **css_n** is connected to **css_n** of the master, the slave mode register start address (not considering slave identification bits described in 4.3.14.9) will be 0xac000000.

There is a possibility to configure the signals **csp0_n** - **csp3_n** as chip selects for the r_ch0_seq_data - rw_ch3_seq_data registers, without further address qualification. The slave identification bits are ignored when these chip selects are used. The use of **csp0_n** - **csp3_n** is configured in the data_cs fields of internal mode registers rw_ch0_cfg - rw_ch3_cfg.

4.3.14.6 Address register mode

In the address register mode, the external bus master should first write to the address register for the channel, rw_ch0_addr ... rw_ch3_addr, to set the internal address to be accessed within the slave.

The slave channel can be configured to either increment its address register by 4 for each access to the sequential data register r_ch0_seq_data ... rw_ch3_seq_data, or to keep the address register constant.

For an output channel, the write to the rw_ch0_addr or rw_ch2_addr register initiates an internal read within the slave, and the result is placed in the data register (r_ch0_data or r_ch2_data) and in the sequential data register (r_ch0_seq_data or r_ch2_seq_data) of the channel. When the internal read cycle is completed, the dav field in the status register (r_ch0_stat or r_ch2_stat) is set, to indicate to the external master that the data is valid. The external master can now read the data either from the data register or from the sequential data register of the channel. In the latter case, the read will initiate a new internal read within the slave.

For an input channel, a write to the the data register (rw_ch1_data or rw_ch3_data) or to the sequential data register (rw_ch1_seq_data or rw_ch3_seq_data) initiates an internal write within the slave. A write to the sequential data register will increment the address register rw_ch1_addr or rw_ch3_addr by 4 if the channel is configured for address increment, whereas a write to the (plain) data register will never increment the address register. When the master writes to the data register or the sequential data register of the channel, the data_rdy field in the status register (r_ch1_stat or r_ch3_stat) is cleared. The data_rdy field is set again when the internal write is completed.

Bit 31 in the address register is not used in the internal memory map. Instead, this bit is used to select whether the cache coherence mechanism of the internal memory arbiter, see 14, will be applied or not. Normally the slave mode accesses to e.g. internal RAM will participate in the cache coherence protocol, but this can be prevented by setting bit 31 of the address to 1.

Bit 1 and 0 in the address register will not be used by the slave interface, but are present in the address register.

4.3.14.7 DMA mode

In the DMA mode, the slave channel is connected to an external DMA channel within the slave. The slave_ch0 can be connected to ext_dma0, slave_ch1 to ext_dma1 and so on.

The bus width and burst length configurations of the external DMA channel are effective in this mode, as well as the `tc_in`, `tc_out`, transfer counter etc.. The external DMA address and `dack` configurations of the external DMA interface are not applicable in slave mode.

In slave mode operation, the actual DMA request signal in the system is the `sreq` signal going from the slave to the master, see 4.3.14.8. The `dreq` functionality of the external DMA channel can still be used, but the only function of it is that `dreq` (either from a pin or permanently set in the configuration register) must be active to allow a transition

from inactive to active state on the sreq of the channel.

For an output channel, data is read from the sequential data register (r_ch0_seq_data or r_ch2_seq_data). A read from this register will read out 1, 2 or 4 bytes from the associated internal DMA channel, depending on the configured bus width of the external DMA channel. The address register of the channel will be incremented by 4 if address increment is configured, but the address will not be used internally in the DMA mode. Data can also be read from the data register (r_ch0_data or r_ch2_data), but this will not cause the internal DMA to advance to the next data and will not cause any address increment. The dav field in the status register of the channel (r_ch0_stat or r_ch2_stat) will go inactive when a data read burst is started, but will be set active again after the burst as soon as there is enough data available from the internal DMA for a new complete bus burst of the configured burst length.

For an input channel, data is written to the sequential data register (rw_ch1_seq_data or rw_ch3_seq_data). This will enter 1, 2 or 4 bytes of data into the associated internal DMA channel, depending on the configured bus width of the external DMA channel. The address register of the channel will be incremented by 4 if address increment is configured, but the address will not be used internally in the DMA mode. Data can be written to the data register (rw_ch1_data or rw_ch3_data) instead. This will have the same effect as writing to the sequential data register, except that the address register will not be incremented. The data_rdy field in the status register of the channel (r_ch1_stat or r_ch3_stat) will go inactive when a data write burst is started, but will be set active again after the burst as soon as the internal DMA is ready to receive a new complete bus burst of the configured burst length.

4.3.14.8 External handshake pins

There are eight handshake pins, **hsh0** - **hsh7**, that are shared between the external DMA interface and the slave mode interface. When configured for use by the slave mode interface, these pins will output sreq (slave request) signals, which reflect the status of the dav or data_rdy fields in the status registers r_ch0_stat - r_ch3_stat.

Any of the two data_rdy and the two dav bits can be configured to be output as an sreq signal on any of the eight handshake pins. The polarity of each pin is also configurable. Configuration of the sreq signals is done in the external DMA interface registers rw_pin0_cfg - rw_pin7_cfg.

4.3.14.9 Slave identification

The slave mode interface can be configured to have a 3-bit slave identification number. When the slave identification number is enabled, the identification number is compared with address bits **a6** - **a8** to qualify an access from the master to the slave mode registers. Thus, up to 8 different slaves can be separately addressed without any external address decoding. The slave identification number is configured in the rw_slave_cfg register.

The slave identification number is disabled after reset. There are different possible methods for slave identification resolution after reset, depending on how the system is designed. Here is one example, where all chips boot up from the same flash PROM:

1. The initial master after reset starts to execute code from the flash PROM. It knows that it is the initial master by reading init_mode, and therefore sets its own slave ID to 0. It also writes its slave ID (= 0) to a predefined position in the external RAM. Thereafter it enables its bus arbitration.
2. The next chip in line will now win the bus and start to execute from the flash PROM. It knows that it was an initial slave by reading init_mode. Therefore, it knows that it can read the slave ID of the previous master from the predefined position in the RAM. It takes this value + 1 as its own slave ID, and updates the RAM position. Thereafter it enables its bus arbitration.
3. The third chip in the line will now win the bus, and will repeat the procedure of the second chip.
4. ... and so on until all chips have been initialized.

4.3.14.10 Boot methods

While in slave mode, a chip may boot up with any boot method that does not require access to the external bus. For a description of the available boot methods see 6.

Two of the boot methods, *Master chip boots slave* and *Slave chip boots master*, involve both a master and a slave chip. Both these boot methods use the boot_rdy field in the rw_slave_cfg register to handshake the boot progress between the master and the slave. The boot_rdy field is also available for read in the boot fields of the external slave mode registers r_ch0_stat - r_ch3_stat.

4.3.14.11 Loop back mode and slave mode disable

The slave mode interface is by default enabled only while the bus interface is in slave mode. The slave mode interface can be set to be constantly enabled through the loopback field of the rw_slave_cfg register. A master with the loopback field set will be able to access its own slave mode interface via the external bus. Accesses to the slave mode interface will still be qualified by the slave identification number if the slave identification number is enabled.

The slave mode interface can be completely disabled by setting the dis field in the rw_slave_cfg register. When disabled, accesses to the slave interface from the current bus master will not take effect. Disabling the slave mode does however not affect the participation in the bus arbitration, since this is controlled through rw_arb_cfg.

4.4 Hardware interface

4.4.1 Interface signals

The direction of the bus interface signals in master and slave modes is given in the tables below. Explanations:

I/O Bidirectional signal that changes direction during operation.

In Input signal.

Out Output signal.

Off The signal is not used and is not driven.

The mapping of the signals to package balls is described in 16.

4.4.1.1 Data bus

Description Data bus.

External pin	Master mode direction	Slave mode direction
d0-d31	I/O	I/O

4.4.1.2 Address bus

Description Address bus.

Address bit **a1** is also used as byte enable 3 in 32-bit common write enable mode, and used as data mask 7 in SDRAM cycles in 64-bit module mode.

Address bits **a2-a8** are also used as address inputs for register addressing and slave selection in slave mode.

Address bits **a19-a22** are also used as as data qualify mask outputs in SDRAM cycles.

Address bits **a23-a25** also serve as write enable, CAS and RAS signals for the SDRAM.

External pin	Master mode direction	Slave mode direction	Also used as:
a1	Out	Off	`be3_n` and `dqm7`
a2-a8	Out	In	-
a9-a18	Out	Off	-
a19-a22	Out	Off	`dqm0-dqm3`
a23	Out	Off	`sdwe_n`
a24	Out	Off	`cas_n`
a25	Out	Off	`ras_n`

4.4.1.3 Chip selects signals

Description Chip select outputs.

In slave mode, the **css_n** pin is used as the chip select input, and the **csp0_n-csp3_n** signals can be configured as data chip select inputs.

Pins **csp2_n**, **csp3_n**, **csp5_n** and **csp6_n** are multiplexed on the general I/O pins **pa0-pa3**, see 16. When used as chip selects, these signals need to have pull-up resistors to have a defined value at reset.

External pin	Master mode direction	Slave mode direction
cse0_n	Out	Off
cse1_n	Out	Off
csr0_n	Out	Off
csr1_n	Out	Off
csp0_n	Out	In
csp1_n	Out	In
csp2_n	Out	In
csp3_n	Out	In
csp4_n	Out	Off
csp5_n	Out	Off
csp6_n	Out	Off
css_n	Out	In

4.4.1.4 Read signal

Description Read signal.

External pin	Master mode direction	Slave mode direction
rd_n	Out	In

4.4.1.5 Write signals

Description Write signals.

The write signals **wr0_n-wr2_n** also serve as byte enables for bytes 0-2 in common write enable mode, and are used as data qualify masks 4-6 in SDRAM cycles in 64-bit wide module mode.

The write signal **wr3_n** is also used as common write enable output in common write enable mode, and as common write enable input in slave mode.

External pin	Master mode direction	Slave mode direction	Also used as:
wr0_n	Out	Off	`be0_n` and `dqm4`
wr1_n	Out	Off	`be1_n` and `dqm5`
wr2_n	Out	Off	`be2_n` and `dqm6`
wr3_n	Out	In	`cwe_n`

4.4.1.6 SDRAM signals

Description SDRAM chip select, clock and clock enable outputs.

In slave mode with shared DRAM control, these signals are used to detect if the current master accesses the SDRAM, to be able to decide if the stored row address is valid or not. In non-shared SDRAM mode, the signals are driven during slave mode.

External pin	Master mode direction	Slave mode direction	Description
csd0_n-csd1_n	Out	Out/In	SDRAM chip selects
sdclk	Out	Out/In	SDRAM clock
sdcke	Out	Out/In	SDRAM clock enable

4.4.1.7 Handshake signals

Description Handshake signals for the external DMA interface and the slave mode interface.

Pins **hsh4-hsh7** are multiplexed on the general I/O pins **pa4-pa7**, see 16.

External pin	Master mode direction	Slave mode direction
hsh0-hsh7	Out/In	Out/In

In master mode, each signal can be configured as:

Signal	Description	Direction
`dreq`	DMA request	In
`dack`	DMA acknowledge	Out
`tc_in`	Terminal count	In
`tc_out`	Terminal count	Out

In slave mode, each signal can be configured as:

Signal	Description	Direction
`dreq`	DMA request qualifier	In
`sreq`	Slave request	Out
`tc_in`	Terminal count	In
`tc_out`	Terminal count	Out

4.4.1.8 Wait signal

Description Wait input. Makes it possible for external units to generate extra wait states.

External pin	Master mode direction	Slave mode direction
wait_n	In	Off

4.4.1.9 Bus arbitration signals

Description Bus arbitration signals.

External pin	Master mode direction	Slave mode direction	Description
brin	In	In	Bus request input
brout	Out	Out	Bus request output
bg	Out	In	Bus grant signal

4.4.2 Reset behavior

In master mode, the dedicated chip select outputs and the **rd_n** and **wr0_n - wr3_n** pins are driven to the inactive (high) state immediately at system reset. The address bus is driven to an undefined value. The data bus and the **hsh0 - hsh3** pins are turned off immediately at reset.

During the start up sequence, the **sdclk** pin is cycled 5 times with **sdcke** high, to terminate any command that may have been initialized in the SDRAM before the reset. After this, the **sdclk** and **sdcke** pins are set low until the SDRAM interface is enabled.

In slave mode, all bus interface signals are turned off immediately at reset.

4.4.3 Detailed timing

4.4.3.1 SRAM/Flash/peripheral timing

4.4.3.1.1 Read cycle

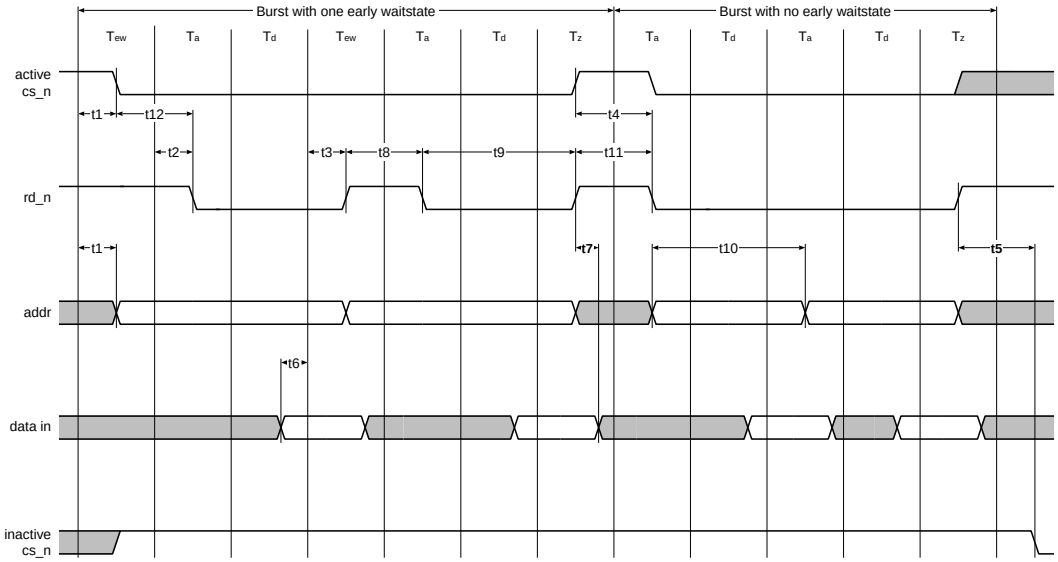


Figure 4.23: *SRAM/Flash/peripheral read timing*

Parameter	Description	Min	Nom	Max	Add with wait states	Unit
t1	Address and chip select delay from clock.	2	-	8	-	ns
t2	Read low delay from clock.	2	-	8	-	ns
t3	Read high delay from clock.	2	-	7	-	ns
t4	Chip select high to read low.	8	-	-	-	ns
t5	Read high to chip select low.	10	-	-	-	ns
t6	Data in setup time to clock.	0	-	-	-	ns
t7	Data in hold time from read.	0	-	-	-	ns
t8	Read inactive width within burst.	-2	-	-	10*ew	ns
t9	Read active width.	16	20	-	10*lw	ns
t10	Read cycle.	-	20	-	10*(lw+ew)	ns
t11	Read inactive after burst.	8	-	-	10*zw	ns
t12	Read inactive time after chip select or address.	-2	-	-	10*ew	ns

Table 4.23: *SRAM/Flash/peripheral read timing*

4.4.3.1.2 Write cycle

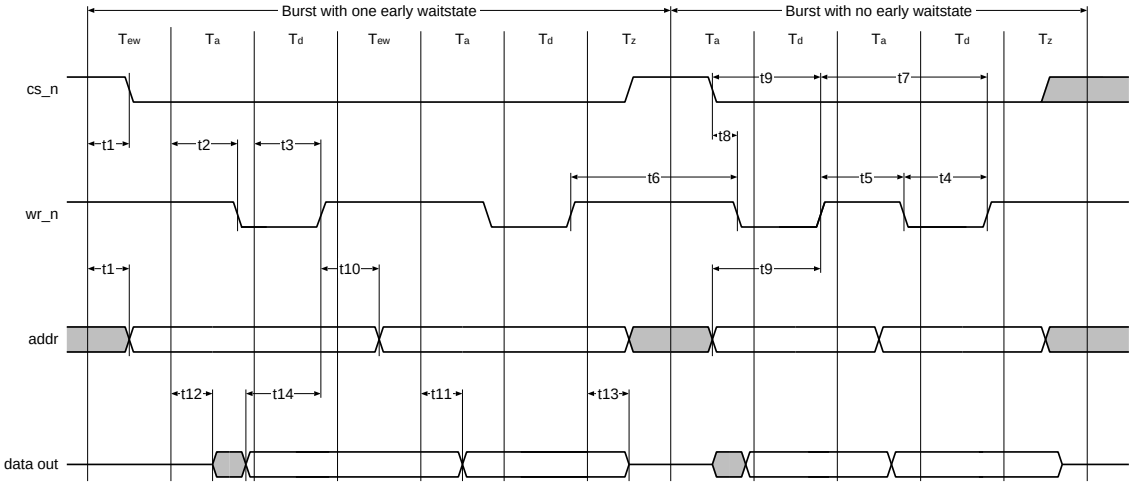


Figure 4.24: *SRAM/Flash/peripheral write timing*

Parameter	Description	Min	Nom	Max	Add with wait states	Unit
t1	Address and chip select delay from clock.	2	-	8	-	ns
t2	Write low delay from clock.	7	-	13	-	ns
t3	Write high delay from clock.	7	-	13	-	ns
t4	Write pulse width.	6	-	-	10*lw	ns
t5	Write inactive width within burst.	9	-	-	10*ew	ns
t6	Write inactive width after burst.	19	-	-	10*zw	ns
t7	Write cycle time.	-	20	-	10*(ew+lw)	ns
t8	Chip select and address setup to write low.	2	-	-	10*ew	ns
t9	Chip select and address setup to end of write.	11	-	-	10*(ew+lw)	ns
t10	Address hold after write high.	3	-	-	-	ns
t11	Data delay from clock.	6	-	13	-	ns
t12	Data turn on time from clock.	6	-	-	-	ns
t13	Data turn off time from clock.	6	-	10	10*zw	ns
t14	Data valid to end of write.	6	-	-	10*lw	ns

Table 4.24: *SRAM/Flash/peripheral write timing*

4.4.3.1.3 Extended write cycle

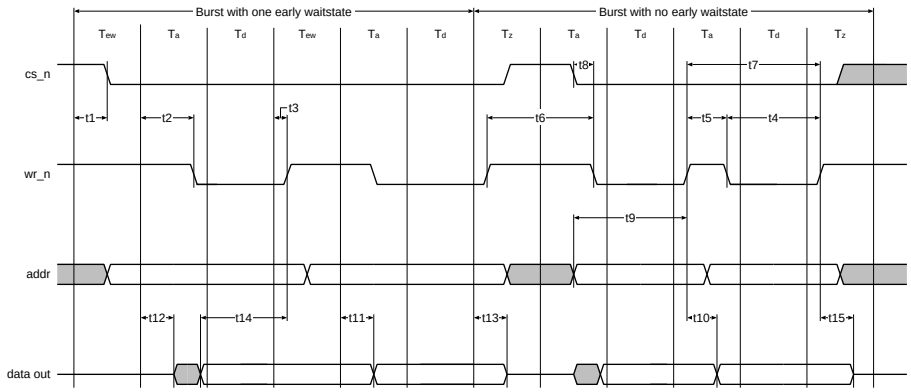


Figure 4.25: *SRAM/Flash/peripheral extended write timing*

Parameter	Description	Min	Nom	Max	Add with wait states	Unit
t1	Address and chip select delay from clock.	2	-	8	-	ns
t2	Write low delay from clock.	7	-	13	-	ns
t3	Write high delay from clock.	2	-	7	-	ns
t4	Write pulse width.	11	-	-	10*lw	ns
t5	Write inactive width within burst.	4	-	-	10*ew	ns
t6	Write inactive width after burst.	13	-	-	10*zw	ns
t7	Write cycle time.	-	20	-	10*(ew+lw)	ns
t8	Chip select and address setup to write low.	2	-	-	10*ew	ns
t9	Chip select and address setup to end of write.	16	-	-	10*(ew+lw)	ns
t10	Data hold after end of write, within burst.	2	-	-	10*ew	ns
t11	Data delay from clock.	6	-	13	-	ns
t12	Data turn on time from clock.	6	-	-	-	ns
t13	Data turn off time from clock.	6	-	10	10*zw	ns
t14	Data valid to end of write.	11	-	-	10*lw	ns
t15	Data hold after end of write, at end of burst.	2	-	-	10*zw	ns

Table 4.25: *SRAM/Flash/peripheral extended write timing*

4.4.3.1.4 External wait input timing

The timing diagram shows an example with internal wait state configuration set to one early wait state and two late wait states.

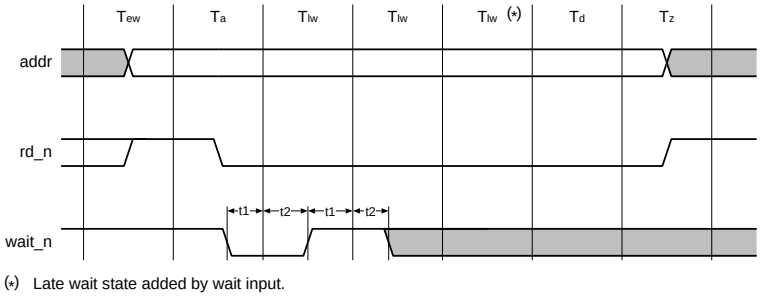


Figure 4.26: *External wait input timing*

Parameter	Description	Min	Max	Unit
t1	Wait input setup to clock.	3	-	ns
t2	Wait input hold from clock.	0	-	ns

Table 4.26: *External wait input timing*

4.4.3.2 SDRAM timing

4.4.3.2.1 SDRAM read timing

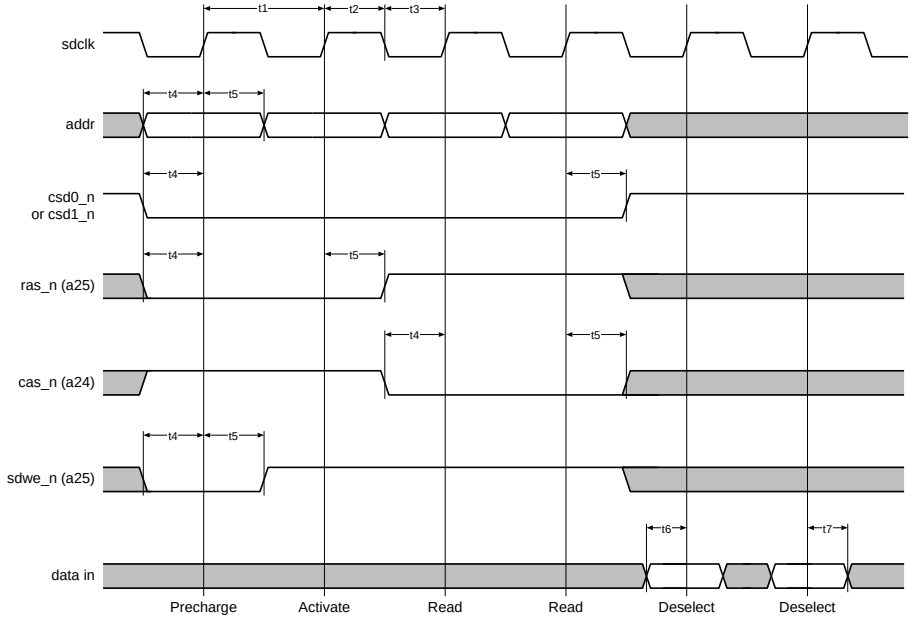


Figure 4.27: *SDRAM read timing*

Parameter	Description	Min	Nom	Max	Unit
t1	Clock period.	-	10	-	ns
t2	Clock high time.	3	-	-	ns
t3	Clock low time.	3	-	-	ns
t4	Setup time to clock.	2	-	-	ns
t5	Hold time from clock.	1	-	-	ns
t6	Data in setup time to clock.	0	-	-	ns
t7	Data in hold time from clock.	3	-	-	ns

Table 4.27: *SDRAM read timing*

4.4.3.2.2 SDRAM write timing

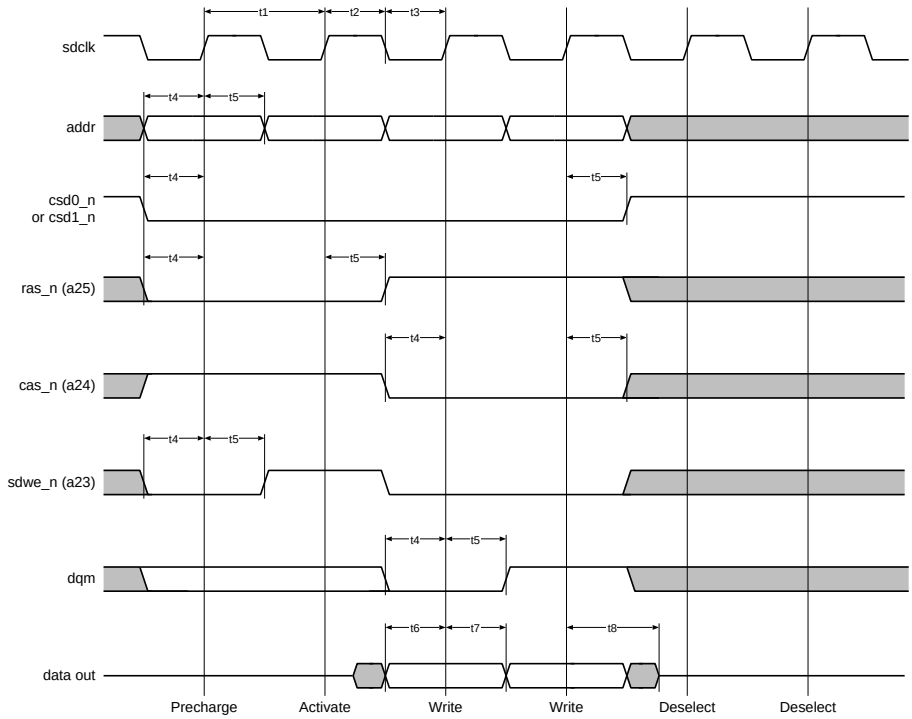


Figure 4.28: *SDRAM write timing*

Parameter	Description	Min	Nom	Max	Unit
t1	Clock period.	-	10	-	ns
t2	Clock high time.	3	-	-	ns
t3	Clock low time.	3	-	-	ns
t4	Setup time to clock.	2	-	-	ns
t5	Hold time from clock.	1	-	-	ns
t7	Data in hold time from clock.	3	-	-	ns
t6	Data out setup time to clock.	2	-	-	ns
t7	Data out hold time from clock.	1	-	-	ns
t8	Data out turn off time from clock	-	-	8	ns

Table 4.28: *SDRAM write timing*

4.4.3.3 External DMA timing

4.4.3.3.1 External DMA read timing

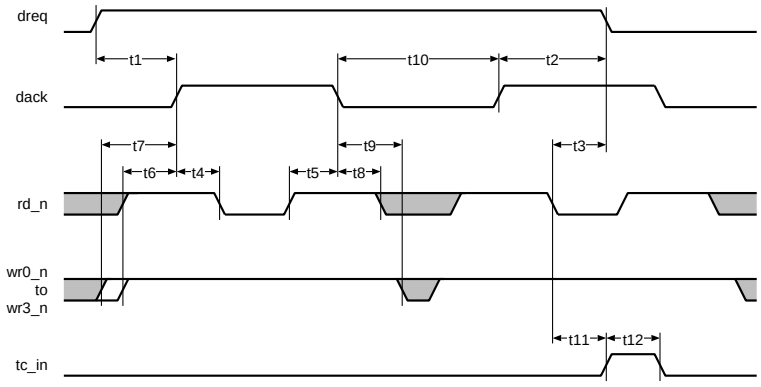


Figure 4.29: *External DMA read timing*

The timing diagram 4.29 is shown with dreq, dack, tc_in and tc_out configured to be active high.

Parameter	Description	Min	Max	Add with wait states	Unit
t1	dreq to dack delay.	27	-	-	ns
t2	dreq hold time from dack.	0	-	-	ns
t3	dreq inactive time from rd_n low to inhibit next transfer (applies to the last rd_n of a bus burst).	-	17	10*lw	ns
t4	dack active to rd_n low delay.	2	-	10*ew+10* ewb+10*dw	ns
t5	rd_n high to dack inactive delay.	2	-	10*erc	ns
t6	wr_n high (extended write mode) or rd_n high to dack active delay.	2	-	-	ns
t7	wr_n high (normal write mode) to dack active delay.	7	-	-	ns
t8	dack inactive to rd_n low delay.	2	-	-	ns
t9	dack inactive to wr_n low delay.	7	-	-	ns
t10	dack inactive to dack active delay.	27	-	-	ns
t11	tc_in active time from rd_n low to inhibit next transfer (applies to the last rd_n of a bus burst).	-	17	10*lw	ns
t12	tc_in active pulse width.	14	-	-	ns

Table 4.29: *External DMA read timing*

4.4.3.3.2 External DMA write timing

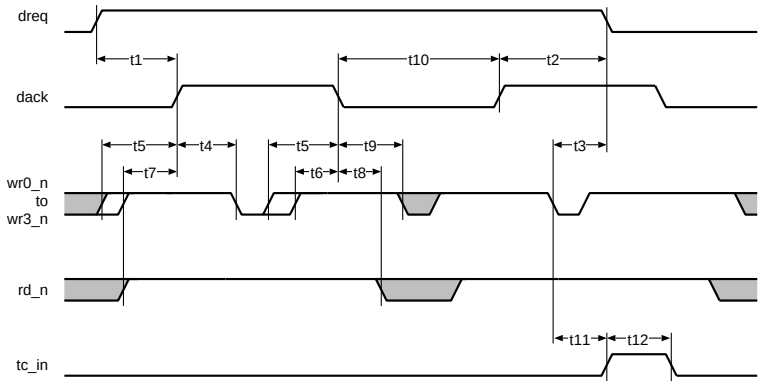


Figure 4.30: *External DMA write timing*

The timing diagram 4.30 is shown with dreq, dack, tc_in and tc_out configured to be active high.

Parameter	Description	Min	Max	Add with wait states	Unit
t1	dreq to dack delay.	27	-	-	ns
t2	dreq hold time from dack.	0	-	-	ns
t3	dreq inactive time from wr_n low to inhibit next transfer (applies to the last wr_n of a bus burst).	-	12	10*lw	ns
t4	dack active to wr_n low delay.	7	-	10*ew+10* ewb+10*dw	ns
t5	wr_n high (normal write mode) to dack active or inactive delay.	7	-	-	ns
t6	wr_n high (extended write mode) to dack inactive delay.	2	-	-	ns
t7	wr_n high (extended write mode) or rd_n high to dack active delay.	2	-	-	ns
t8	dack inactive to rd_n low delay.	2	-	-	ns
t9	dack inactive to wr_n low delay.	7	-	-	ns
t10	dack inactive to dack active delay.	27	-	-	ns
t11	tc_in active time from wr_n low to inhibit next transfer (applies to the last wr_n of a bus burst).	-	12	10*lw	ns
t12	tc_in active pulse width.	14	-	-	ns

Table 4.30: *External DMA write timing*

4.4.3.3.3 External DMA tc_out timing

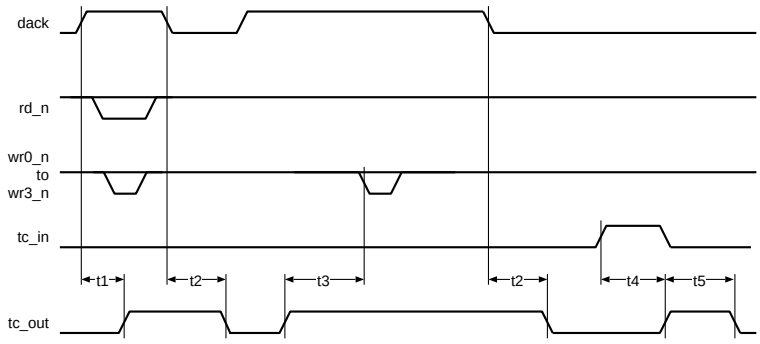


Figure 4.31: *External DMA tc_out timing*

Parameter	Description	Min	Max	Add with wait states	Unit
t1	dack active to tc_out active delay, when tc_out is triggered by transfer counter.	2	8	-	ns
t2	dack inactive to tc_out inactive delay, when tc_out is generated within an external DMA bus burst. [6]	2	8	-	ns
t3	tc_out active to wr_n low delay, when tc_out is triggered by out_eop.	2	-	10*ew+ 10*dw	ns
t4	tc_in active to tc_out active delay.	22	45	-	ns
t5	tc_out pulse width when generated outside an external DMA bus burst.	17	23	-	ns

Table 4.31: *External DMA tc_out timing*

[6] If tc_out is generated later than 15 ns before dack inactive, t5 applies instead.

4.4.3.4 Slave mode timing

4.4.3.4.1 Slave mode read timing

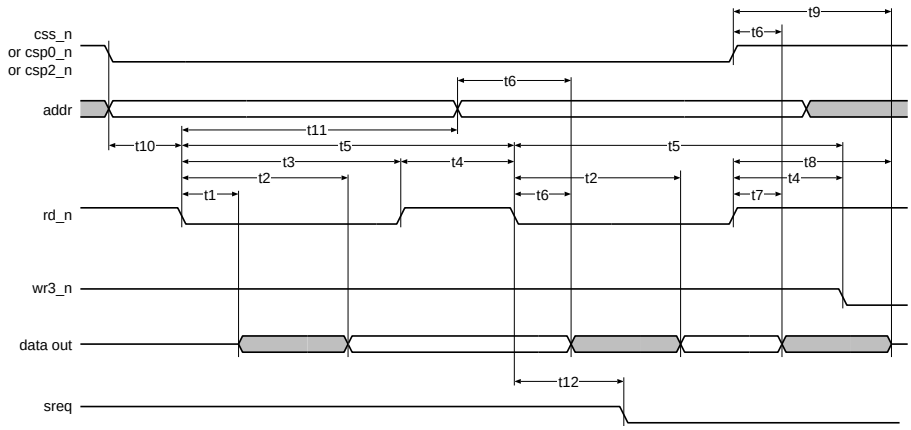


Figure 4.32: *Slave mode read timing*

The timing diagram 4.32 is shown with sreq configured as active high and the data chip select inputs csp0_n and csp2_n configured as active low.

Parameter	Description	Min	Max	Unit
t1	Output enable time from rd_n.	2	-	ns
t2	Access time from rd_n.	-	12	ns
t3	rd_n active pulse width.	6	-	ns
t4	rd_n inactive until next rd_n or wr3_n.	6	-	ns
t5	Read cycle time.	20	-	ns
t6	Data hold after rd_n low, chip select inactive or address change, whichever occurs first, if data not turned off due to rd_n high.	2	-	ns
t7	Data hold after rd_n high if no new rd_n low, address change or chip select high occurs. [7]	2+10*rh	-	ns
t8	Data turn-off time after rd_n high. [7]	-	11+27*rh	ns
t9	Data turn-off time after chip select high.	-	12	ns
t10	Address and chip select setup time to rd_n low.	5	-	ns
t11	Address hold time from rd_n low.	5	-	ns
t12	sreq inactive delay after rd_n low.	2	12	ns

Table 4.32: *Slave mode read timing*

[7] rh can be set to 0 or 1. Default is rh=1.

4.4.3.4.2 Slave mode write timing

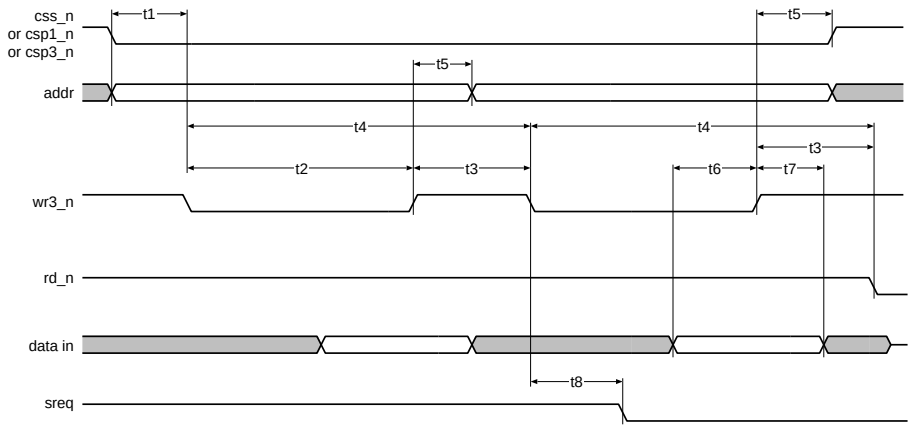


Figure 4.33: *Slave mode write timing*

The timing diagram 4.33 is shown with sreq configured as active high and the data chip select inputs csp1_n and csp3_n configured as active low.

Parameter	Description	Min	Max	Unit
t1	Address and chip select setup time to write low.	5	-	ns
t2	wr3_n active pulse width.	6	-	ns
t3	wr3_n inactive until next wr3_n or rd_n.	6	-	ns
t4	Write cycle time.	20	-	ns
t5	Address and chip select hold time after wr3_n high.	0	-	ns
t6	Data setup time to wr3_n high.	2	-	ns
t7	Data hold time from wr3_n high.	0	-	ns
t8	sreq inactive delay after wr3_n low.	2	12	ns

Table 4.33: *Slave mode write timing*

4.5 Software interface

4.5.1 Bus interface general registers

The bus interface general mode registers are specified in 25.4. To access the registers, fields and register constants from a C program, a set of macros is defined in [BIF_MACROS].

4.5.2 External DMA

The external DMA mode registers are specified in 25.5. To access the registers, fields and register constants from a C program, a set of macros is defined in [EXTDMA_MACROS].

Before an external DMA channel can be used, it has to be connected to an internal DMA channel. The connections between external and internal DMA channels are described in 10.

The external DMA uses simple data descriptor lists for its operation. The `wait` field of the DMA descriptors should not be used with external DMA. For further details on how to set up the internal DMA, see 5.

4.5.3 Slave mode and master/slave arbitration

4.5.3.1 Internal mode registers

The internal mode registers for the slave mode and master/slave arbitration are specified in 25.6. To access the registers, fields and register constants from a C program, a set of macros is defined in [SLAVE_MACROS].

4.5.3.2 External slave mode registers

The external slave mode registers are specified in 25.7. To access the registers, fields and register constants from a C program, a set of macros is defined in [EXT_MACROS].

These registers are available to an external host directly via the slave mode interface. They are not accessible from within the slave, but some of the functions are affected also by internal mode registers.

4.5.4 Programming considerations

4.5.4.1 Race avoidance between mode registers and external bus cycles

External bus write cycles, as well as writes to internal registers, are buffered to allow the CPU to continue execution without waiting for write completion.

Completion of external bus cycles and completion of accesses to internal mode registers are independent of each other, and the order of completion may be different from the order of the instructions in the program code. However, different accesses within the address space of the external bus are always completed in order, and the same holds valid for different accesses within the internal mode register address space.

In cases where internal mode register accesses and external bus cycles can interact with each other, it is necessary to take care of this possible race condition. To make sure that a write to an internal mode register is completed, it can be followed by a read from the same or any other internal mode register. In the same way, to make sure that an external bus write cycle is completed, it can be followed by a read from anywhere within the address space of the external bus.

Chapter 5

DMA

5.1 References

Tables 5.1 and 5.2 lists references to other documents and terms defined elsewhere in documentation.

[DMA_REG_DOC]	DMA Register Documentation, chapter 25.11
[IOP_DMC_IN]	I/O Processor DMC In Register Documentation, chapter 25.16
[IOP_DMC_OUT]	I/O Processor DMC Out Register Documentation, chapter 25.17

Table 5.1: *References*

References below are for SW development only.

[DMA_REG_MACROS]	DMA Register Macros, http://developer.axis.com
[DMA_MACROS]	DMA Operation and struct Macros, http://developer.axis.com

Table 5.2: *Software development references*

5.2 Definitions

The following special definitions are used in this chapter:

DMA	The unit that performs Direct Memory Access.
Client	A hardware unit using a DMA to stream data to or from memory.
Context	Complete description of the current state of the DMA including internal state and list position.
In channel DMA	A DMA channel that transports data from a client to memory.
Out channel DMA	A DMA channel that transports data from memory to a client.
dma_x	DMA channel number x.
Virtual channel	Time multiplexed data streams on the same physical DMA channel.

eol	End of descriptor list.
in_eop	In channel descriptor end of packet.
out_eop	Out channel descriptor end of packet.
tol	Top of list. Current descriptor has no descriptor at the level above. Pointer to upper level is not valid.
bol	Bottom of list. Indicates there may be a context list attached.
wait_ack	Wait acknowledge sequence. Out channel specific behavior issued by the wait bit of the data descriptor ctrl field.
Stream command	Command controlling the DMA operation.
SW	The software that runs on the system.
Byte	8 bits of data in a register or memory.
Word	16 bits of data in a register or memory.
Dword	32 bits of data in a register or memory.

Table 5.3: *Definitions*

5.3 Overview

The DMA is used to transport data efficiently between memory and DMA clients. Data is read from and written to buffers in memory, linked together with data structures called descriptors.

The data is transported between the DMA and memory in chunks of 32 bytes whenever possible, to take advantage of the 32 byte wide on-chip bus. Theoretically, each physical DMA channel is able to handle an unlimited number of virtual channels, by means of the advanced list structure shown by the figure 5.3 in section 5.4.3.

A DMA list is a list of descriptors keeping track of the list processing at three levels: data, context and group. Each descriptor contains control data to keep track of the list boundaries, interrupt positions, and level specific information that is read by the DMA. The DMA writes necessary status information at each descriptor level.

All descriptors have a meta data field for communication between SW and the client. When a group or context descriptor is loaded into the DMA registers, the meta data of each type is sent to the client. The group or context meta data from the client to the descriptor is sampled when the DMA receives a stream command that requires storing the group or context descriptor.

A data descriptor keeps track of one data buffer in memory, and points to the next data descriptor in a list of data descriptors. A context descriptor keeps track of the data descriptor under process and its present data buffer position. This allows the DMA to save and restore its list state in the context descriptor when commanded by its client or by SW.

The context saved always represents what the DMA client sees. If the DMA internally buffers data or has other speculative behavior for e.g. performance reasons, the DMA will automatically compensate for this to guarantee that it is the context seen by the DMA client that is saved and later restored. To support virtual channels, context descriptors may be linked together.

When needed, group descriptors are used to handle multiple context descriptor lists,

and to get a certain data transfer rate from virtual channels.

The traversing of DMA lists is primarily controlled by stream commands from the client. This makes it possible to support both simple clients (e.g simple data streaming), and more complex clients that need more control of the DMA operation (e.g. use virtual channels).

The specific client may implement a subset of the complete functionality and list levels. The client documentation will describe how the list is to be processed and in which order descriptors are traversed.

SW initiates list pointers, starts or stops the DMA, and may read status information. It is also possible for SW to give stream commands to assist a simple client requiring advanced list traversing and SW workarounds.

An interrupt is generated when the DMA follows a link of a descriptor with the intr bit set. If the intr bit of the last data descriptor of a list is set, a data descriptor interrupt is generated when the data descriptor has been fully processed. Note that there may be buffered data in the DMA at the time the interrupt is generated.

An in channel end of packet interrupt is generated after the DMA has written all the data to the associated data buffer and the data descriptor has been updated with in_eop. A stream command interrupt is generated when a stream command given by the client or the SW has been executed.

5.4 Functional Description

DMA operation is controlled by the DMA lists representing the data, stream commands given by the stream client to control how the lists are processed, and register settings where software do general configuration.

Before list processing and data transfers can start software must provide an initial pointer to a root descriptor. The root descriptor may be a group or context descriptor depending of the chosen list structure. A stream command is needed to load this root descriptor and start processing the list structure and transfer data.

The following DMA registers are used to point to the root descriptor of the list structure.

rw_group When starting at group level

rw_group_down When starting at context level

5.4.1 Data Level

The data descriptor is described in section 5.5.6.1.

The simplest DMA list has a context descriptor as header of linked data descriptors. When the context descriptor has been loaded into the DMA, the next step is to command the DMA to load the data descriptor pointed to by the context descriptor. When the first data descriptor is loaded, the DMA automatically starts processing the buffer also pointed to by the context descriptor.

The following DMA registers are used for holding pointers:

rw_saved_data Initially this register holds the pointer to the first data descriptor. The register is updated during the processing of the data descriptor list. When the DMA stores the context descriptor, the register content is stored.

rw_saved_data_buf Initially this register holds the pointer to the first byte of the data buffer. The register is updated during the processing of the data buffer. When the DMA stores the context descriptor, the register content is stored.

These pointers mark the position where the transfer is to be started or restarted. The context descriptor is updated when the DMA receives a stream command that requests storing the current DMA context. At restore, the DMA loads the latest saved values.

A packet can be split over several data buffers, linked together by the data descriptors. The last data descriptor of the list is marked with eol. A list of data descriptors may contain several packets. The first packet starts at the beginning of the list and ends at a data descriptor marked by eop. The following packet begins at the next data descriptor. See the example in figure 5.1.

When the DMA reaches the end of the data descriptor list the context is disabled by the DMA, and data_at_eol is set in the list_state field of the DMA status register rw_stat. dis is set internally in rw_ctxt_stat, and when the DMA client gives a stream command to save the context the corresponding bit in the context descriptor is set.

The client may give a stream command that updates the meta data field of the data descriptor, (i.e. that stores meta data).

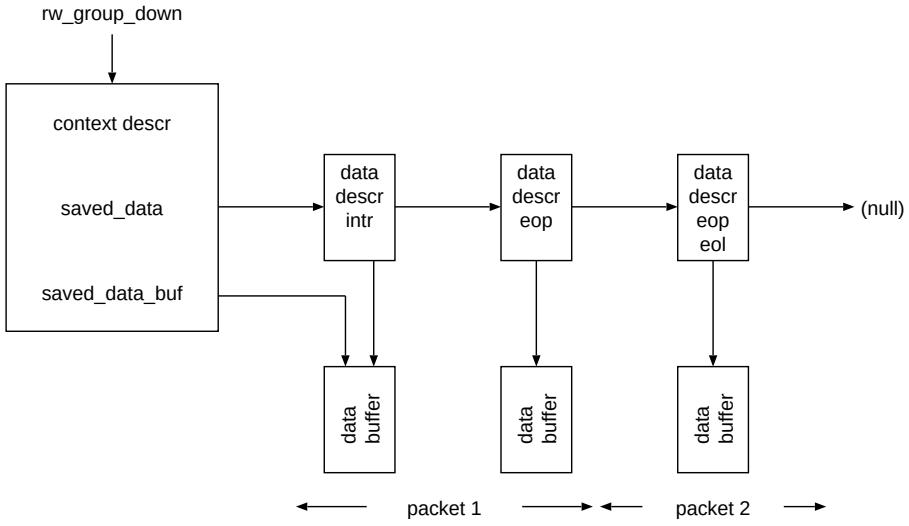


Figure 5.1: *Data descriptor list*

See the section 5.5.6.4.1, for an in and out channel descriptor setup example of this list.

5.4.1.1 Out Channel

Normally the out channel DMA continues from one buffer to the next without waiting for all data in the first to be consumed by the client. If it is desired that the out channel DMA should wait for all data in a buffer to be consumed by the client before continuing, the wait bit in the data descriptor ctrl field should be set. When the DMA has read all data in the first buffer and the wait bit is set, it stops and waits for the client to acknowledge that it has read all data. The client acknowledges by giving a stream command. This is called the wait_ack sequence.

There is a stream command to search for the start of the next packet, skipping the rest of the current packet. When the next packet is found data streaming may be initiated automatically.

The DMA only reads and propagates meta data from the first data descriptor of a packet and after a wait_ack.

5.4.1.2 In Channel

When the last data of an incoming packet has been stored in the data buffer, the data descriptor is updated with the following:

- The in_eop status bit is set.
- The after field is updated with the pointer to the end of the buffer. The pointer refers to the byte after the last byte written to the data buffer.
- The meta data field is updated with the meta data that was received from the client with the last received word.

  When proceeding to the next data descriptor an end of packet interrupt is generated.

5.4.2 Context Level

The context descriptor is described in section 5.5.6.2.

Lists of context descriptors are always used together with data lists, and are used to keep track of virtual channel contexts. To prevent the data of the data buffer to be unintentionally overwritten, only one context descriptor should point to each data descriptor list of an in channel.

The context descriptor holds 10 bytes of meta data.

The context descriptor list can be a singly linked circular list of context descriptors with the last descriptor pointer set to the first context descriptor of the list. The context descriptors may also be set up as an array to get random access.

The last descriptor in the list or array is marked by an end of list marker (eol). Enabled context descriptors are marked with en.

For a descriptor setup example of this list see section 5.5.6.4.4.

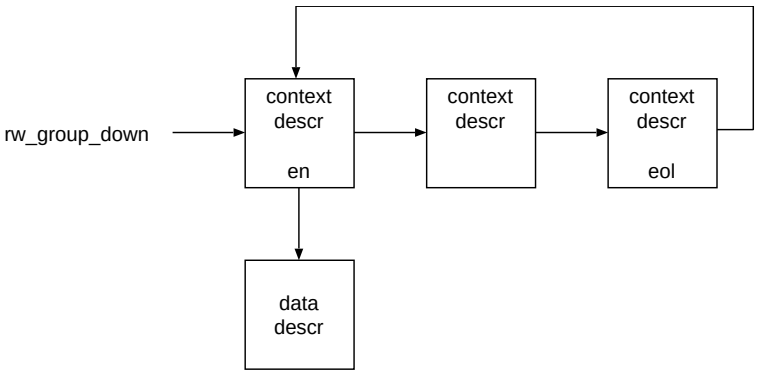


Figure 5.2: *Context descriptor list*

The circular list of context descriptors is processed by the client giving a stream command that makes the DMA load the next context descriptor. An alternative for the client is to give a stream command that loads the next enabled context descriptor. If no enabled context descriptor is found, the DMA disables the group descriptor if the group descriptor level exists, and the DMA stops at the context descriptor marked with eol when passing the second time. The busy field in the rw_stream_cmd register, discussed in section 5.5.5.1 is always released when the stream command has finished. To be able to determine if an enabled context descriptor or not was found the client has to observe the context meta data status and control fields.

To gain random access, a stream command is used that makes the DMA load a context descriptor with a multiplied offset of the context descriptor memory size related to the rw_group_down address. This assumes the context descriptors to be in consecutive memory locations and of the same size.

At random access, the DMA may also be commanded to find the next enabled context descriptor by parsing each context descriptor of the list starting by the n:th context descriptor until an enabled context descriptor is found.

When changing context (i.e. changing virtual channel), the context descriptor has to be stored to be able to restore the state when returning to the particular virtual channel.

The DMA passes the last context descriptor of a list, if commanded to load the next context descriptor when at the last context descriptor.

5.4.3 Group Level

The group descriptor is described in the section 5.5.6.3.

Group descriptors are organized in a two dimensional group list structure, and are used to keep track of other group descriptors and context descriptors.

A group descriptor may be an element in two lists at the same time:

1. Group descriptors may be organized at the same level in a singly linked list, linked together by the next pointers.

2. Each group descriptor may be in a doubly linked list with upper and lower levels pointed to by the up and down pointers.

The two dimensional list structure makes it possible to organize and keep track of context descriptors in a flexible way. A typical example when this flexibility is useful is when the DMA is working with a USB interface. See the somewhat simplified example below.

5.4.3.1 A USB Example

A top level horizontal group list may be used with one group descriptor for each USB traffic type (intr, iso, ctrl, bulk).

The group descriptor for bulk and ctrl traffic then points to a context descriptor in a circular list of context descriptors for the respective traffic types. The function of these group descriptors is to remember which context descriptor to be used when to send data of that traffic type.

For iso and intr traffic the top level group descriptors point to an intermediate level of group descriptors with their down pointer. These intermediate group descriptors then point to the actual context descriptors. This intermediate level may be used for transmission rate control. For each USB frame the next group descriptor are used to locate which context to use as the first context in that frame.

With this arrangement it is possible to get different transmit intensity for different contexts, like different USB end points. For example, in the 0, 1, 2, 2, 0, 1, 2, 2, ... sequence shown in figure 5.3, context (2) can get twice as many transmission opportunities ascontext (0) and (1).

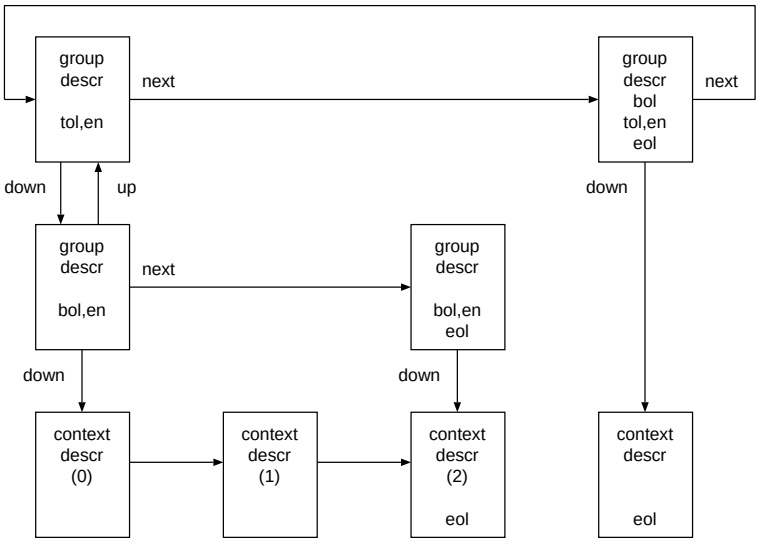


Figure 5.3: *USB Example*

This is also an example of how to use the flexibility of the list structure. The specific

client documentation will describe if and how to use this two dimensional group list structure.

Group lists are always used together with context lists and data lists.

The DMA may be commanded to move between levels of the group descriptor hierarchy and sideways at the current level. The up and down pointers may be modified by the DMA as commanded. The DMA does not pass group descriptors marked with tol along the up pointer, and does not pass group descriptors marked with bol along the down pointer, but may pass descriptors marked with eol along the next pointer if commanded to load the next group descriptor. In this case the next pointer of the group descriptor with eol set must be valid.

The circular list of group descriptors is processed by the client giving a stream command that makes the DMA load the next group descriptor. An alternative for the client is to give a stream command that loads the next enabled group descriptor. If no enabled group descriptor is found, the DMA stops at the group descriptor marked with eol when passing the second time. The busy field in the rw_stream_cmd register, discussed in the section 5.5.5.1 is always released when the stream command has finished. To be able to determine if an enabled group descriptor or not was found the client has to observe the group meta data status and control fields.

Note that the DMA does not disable group descriptors when searching for enabled group descriptors, the DMA only disables group descriptors when searching for enabled context descriptors.

5.5 Software Interface

The software interface use a set of C structs to represent the DMA list descriptors, and a set of macros for basic stream commands, and register configuration. These structs and macros are defined in [DMA_MACROS].

To access registers, fields, and register constants from a C program, a set of macros is defined in [DMA_REG_MACROS].

The DMA registers are described in detail in 25.11.

Figure 5.4 provides a brief summary of the DMA pointer registers. For a more detailed description see 25.11.

5.5.1 Pointer Registers and Descriptors

Each DMA channel has a set of list managing registers for pointers, status, ctrl and meta data. Other registers such as general, interrupt, and the stream command registers are used to control the DMA operation. The only registers in figure 5.4 that should be modified by SW are either rw_group or rw_group_down.

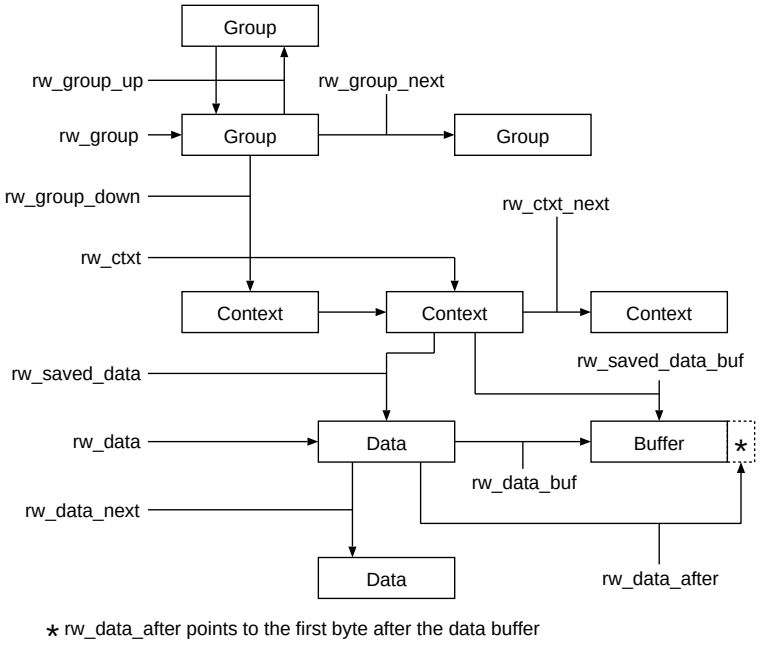


Figure 5.4: *Pointer registers*

Note that the pointer registers represent how far the DMA has processed the list, and that this may differ from what the DMA client sees due to internal buffering of data or other speculative behavior. However, when the DMA saves the current context it automatically compensates for this and always saves the context seen by the DMA client.

5.5.2 DMA List Pointers

5.5.2.1 DMA List Pointer Registers

The following pointer registers can be modified by software and are used for DMA list pointer configuration:

Register	Description
rw_group	Group descriptor register. This register holds the pointer to the current group descriptor.
rw_group_down	Group descriptor down register. This register holds the down pointer of the current group descriptor.

Table 5.4: *DMA List Pointer Registers*

Note that the other DMA list pointer registers should not normally be used. If software wants to examine the exact position of the current context, as seen by the DMA client, SW must save the current context by a stream command and examine the contents of the context descriptor in memory.

5.5.2.2 DMA List Pointer Configuration

The following shows how to configure rw_group and rw_group_down by using the macro definitions in [DMA_REG_MACROS].

Set Group Start Pointer:

```
// Write the initial group pointer to the rw_group register.
//
REG_WR( dma, instance, rw_group, (unsigned int) *group );
```

Set Context Start Pointer:

```
// Write the initial context pointer to the rw_group_down register.
//
REG_WR( dma, instance, rw_group_down, (unsigned int) *context );
```

5.5.3 General DMA Operation

5.5.3.1 General DMA Operation Registers

5.5.3.1.1 rw_cmd

Register rw_cmd is used by software after appending new data descriptors to a data list. See 25.11 for register details, and [DMA_MACROS] for macro definition.

```
// Set the continue_data-bit of rw_cmd.
//
DMA_CONTINUE_DATA( instance );
```

5.5.3.1.2 rw_cfg

Register rw_cfg is used to start and stop DMA operation. See 25.11 for register details, and [DMA_MACROS] for macro definition.

```
// Set the en-bit of rw_cfg.
//
DMA_ENABLE( instance );

// Deassert the en-bit of rw_cfg.
//
DMA_RESET( instance );

// Set the stop-bit of rw_cfg.
//
```

```
DMA_STOP( instance );

// Deassert the stop-bit of rw_cfg.
//
DMA_CONTINUE( instance );
```

5.5.3.1.3 rw_stat

Register rw_stat is updated by the DMA hardware with information about the internal operation. See 25.11 for register details.

5.5.3.1.4 Setup and Start a Data Level DMA List

The following sequence shows an example of setup and start of a data level DMA list. See [DMA_MACROS] and [DMA_REG_MACROS].

- **Example:**

```
// Build the list.
//
init_list();

// Reset DMA channel.
//
DMA_RESET( dma_x );

// Enable DMA channel.
//
DMA_ENABLE( dma_x );

// Load the first pointer of the list, into the DMA register.
//
REG_WR( dma, dma_x, rw_group_down, (unsigned int) *context );

// The transfer word size is 1 byte by default. In this example the
// transfer word size is changed to 4 bytes.
//
DMA_WR_CMD( dma_x, regk_dma_set_w_size4 );

// Load the context descriptor.
//
DMA_WR_CMD( dma_x, regk_dma_load_c );

// Load the start position and start the data burst.
//
DMA_WR_CMD( dma_x, regk_dma_load_d | regk_dma_burst );
```

5.5.4 Interrupt

5.5.4.1 Interrupt Registers

The following registers are used to handle the interrupts from group, context and data descriptors, and in-channel eop and stream commands. When an interrupt is generated, the DMA operation continues without waiting for the acknowledge.

Register	Description
r_intr	
r_masked_intr	
rw_intr_mask	Enable/disable interrupt generation.
rw_ack_intr	Clear interrupts; stream_cmd, eop, data, context, and group

Table 5.5: *DMA Interrupt Registers*

5.5.4.2 Interrupt Signals

Table 5.6 lists the DMA interrupt signals.

Interrupt	Description
group	A group descriptor interrupt is generated when a group descriptor pointer is followed and the intr bit of the current group descriptor is set.
ctxt	A context descriptor interrupt is generated when a context descriptor pointer is followed and the intr bit of the current context descriptor is set.
data	A data descriptor interrupt is generated when a data descriptor pointer is followed and the intr bit of the current data descriptor is set. Also generated when the last data descriptor has been fully processed and the intr bit of the last data descriptor is set.
in_eop	An in_eop interrupt is generated when the status field of the current data descriptor field is updated.
stream_cmd	A stream command interrupt is generated when a stream command has been executed.

Table 5.6: *DMA Interrupt Signals*

All DMA interrupts listed in table 5.6 are generated after the descriptor, data buffer, and status registers are updated, i.e. it is safe for the interrupt handler to examine the updated DMA list and registers.

5.5.5 Stream Commands Controlling the DMA List Operation

As described in section 5.4, the traversing of DMA lists is primarily controlled by stream commands. The client documentation will describe how to set up and command the DMA for that specific client.

Some stream clients allow software to give stream commands and examine stream

command status, other clients do not have this feature. The rest of section 5.5.5 and subsections assume the connected stream client has this feature, for other clients some or all functionality described may be missing. Please refer to the appropriate stream client documentation.

5.5.5.1 rw_stream_cmd

The rw_stream_cmd register contains the last stream command given by SW. This register is not updated with commands given by the DMA client. The client documentation will describe if and when SW may write to this register to load and store descriptors etc.

For all details see 25.11.

5.5.5.2 Summary of Stream Commands

Table 5.7 gives a summary of the stream commands and the options which can be used with each command. Each command and option are explained in greater detail in the following sections below. Stream commands are written to the cmd field in the rw_stream_cmd register.

Type of Command	Command	Options
General	store_descr	dis_g, dis_c, store_g, store_c, store_md
General	set_reg	store_g, store_c, store_md
Group Level	load_g	
Group Level	load_g_next	copy_up, next_en, array, store_g, store_c, store_md
Group Level	load_g_up	save_down, copy_next, store_g, store_c, store_md
Group Level	load_g_down	save_up, store_g, store_c, store_md
Context Level	load_c	store_g, store_c, store_md
Context Level	load_c_next	update_down, next_en, array, store_g, store_c, store_md
Context Level	load_c_n	update_down, next_en, array, store_g, store_c, store_md
Data Level	ack_pkt	restore, store_g, store_c, store_md
Data Level	load_d	burst, next_pkt, store_g, store_c, store_md
Data Level	set_w_size1	store_g, store_c, store_md
Data Level	set_w_size2	store_g, store_c, store_md
Data Level	set_w_size4	store_g, store_c, store_md

Table 5.7: *Summary of Stream Commands*

5.5.5.3 Stream Command Option Descriptions

Table 5.8 gives a detailed explanation of the options which can be used with stream commands.

Option	Description

dis_g	Disable the current group descriptor. Assumes a group descriptor already loaded into the DMA registers. Normally used with the store_g option. This changes the list structure when the current group descriptor is stored to memory.
dis_c	Disable the current context descriptor. Assumes a context descriptor already loaded into the DMA registers. Normally used with the store_c option. This changes the list structure when the current context descriptor is stored to memory.
copy_up	Copy up. Keeps the current pointer to upper descriptor, rw_group_up, when loading the next descriptor. Observe that this changes the list structure if the current group descriptor is stored to memory. Use with load_g_next.
next_en	Next enabled. If the next group or context descriptor is not enabled its next group or context descriptor is loaded, and so on, until an enabled descriptor is found, or the end of list is visited for the second time. This assumes the last group descriptor of the list has a valid pointer to its next group descriptor. Use with load_g_next, load_c_next, and load_c_n.
array	Array. Override the next_en circular list behavior and stop at first end of list if no enabled group or context descriptor is found. Only considered when next_en is true. Use with load_c_next, and load_c_n.
save_down	Save down. Copies the current group descriptor address, rw_group, to the lower group descriptor pointer register, rw_group_down, and then keeps it when loading the upper group descriptor. Observe that this changes the list structure if the current group descriptor is stored to memory. Use with load_g_up.
copy_next	Copy the next pointer of the current group descriptor. Overrides save_down. Copies the current pointer to the next group descriptor address, rw_group_next, to the lower group descriptor pointer register, rw_group_down, and then keeps it when loading the upper group descriptor. Observe that this changes the list structure if the current group descriptor is stored to memory. Use with load_g_up.
save_up	Save up. Copies the current group descriptor address, rw_group, to the upper group descriptor pointer register, rw_group_up, and then keeps it when loading the lower group descriptor. Observe that this changes the list structure if the current group descriptor is stored to memory. Use with load_g_down.
update_down	Update down. Makes the DMA register, rw_group_down, point to the loaded context descriptor. Observe that this changes the list structure if the current group descriptor is stored to memory. Use with load_c_next and load_c_n.
restore	Restore context. Restarts transfer at rw_saved_data and rw_saved_data_buf. All data received since the position was last saved is discarded, and all transmitted data is resent. Internally buffered data is discarded. Used with ack_pkt.
burst	Start burst. Starts transfer from the buffer address in rw_saved_buf. Used with load_d.

next_pkt	Find the next packet. Current packet, and data internally buffered in the DMA, is discarded. The DMA finds the next data descriptor with out_eop set and loads the first data descriptor of next packet. If the burst flag is set the transfer starts from the buffer address in rw_data_buf. If eol is found before the beginning of a new packet, the context is disabled. Wait should be set in all data descriptors with out_eop set, to avoid multiple packets in the internal DMA buffer, since they will be discarded. Used with load_d.
store_md	Store meta data. Stores meta data to the current data descriptor. The meta data is sampled from the DMA client when the stream command is sampled by the DMA. Used with all stream commands except load_g.
store_c	Store the current context in the context descriptor. Stores status, the pointer to current data descriptor, the pointer to the current buffer position and the meta data. The meta data is sampled when the client strobes this stream command. Note that the values stored in the descriptor may differ from the values seen by SW in the DMA registers, due to internal buffering and speculative behavior of the DMA. Used with all stream commands except load_g.
store_g	Store the current group descriptor. Stores status, current pointers to upper and lower descriptors, where the lower may be a context descriptor or a group descriptor, and the meta data. The meta data is sampled when the client strobes this stream command. Used with all stream commands except load_g.

Table 5.8: *Stream Command Options*

Note The store_g and store_c options are generally executed before a command. If not, the exception to this rule is described with the detailed explanation of the command.

5.5.5.4 General Stream Commands

Table 5.9 explains the general stream commands and lists the options which can be used with it. For an explanation of the options, see table 5.8.

Stream Command	Description	Options
store_descr	No operation unless the options are set. The disable options are executed before the store options.	dis_g dis_c store_g store_c store_md
set_reg	Used to modify pointer registers, see section 5.5.5.4.1. For non pointer registers the command is a no operation.	store_g store_c store_md

Table 5.9: *General Stream Command*

5.5.5.4.1 Pointer registers

The following registers are pointer registers: rw_data, rw_data_next, rw_data_buf, rw_data_after, rw_ctxt, rw_ctxt_next, rw_saved_data, rw_saved_data_buf, rw_group, rw_group_next, rw_group_up, and rw_group_down. The value written is taken from the stream client, where it must be accessible. Otherwise an unknown value will be used, making access to pointer registers useless. For a DMA channel connected to the I/O Processor the value is taken from I/O Processor register iop_dmc_in.rw_ctxt_descr_md1 for a DMA input channel, and from iop_dmc_out.rw_ctxt_descr_md1 for a DMA output channel. The field rw_stream_cmd.n * 8 select which register is updated according to the offset field in 25.11 by the set_reg stream command.

5.5.5.5 Group Level Stream Commands

Table 5.10 explains the group level stream commands and lists the options which can be used with them. For an explanation of the options, see table 5.8.

Command	Description	Options
load_g	Load group descriptor. Loads group descriptor from the address in rw_group. rw_group must be preloaded.	
load_g_next	Load the next group descriptor. Loads group descriptor from the address in rw_group_next. This assumes a group descriptor already has been loaded into the DMA registers, and that the pointer to the next group is valid. The store options are executed before the loading of the next group descriptor. Note that eol in the current group descriptor is only used when the next_en option is true.	copy_up next_en array store_g store_c store_md
load_g_up	Load upper group descriptor. Loads group descriptor from the address in rw_group_up. This assumes a group descriptor already loaded into the DMA registers, and that the tol bit in current group descriptor is false. If tol in the current group descriptor is true no new group descriptor is loaded. The store options are always executed, and are executed before loading the upper group descriptor.	save_down copy_next store_g store_c store_md
load_g_down	Load lower group descriptor. Loads group descriptor from the address in rw_group_down. This assumes a group descriptor already loaded into the DMA registers, and that the bol bit in current group descriptor is false. If bol in the current group descriptor is true no new group descriptor is loaded. The store options are always executed, and are executed before loading the lower group descriptor.	save_up store_g store_c store_md

Table 5.10: *Group Level Stream Commands*

5.5.5.6 Context Level Stream Commands

Table 5.11 explains the context level stream commands and lists the options which can be used with them. For an explanation of the options, see table 5.8.

Command	Description	Options
load_c	Load context descriptor. Loads context descriptor from the address in rw_group_down. If a group level descriptor is not present, rw_group_down must be preloaded. The store options are executed before the loading of the context descriptor.	store_g store_c store_md
load_c_next	Load the next context descriptor. Loads context descriptor from the address in rw_ctxt_next. This assumes a context descriptor already has been loaded into the DMA registers, and that the pointer to the next context is valid. The store options are executed before the loading of the next context descriptor. Note that eol in the current context descriptor is only used when the next_en option is true.	update_down next_en array store_g store_c store_md
load_c_n	Load the n:th context descriptor. Random access in array of context descriptors. Loads context descriptor from the address in rw_group_down + sizeof(context descriptor) * n. To load the first context with this stream command, n must be zero. Use this stream command with caution. The store options are executed before the loading of the next group descriptor. Note that eol in the current context descriptor is only used when the next_en option is true.	update_down next_en array store_g store_c store_md

Table 5.11: *Context Level Stream Commands*

5.5.5.7 Data Level Stream Commands

Table 5.12 explains the data level stream commands and lists the options which can be used with them. For an explanation of the options, see table 5.8.

Command	Description	Options
ack_pkt	Acknowledge packet. Nop stream command making the transfer continue. Normally used to ack wait bit in data descr. Loads the next data descriptor. The store_g and store_md options are executed before the loading of the next data descriptor. The store_c option is executed after the loading of the next data descriptor. If eol is true in the current descriptor the context will be disabled and no new data descriptor will be loaded. The command options are executed in the following order: 1. store_g 2. store_md 3. restore or store_c	restore store_g store_c store_md

load_d	Load data descriptor. Loads data descriptor from address rw_saved_data. load_d assumes a context descriptor already loaded into the DMA registers. (note) The option next_pkt assumes a data descriptor already loaded into the DMA registers. The store_g and store_md options are executed before the data descriptor is loaded. When the option next_pkt is used, the option store_c is executed after the new data descriptor is loaded. The command options are executed in the following order: 1. store_g 2. store_md 3. next_pkt 4. store_c 5. burst	burst next_pkt store_g store_c store_md
set_w_size1	Set word size to 1 byte (default). May only be set at the packet boundary.	store_md store_c store_g
set_w_size2	Set word size to 2 bytes. May only be set at the packet boundary.	store_md store_c store_g
set_w_size4	Set word size to 4 bytes. May only be set at the packet boundary.	store_md store_c store_g

Table 5.12: *Data Level Stream Commands*

Note: An alternative to load a context descriptor is to let software write to rw_saved_data, rw_saved_data_buf, and rw_ctxt_ctrl.

5.5.5.8 Stream command ready

When a stream command is given, the stream client must wait for the ready condition, before evaluating status and examining meta data.

Table 5.14 defines how a stream client can detect when a stream command is ready, the resulting status of the stream command, and how the stream command change meta data.

In table 5.14 the following abbreviations of stream signals etc. are used. DMA registers are described in 25.11 , and I/O Processor DMC registers are described in 25.16 and 25.17 .

Abbreviations	Definition
busy	rw_stream_cmd.busy
grp.en	rw_group_ctrl.en
grp.tol	rw_group_ctrl.tol
grp.bol	rw_group_ctrl.bol
grp.dis	rw_group_stat.dis
ctxt.en	rw_ctxt_ctrl.en
ctxt.dis	rw_ctxt_stat.dis
next_en	Stream command option, See table 5.8

gmdv	Group Meta Data Valid as seen by the stream client. Eg. I/O Processor register field iop_dmc_in.r_stream_stat.group_md_valid for a DMA input channel, and iop_dmc_out.r_stream_stat.group_md_valid for a DMA output channel.
cmdv	Context Meta Data Valid as seen by the stream client. Eg. I/O Processor register field iop_dmc_in.r_stream_stat.ctxt_md_valid for a DMA input channel, and iop_dmc_out.r_stream_stat.ctxt_md_valid for a DMA output channel.
mdv	Meta Data Valid as seen by the stream client. Eg. I/O Processor register field iop_dmc_in.r_stream_stat.data_md_valid for a DMA input channel, and iop_dmc_out.r_stream_stat.data_md_valid for a DMA output channel.
toggle	First cleared then set, to always generate a positive transition. If initially set: 1->0->1 If initially clear: 0->0->1
at_eol	DMA has used all of the data list
IN	DMA input channel
OUT	DMA output channel

Table 5.13: *Abbreviations for Stream Command Ready table*

Command	Ready	Status OK	MD change, if any
store_descr	!busy	1	dis_g => gmdv toggle, dis_c => cmdv toggle, store_c & at_eol => cmdv toggle
set_reg	!busy	1	store_c & at_eol => cmdv toggle
load_g	!busy	1	gmdv toggle, cmdv cleared, mdv cleared
load_g_next	!busy	!next_en | (next_en & grp.en & !grp.dis)	gmdv toggle, cmdv cleared, mdv cleared
load_g_up	!busy	!grp.tol	gmdv toggle, cmdv cleared, mdv cleared
load_g_down	!busy	!grp.bol	gmdv toggle, cmdv cleared, mdv cleared
load_c	!busy	1	cmdv toggle, mdv cleared
load_c_next	!busy	!next_en | (next_en & ctxt.en & !ctxt.dis)	cmdv toggle, mdv cleared, next_en & (!ctxt.en | ctxt.dis) => gmdv toggle
load_c_n	!busy	!next_en | (next_en & ctxt.en & !ctxt.dis)	cmdv toggle, mdv cleared, next_en & (!ctxt.en | ctxt.dis) => gmdv toggle
ack_pkt	!busy + 20ns NOT TESTED	1	IN: mdv toggle, OUT: mdv toggle at pkt start out of internal buffer, see section 5.5.5.8.1, store_c & at_eol => cmdv toggle

load_d	!busy + 20ns NOT TESTED	IN: 1 OUT: !next_pkt | (next_pkt & mdv)	mdv toggle
set_w_size1	!busy	1	store_c & at_eol => cmdv toggle
set_w_size2	!busy	1	store_c & at_eol => cmdv toggle
set_w_size4	!busy	1	store_c & at_eol => cmdv toggle

Table 5.14: *Stream Command Ready*

5.5.5.8.1 ack_pkt and mdv

The mdv field toggles when the beginning of a new packet is visible to the stream client . If ack_pkt is given as response to a DMA request to the stream client , eg. iop_dmc_out.r_stream_stat.cmd_rq, the internal buffer is empty and mdv will toggle. This is the normal case for a stream client.

5.5.5.9 rw_stream_cmd MACRO

See [DMA_MACROS] and [DMA_REG_MACROS].

```
// Wait until the busy-bit is unset, then writes the command to
// rw_stream_cmd.
//
DMA_WR_CMD( instance, command );
```

5.5.6 Descriptor Format

5.5.6.1 Data Descriptor

A data descriptor has a 16 byte memory area which must be located within a single cache line, i.e. if bit 4 in addr below is set, bits 3 to 0 must be zero.

Data descriptor format:

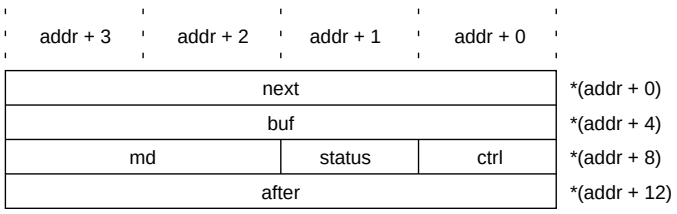


Figure 5.5: *Data descriptor format*

Field	Size	Description
next	dword	Pointer to the next data descriptor.

buf	dword	Pointer to first byte of the data buffer.
ctrl	byte	Control field read by the DMA.
status	byte	Status field written to by the DMA.
md	word	Meta data. Sent to the client at the beginning of each transfer. Stored when the client signals eop, or gives a command that requires storage.
after	dword	Pointer to the byte after the last byte of the data buffer.

Table 5.15: *Data Descriptor Format*

Data descriptor **ctrl** field format:

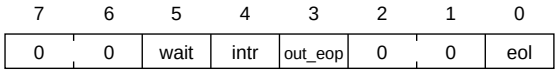


Figure 5.6: *Data descriptor control field*

Bit	Name	Description
7-6		Reserved.
5	wait	Initiates wait_ack sequence. Makes the DMA stop and wait for a stream command, after the last readable data of the DMA buffer has been sent to the client. Out channel only. Ignored by in channel.
4	intr	Data descriptor interrupt generated when the next data descriptor is loaded. If the eol bit in the ctrl field of this data descriptor is set to one, i.e. there is no next data descriptor, the interrupt is generated after this data descriptor is fully processed.
3	out_eop	This is the last descriptor of packet to be transmitted. Out channel only. Ignored by in channel.
2-1		Reserved.
0	eol	Last descriptor of the list.

Table 5.16: *Data Descriptor ctrl Field Format*

Data descriptor **status** field format:

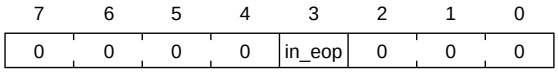


Figure 5.7: *Data descriptor status field*

Bit	Name	Description
7-4		Reserved.
3	in_eop	This is the last descriptor of the received packet. In channel only. Ignored by out channel
2-0		Reserved.

Table 5.17: *Data Descriptor status Field Format:*

5.5.6.2 Context Descriptor

A context descriptor is a 32 byte memory area which must be located within a single cache line, i.e. bits 4..0 in addr below must be zero.

General context descriptor format:

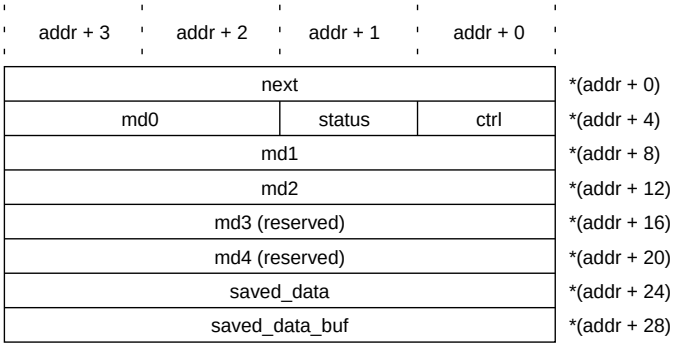


Figure 5.8: *General Context Descriptor Format*

Field	Size	Description
next	dword	Pointer to the next context descriptor.
ctrl	byte	Control field read by the DMA.
status	byte	Status field written to by the DMA.
md0	word	Meta data 0. Sent to the client when the DMA context descriptor registers are updated. Stored when the client gives a command that requires storage.
md1	dword	Meta data 1. Sent to the client when the DMA context descriptor registers are updated. Stored when the client gives a command that requires storage.
md2	dword	Meta data 2. Sent to the client when the DMA context descriptor registers are updated. Stored when the client gives a command that requires storage.
md3	dword	Reserved.
md4	dword	Reserved.
saved_data	dword	Pointer to saved data descriptor.
saved_data_buf	dword	Pointer to the last saved position of the data buffer.

Table 5.18: *General Context Descriptor Format*

General context descriptor **ctrl** field format:

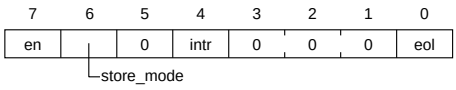


Figure 5.9: *General context descriptor ctrl field format*

Bit	Name	Description
7	en	Indicates if this context descriptor is enabled.
6	store_mode	Indicates that the context descriptor may be stored at any time during data transfer(1), or at wait_ack only(0). The buffer space must be at least 68 bytes to guarantee full DMA performance, when a context descriptor is to be stored at any time. Out channel only. Ignored by in channel.
5		Reserved.
4	intr	Indicates that an interrupt is generated when the next or n:th context descriptor is loaded.
3-1		Reserved.
0	eol	This is the last descriptor of the list.

Table 5.19: *General context descriptor ctrl field format*

General context descriptor **status** field format:

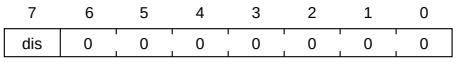


Figure 5.10: *General context descriptor status field format*

Bit	Name	Description
7	dis	Disable. Indicates if this context descriptor has been disabled by the DMA. This bit has higher priority than the en bit of the ctrl field.
6-0		Reserved.

Table 5.20: *General Context Descriptor status Field Format*

5.5.6.3 Group Descriptor

A group descriptor has a 16 byte memory area which must be located within a single cache line, i.e. if bit 4 in addr below is set, bits 3 to 0 must be zero.

Group Descriptor Format:

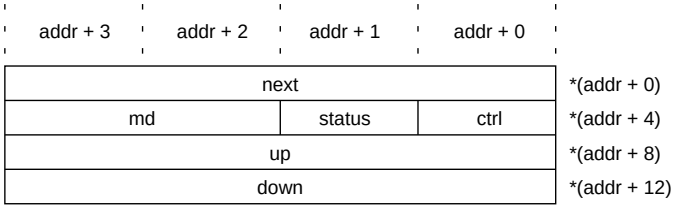


Figure 5.11: *Group descriptor format*

Field	Size	Description
next	dword	Pointer to the next group descriptor at the same level of the group descriptor hierarchy.
ctrl	byte	Control field read by the DMA.
status	byte	Status field written to by the DMA.
md	dword	Meta data. Sent to the client when the DMA group descriptor registers are updated. Stored when the client gives a command that requires storage.
up	dword	Pointer to group descriptor at level above the present group in the group hierarchy.
down	dword	Pointer to group descriptor at level beneath the present group in the group hierarchy. In a leaf group descriptor down is a pointer to a context descriptor.

Table 5.21: *Group descriptor format*

Group descriptor **ctrl** field format:

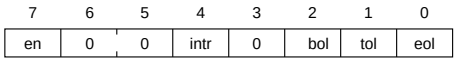


Figure 5.12: *Group descriptor ctrl field format*

Bit	Name	Description
7	en	Indicates if the group descriptor is enabled.
6-5		Reserved.
4	intr	Indicates that an interrupt is generated when the next, upper or lower group descriptor is loaded.
3		Reserved.
2	bol	Current group descriptor is at the lowest level.
1	tol	Current group descriptor is at the highest level.
0	eol	End of group descriptor list at the current level.

Table 5.22: *Group Descriptor ctrl Field Format*

Group descriptor **status** field format:

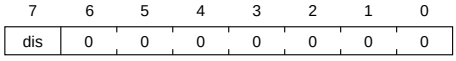


Figure 5.13: *Group descriptor status field format*

Bit	Name	Description
7	dis	Indicates if this group descriptor is disabled by the DMA. This bit has higher priority than the en bit of the ctrl field.
6-0		Reserved.

Table 5.23: *Group descriptor status field format*

5.5.6.4 Examples

Examples of DMA descriptor and list initialization, and stream command usage. The examples use the C structs and macros defined in [DMA_MACROS].

5.5.6.4.1 Data Level List Setup

The following code shows the initialization of the in and out channel list described in section 5.4.1:

```
#define NBR_OF_CONTEXTS_IN_ARRAY 1
#define NBR_OF_DATA 3
#define BUFFER_SIZE 128
 dma_descr_context c[NBR_OF_CONTEXTS_IN_ARRAY];
 dma_descr_data data[NBR_OF_CONTEXTS_IN_ARRAY * NBR_OF_DATA];
 char buffers[NBR_OF_DATA * BUFFER_SIZE];
```

Out channel data level list setup:

```
void init_out_channel_list() {
  c[0].next           = NULL;
  c[0].eol            = 0;
  c[0].intr           = 0;
  c[0].store_mode     = 1;
  c[0].en             = 1;
  c[0].dis            = 0;
  c[0].md0            = Out_List_meta_data_0;
  c[0].md1            = Out_List_meta_data_1;
  c[0].md2            = Out_List_meta_data_2;
  c[0].saved_data     = &data[0];
  c[0].saved_data_buf = &buffers[first_byte_of_packet_1];

  data[0].next      = &data[1];
  data[0].buf       = &buffers[first_byte_of_packet_1];
  data[0].eol       = 0;
  data[0].out_eop   = 0;
  data[0].intr      = 1;
  data[0].wait      = 0;
  data[0].md        = packet_1_meta_data;
  data[0].after     = &buffers[first_byte_of_packet_1 + BUFFER_SIZE];

  data[1].next      = &data[2];
  data[1].buf       = &buffers[first_byte_of_packet_1 + BUFFER_SIZE];
  data[1].eol       = 0;
  data[1].out_eop   = 1;
  data[1].intr      = 0;
  data[1].wait      = 1;  // wait for client to give stream command
  data[1].md        = packet_1_meta_data;
  data[1].after     = &buffers[last_byte_of_packet_1 + 1];
```

```
  data[2].next      = NULL;
  data[2].buf       = &buffers[first_byte_of_packet_2];
  data[2].eol       = 1;
  data[2].out_eop   = 1;
  data[2].intr      = 0;
  data[2].wait      = 1;  // wait for client to give stream command
  data[2].md        = packet_2_meta_data;
  data[2].after     = &buffers[last_byte_of_packet_2 + 1];
}
```

In channel data level list setup:

```
void init_in_channel_list() {
  c[0].next           = NULL;
  c[0].eol            = 0;
  c[0].intr           = 0;
  c[0].store_mode     = 1;
  c[0].en             = 1;
  c[0].dis            = 0;
  c[0].md0            = In_List_meta_data_0;
  c[0].md1            = In_List_meta_data_1;
  c[0].md2            = In_List_meta_data_2;
  c[0].saved_data     = &data[0];
  c[0].saved_data_buf = &buffers[In_packet_buffer];

  data[0].next      = &data[1];
  data[0].buf       = &buffers[In_packet_buffer];
  data[0].eol       = 0;
  data[0].intr      = 1;
  data[0].in_eop    = 0;
  data[0].md        = In_List_meta_data;
  data[0].after     = &buffers[In_packet_buffer + 1 * BUFFER_SIZE];

  data[1].next      = &data[2];
  data[1].buf       = &buffers[In_packet_buffer + 1 * BUFFER_SIZE];
  data[1].eol       = 0;
  data[1].intr      = 0;
  data[1].in_eop    = 0;
  data[1].md        = In_List_meta_data;
  data[1].after     = &buffers[In_packet_buffer + 2 * BUFFER_SIZE];

  data[2].next      = NULL;
  data[2].buf       = &buffers[In_packet_buffer + 2 * BUFFER_SIZE];
  data[2].eol       = 1;
  data[2].intr      = 0;
  data[2].in_eop    = 0;
  data[2].md        = In_List_meta_data;
  data[2].after     = &buffers[In_packet_buffer + 3 * BUFFER_SIZE];
 }
```

5.5.6.4.2 Data List Modification

This example assumes the DMA has stored one packet in data[0] and one packet in data[1] above, it is also assumed SW has processed the contents of the packets of data[0] and data[1]. The code below re-initializes data[0] and data[1] and append them at the end of the list above. The resulting list then is data[2], data[0] and data[1].

In channel reuse processed data descriptors and buffers:

```
// Re-initialize the fields that were effected by the previous transfer

data[0].in_eop    = 0;
data[0].md        = In_List_meta_data;
data[0].after     = &buffers[In_packet_buffer + 1 * BUFFER_SIZE];

data[1].in_eop    = 0;
data[1].md        = In_List_meta_data;
data[1].after     = &buffers[In_packet_buffer + 2 * BUFFER_SIZE];

// Make data[1] the eol descriptor.

data[1].eol       = 1;

// Update the former eol descriptor with next-pointer and unset eol.

data[2].next      = &data[0];
data[2].eol       = 0;

DMA_CONTINUE_DATA( dma_x ); // Make the DMA reload the last descriptor.
```

Out channel allocate more data descriptors and buffers:

```
// Create a new list
dma_descr_data new_data;
char new_buffer[BUFFER_SIZE];

new_data.next      = NULL;
new_data.buf       = &new_buffer[0];
new_data.eol       = 1;
new_data.out_eop   = 1;
new_data.intr      = 0;
new_data.wait      = 1;
new_data.md        = new_packet_meta_data;
new_data.after     = &new_buffer[BUFFER_SIZE];

// Update the former eol descriptor, created by init_out_channel_list(),
// with next-pointer and unset eol.

data[2].next      = &new_data;
data[2].eol       = 0;
```

```
  // This should work independent of the current DMA state.

  DMA_CONTINUE_DATA( dma_x );
```

5.5.6.4.3 Data List Modification and Multiple Contexts

When the DMA is working with multiple contexts the procedure to append new data to a data list has to be extended to assure the DMA hardware and the DMA data and context lists are synchronized. The procedure is as follows

```
  // Tell DMA there is more data
  DMA_CONTINUE_DATA( dma_x );

  // Wait for continue data to take effect
  reg_dma_rw_stat r;
  r = REG_RD( dma, dma_x, rw_stat);
  while( r.mode != regk_dma_running );

  DMA_WR_CMD( dma_x, regk_dma_store_descr ); // store_descr without options
                                            // is a NOP, that is guaranteed
                                            // to execute after continue data.

  // Mark the context as "enabled"
  //
  // Here the DMA may already have consumed the new list and disabled
  // the context, but it is OK to enable it anyway. In this case the DMA
  // will try to run the falsely enabled context next time, and disable
  // it again.
  c[0].dis = 0;
```

5.5.6.4.4 Context Level List Setup

The following code shows the initialization of the list described in section 5.4.2:

```
#define NBR_OF_CONTEXTS_IN_ARRAY 3
#define NBR_OF_DATA 1
#define BUFFER_SIZE 128

dma_descr_context c[NBR_OF_CONTEXTS_IN_ARRAY];

dma_descr_data data[NBR_OF_CONTEXTS_IN_ARRAY * NBR_OF_DATA];

void init_list() {
  c[0].next           = &c[1];
  c[0].eol            = 0;
  c[0].intr           = 0;
  c[0].store_mode     = 1;
  c[0].en             = 1;
```

```
  c[0].dis            = 0;
  c[0].md0            = Context0_meta_data_0;
  c[0].md1            = Context0_meta_data_1;
  c[0].md2            = Context0_meta_data_2;
  c[0].saved_data     = &data[0];
  c[0].saved_data_buf = NULL;

  c[1].next           = &c[2];
  c[1].eol            = 0;
  c[1].intr           = 0;
  c[1].store_mode     = 1;
  c[1].en             = 0;
  c[1].dis            = 0;
  c[1].md0            = Context1_meta_data_0;
  c[1].md1            = Context1_meta_data_1;
  c[1].md2            = Context1_meta_data_2;
  c[1].saved_data     = NULL;
  c[1].saved_data_buf = NULL;

  c[2].next           = &c[0];
  c[2].eol            = 1;
  c[2].intr           = 0;
  c[2].store_mode     = 1;
  c[2].en             = 0;
  c[2].dis            = 0;
  c[2].md0            = Context2_meta_data_0;
  c[2].md1            = Context2_meta_data_1;
  c[2].md2            = Context2_meta_data_2;
  c[2].saved_data     = NULL;
  c[2].saved_data_buf = NULL;


  data[0].next      = NULL;
  data[0].buf       = NULL;
  data[0].eol       = 0;
  data[0].out_eop   = 0;
  data[0].intr      = 0;
  data[0].wait      = 0;
  data[0].in_eop    = 0;
  data[0].md        = Data0_meta_data;
  data[0].after     = NULL;
};
```

5.5.6.4.5 Context List Modification

When modifying the context list the DMA should be stopped to prevent any modification initiated by the DMA. See [DMA_MACROS] and [DMA_REG_MACROS].

```
  // Stop the DMA.

  DMA_STOP( dma_x );
```

```
  // Check that the DMA has stopped.

  reg_dma_rw_stat r;
  r = REG_RD( dma, dma_x, rw_stat);
  while( r.mode != regk_dma_stopped );

  // The current context pointer may be examined.

  REG_RD( dma, dma_x, rw_ctxt);

  // Modify context list.

  c[1].next           = &c[2];
  c[1].saved_data     = &new_data_descr;
```

The next steps depend on the desired behavior.

One way is to simply start the DMA again.

```
  DMA_CONTINUE( dma_x );
```

To check that the DMA is running again.

```
  reg_dma_rw_stat r;
  r = REG_RD( dma, dma_x, rw_stat);
  while( r.mode != regk_dma_running );
```

5.5.6.4.6 Group Level List Setup

The following code shows the initialization of the list described in section 5.4.3:

```
#define NBR_OF_GROUPS 4
#define NBR_OF_CONTEXTS 4
dma_descr_group g[NBR_OF_GROUPS];
dma_descr_context c[NBR_OF_CONTEXTS_IN_ARRAY];
dma_descr_data d[NBR_OF_CONTEXTS_IN_ARRAY * NBR_OF_DATA];
char b[NBR_OF_DATA * BUFFER_SIZE];

void init_list() {
  g[0].next         = &g[1];
  g[0].eol          = 0;
  g[0].tol          = 1;
  g[0].bol          = 0;
  g[0].intr         = 0;
  g[0].en           = 1;
  g[0].dis          = 0;
  g[0].md           = Group0_meta_data;
  g[0].down.group   = &g[2];
```

```
g[1].next         = &g[0];
g[1].eol          = 1;
g[1].tol          = 1;
g[1].bol          = 1;
g[1].intr         = 0;
g[1].en           = 1;
g[1].dis          = 0;
g[1].md           = Group1_meta_data;
g[1].down.context = &c[3];

g[2].next         = &g[3];
g[2].eol          = 0;
g[2].tol          = 0;
g[2].bol          = 1;
g[2].intr         = 0;
g[2].en           = 1;
g[2].dis          = 0;
g[2].md           = Group2_meta_data;
g[2].up           = &g[0];
g[2].down.context = &c[0];


g[3].eol          = 1;
g[3].tol          = 0;
g[3].bol          = 1;
g[3].intr         = 0;
g[3].en           = 1;
g[3].dis          = 0;
g[3].md           = Group3_meta_data;
g[3].down.context = &c[2];

c[0].next           = &c[1];
c[0].eol            = 0;
c[0].intr           = 0;
c[0].store_mode     = 1;
c[0].en             = 0;
c[0].dis            = 0;
c[0].md0            = Context0_meta_data_0;
c[0].md1            = Context0_meta_data_1;
c[0].md2            = Context0_meta_data_2;

c[1].next           = &c[2];
c[1].eol            = 0;
c[1].intr           = 0;
c[1].store_mode     = 1;
c[1].en             = 0;
c[1].dis            = 0;
c[1].md0            = Context1_meta_data_0;
c[1].md1            = Context1_meta_data_1;
c[1].md2            = Context1_meta_data_2;

c[2].eol            = 1;
c[2].intr           = 0;
```

```
  c[2].store_mode     = 1;
  c[2].en             = 0;
  c[2].dis            = 0;
  c[2].md0            = Context2_meta_data_0;
  c[2].md1            = Context2_meta_data_1;
  c[2].md2            = Context2_meta_data_2;


  c[3].eol            = 1;
  c[3].intr           = 0;
  c[3].store_mode     = 1;
  c[3].en             = 0;
  c[3].dis            = 0;
  c[3].md0            = Context3_meta_data_0;
  c[3].md1            = Context3_meta_data_1;
  c[3].md2            = Context3_meta_data_2;
 };
```

5.5.6.4.7 Group List Modification

When modifying the context list the DMA should be stopped to prevent any modification initiated by the DMA. See [DMA_MACROS] and [DMA_REG_MACROS].

```
  // Stop the DMA.

  DMA_STOP( dma_x );

  // Check that the DMA has stopped.

  reg_dma_rw_stat r;
  r = REG_RD( dma, dma_x, rw_stat);
  while( r.mode != regk_dma_stopped );

  // The current group pointer may be examined.

  REG_RD( dma, dma_x, rw_group);

  // Modify group list.

  g[1].bol          = 0;
  g[1].intr         = 1;
  g[1].md           = New_meta_data;
  g[1].down.group   = &g[3];
```

The next steps depend on the desired behavior.

One way is to simply start the DMA again.

```
  DMA_CONTINUE( dma_x );
```

To check that the DMA is running again.

```
reg_dma_rw_stat r;
r = REG_RD( dma, dma_x, rw_stat);
while( r.mode != regk_dma_running );
```

Chapter 6

Boot Methods

6.1 Bootstrap Methods

There are several different methods to bootstrap the ETRAX FS. They are presented in table 6.1. The selection of bootstrap mode is done by pins **bs2-bs0** which are sampled upon reset going inactive. These values are also stored in the internal register field config.r_bootsel.boot_mode. The boot sequence is performed by a program running in ROM in the ETRAX FS.

Pins **bs2-bs0**	Bootstrap method
000	NOR flash
001	Network rx
010	Network tx/rx
011	NAND flash
100	Serial
101	Master
110	Slave
111	No boot

Table 6.1: *Overview of the Bootstrap Methods*

6.2 Initialization

The following initialization is always performed:

- The caches are initialized and enabled.
- The clock region for the bus interface is always enabled in register rw_clk_ctrl.

6.3 Flash

NOR and NAND flash boot is supported.

6.3.1 Empty flash

If the start of the chosen flash is six consecutive 0xff, indicating a MOVEM R15,0xffffffff instruction which could normally only occur in empty flash, the boot mode will be changed to Network rx.

6.3.2 NOR flash

Execution starts at cached address 0x0.

6.3.2.1 Bus width

NOR flash bus width is set from the **bs6** pin, sampled upon reset going inactive. This value is also saved in the internal register r_bootsel.

bs6	Bus width
0	16
1	32

6.3.3 NAND flash

When booting from an 8-bit NAND flash the first 127 KB of the flash is copied to the beginning of the internal RAM where it is called. If the NAND flash is a 16-bit device the copying will only read the lower byte in each 16-bit word from the lower 254 KB of the flash.

NAND flash memories that require a new page address for each new page read will only supply the first page. This is not the normal behavior for NAND flash memories though.

The spare area in the flash is not used.

6.3.3.1 NAND flash connection

The NAND flash has to be connected in the following way to facilitate boot:

NAND flash pin	Connected to
$\overline{\text{CE}}$	**pa4**
$\overline{\text{RE}}$	**csp0_n**
$\overline{\text{WE}}$	**csp1_n**
$\overline{\text{SE}}$	pull up

CLE	**pa5**
ALE	**pa6**
$\overline{\text{WP}}$	pull up
RY	**pa7**

Table 6.3: *NAND flash connection*

6.3.3.2 Address burst length

NAND flashes of different sizes may require different number of address bursts to read data. This is shown in the data sheet for the flash.

bs3	**bs4**	Cycles
0	0	3
1	0	4
1	1	5

The **bs3** pin is also used by network boot, and is only used for address burst length when NAND flash boot is selected.

6.3.3.3 Read command end

Some NAND flashes require an extra command (0x30) to end a read command. This is shown in the data sheet for the flash. Pin **bs6** is used to select if this extra command should be inserted at the end of a read command.

bs6	0x30 cmd
0	off
1	on

The **bs6** pin is also used by NOR flash boot, and is only used for 0x30 command insertion when NAND flash boot is selected.

6.4 Network

6.4.1 Initialization

If any of the network boot methods has been chosen the following initialization is done:

- The 25Mhz transceiver clock on pin **e0phyclk** is started.
- The boot sequence will set pin **phyrst_n** high at a time >200 μs and <300 μs after the transciever clock has been started. This can be used to get a correct reset of network transceivers that require a clock for some time before reset is released.

6.4.2 Network rx

During network boot the ETRAX FS is listening for a specially formatted packet (table 6.6) on Ethernet interface 0. The packet is downloaded to the internal RAM at 0x38000000 and the execution starts at the data part of the packet, at address 0x3800001e.

Byte nr	Content (hex)	Description
0-5	01 40 8c 00 02 00	Destination address. This is a multicast address within the Axis Ethernet address block.
6-11	XX XX XX XX XX XX	Source address. The address of the host transmitting the bootstrap packet. This address is not checked.
12-13	type-length	This is currently not checked but it is recommended that the contents follow the 802.3 standard.
14-21	AA AA 03 00 40 8C 88 56	A SNAP header featuring the Axis vendor code (same as the Ethernet address block). This is the Axis ether-type specifically assigned for this purpose.
22-25	FF FF FF FF	A tag signaling this packet as a bootstrap datagram.
26-29	00 00 00 00	The bootstrap packet sequence number. The number must consist of only zeros in the first packet in the bootstrap sequence.
30-	XX ...	Payload data to be executed. Max 1484 bytes long.

Table 6.6: *Network bootstrap Ethernet header*

6.4.3 Network tx/rx

The ETRAX FS can also transmit a specific packet during network boot to signal that it is ready to accept a network boot packet. The transmission is repeated at about 0.1 second intervals for the first ten seconds after ETRAX FS reset and then at 1*n s starting with n=1 until a boot packet is received. The reception works just as when Network rx boot is selected.

Byte nr	Content (hex)	Description
0-5	01 40 8c 00 04 00	Destination address.
6-11	01 40 8c 00 02 00	Source address.
12-13	00 40	Length of the packet.
14-21	AA AA 03 00 40 8C 88 56	The SNAP header.
22-25	FF FF FF FF	Bootstrap datagram tag.
26-29	00 00 00 00	The bootstrap packet sequence number. Increased for each packet transmitted.
30-60	XX ...	Random padding.

Table 6.7: *Network boot transmitted Ethernet header*

6.4.4 Duplex

Full/half duplex is selected with **bs3**.

bs3	Duplex
0	half
1	full

Table 6.8: *Network duplex selection*

6.5 Serial

Serial port 0 is used and is configured at 115200 baud, 8 bits with no parity, one start bit and one stop bit. A total of 1024 bytes will be downloaded to the internal RAM start plus offset 0x1e, to get the same code start address as during network boot, giving 0x3800001e where execution starts.

If the PLL is turned off, the configured baud rate will be 9600 for an input clock of 12 MHz.

6.6 Master chip boots slave

This is a bootstrap method for multi-chip designs where a chip configured as bus master downloads a program to a slave chip. See chapter 4 and 4.3.12 for more information about master/slave bus interface.

A user program in the master downloads a program to a specific slave's internal RAM at 0x3800001e. After the boot program is loaded, the master sets boot_rdy in the slave.

The boot ROM code in the slave polls boot_rdy, and jumps to 0x3800001e when the bit is set.

The size of the data downloaded by the master must not be larger than 127 KB. There is no minimum size.

6.7 Slave chip boots master

A user program in the slave puts a boot program for the master in the lower 127 KB of the internal RAM of the slave, and then sets boot_rdy in itself.

The boot ROM code in the master polls boot_rdy in the slave. When it is set, the boot ROM program in the master copies the first 127 KB program from the slave over to its own internal RAM and starts to execute it at 0x3800001e.

It is recommended that the loaded boot program in the master clears the boot_rdy flag in the slave to signal that the boot sequence is finished.

6.8 No boot and JTAG boot

Apart from the previously described boot methods it is possible to boot the ETRAX FS via the JTAG interface and the guru mode code located in the same ROM as the boot code. This is why there is a *No boot* mode available which would be a appropriate mode to use for that. This mode will just do an eternal nop loop.

The code for guru mode is described in 17.

6.9 PLL mode

If the PLL should be turned on or not by the boot ROM is selected by pin **bs5**, sampled upon reset going inactive. This value is also saved in register rw_clk_ctrl.pll.

bs5	PLL
0	off
1	on

Chapter 7

MMU

7.1 References

Reference	Description
[CRIS]	CPU, chapter 2
[REGS]	Support registers, chapter 25.37
[MACROS]	CRIS v32 support function register access macros http://developer.axis.com
[DEFS]	MMU support function register constants and data types http://developer.axis.com

7.2 Overview

The CPU is connected to two separate Memory Management Units (MMUs) that manages the physical memory resources available in the system. An instruction MMU is connected between the CPU and the instruction cache and a data MMU is connected between the CPU and the data cache. Both MMUs use the same behavior and this description can be applied to both the instruction and the data MMU.

The MMU has the following purposes:

- Protecting the code and data of one user-level application from other user-level applications.
- Permitting user-level applications to share portions of the address space.
- Protecting the kernel-level code and data from user-level applications.
- Running applications that are partially resident in main memory. Only the most recent part of such an application is normally stored in main memory: the rest of the program is stored on disk until needed. This is more commonly known as paging.
- Reference counting in support of e.g. a garbage collection mechanism.

The MMU implements a virtual memory system where it creates the illusion of a very large amount of memory exclusively available for each application. The MMU is used to translate addresses within the virtual memory space into physical addresses that maps to the physical memory available in the system.

7.3 Functional description

7.3.1 Non-protected mode

By default the MMU is disabled after reset. In this mode the CPU is considered to be in non-protected mode and memory is accessed in the conventional manner. All addresses are treated as physical addresses and will pass through the MMU without any translation and protection mechanisms.

The MMU can be enabled in a general configuration register in the CPU. See the CPU documentation 2 for further details.

7.3.2 Physical memory

The caches in the system are physically addressed. A 32-bit virtual CPU address is always translated by an MMU into a 32-bit physical address before being used in any cache. The most significant bit of the translated physical address (bit 31) is used to select between cached and non-cached accesses. Therefore it is only possible to address 2GByte of physical memory in the system. The physical memory space is divided into several smaller regions to select between different types of external and internal memory regions.

7.3.3 Virtual memory

The CPU uses the whole 32-bit address region to define a 4GByte virtual memory space. Each application is given a separate 8-bit address space identifier that is used by the MMU to separate its memory from other applications. Therefore 4GBytes of virtual memory can be made available to each application.

The virtual memory is divided into 8 KByte pages which can be individually protected and mapped to physical memory. The upper 19-bit part of the 32-bit CPU address is known as the Virtual Page Number (VPN), while the lower 13-bit part is used as an offset within each page. The VPN is translated into a 19-bit Physical Frame Number (PFN) while the page offset remains unchanged through the MMU. The MMU uses the CPU's User and Kernel Modes to restrict access and select the appropriate mapping from virtual to physical addresses in the virtual memory space.

Mapping from virtual address to physical address is handled by a Translation Lookaside Buffer (TLB). The TLB is an on-chip cache that provides translations in the form of TLB entries. Details about the TLB can be found in chapter 7.3.4.

If a virtual-to-physical address translation is not found in the TLB, the MMU will generate an exception to the CPU. The kernel software is then responsible for finding

a correct translation and update the TLB.

The MMU operates with two types of virtual memory areas:

- Kernel/user area - accessed in the CPU User and Kernel Modes, and mainly contains user code and data structures.
- Kernel area - accessed in the CPU Kernel Mode only and may include kernel code and data structures, I/O-buffers, DMA-buffers, mode registers, etc.

7.3.3.1 Kernel/User area

The kernel/user area is a uniform, virtual address area of 4 GBytes in size. It is divided into 8 KByte pages that can be individually protected and mapped to physical memory. Mapping is executed through the TLB, which translates a virtual address into a corresponding physical address.

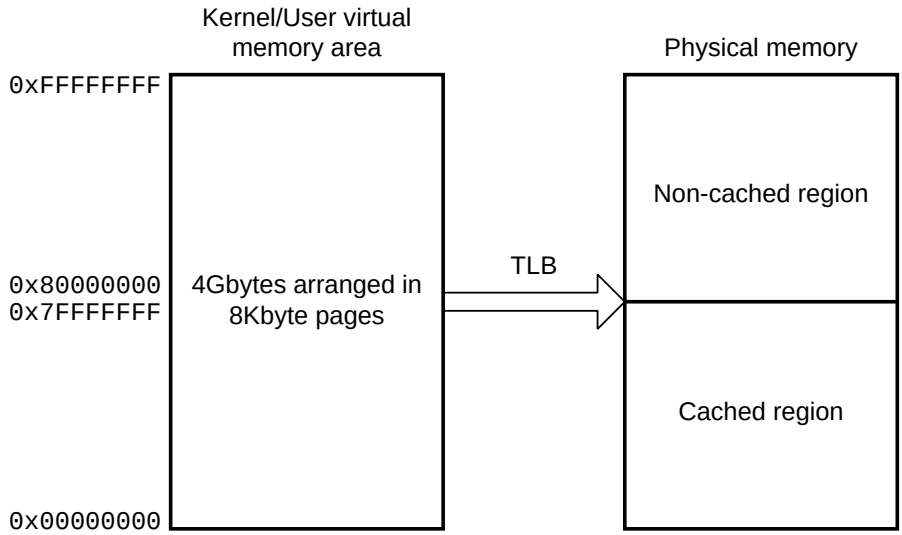


Figure 7.1: *Kernel/User virtual memory area*

The lower 13 bits of the 32-bit virtual address is only used as an offset within each 8Kbyte page. The page offset is not changed by the translation while the 19-bit virtual page number is translated into a 19-bit physical frame number.

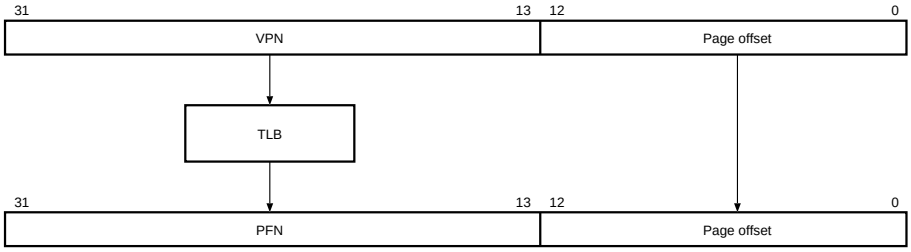


Figure 7.2: *TLB Address translation*

7.3.3.2 Kernel area

The kernel area is divided into sixteen 256 MByte segments designated seg_0 to seg_f. Each segment can be individually configured in the rw_mm_cfg register to use either page mapping or linear segment mapping.

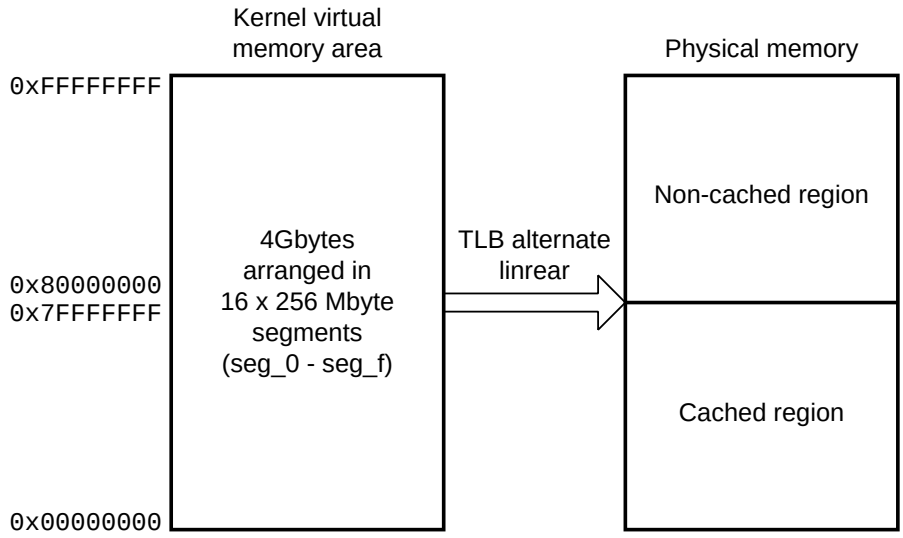


Figure 7.3: *Kernel virtual memory area*

Segments that use page mapping are divided into 8 KByte pages and translated by the TLB.

The virtual addresses of segments that use linear segment mapping are converted to physical addresses by translating the upper four bits of the 32-bit virtual address coming from the CPU. The linear translation is defined in two registers, rw_mm_kbase_hi and rw_mm_kbase_lo. The 4-bit base field in these registers becomes address bits 31 to 28 when translating to physical addresses. Bits 27 to 0 of the virtual address are unchanged when translated to a physical address.

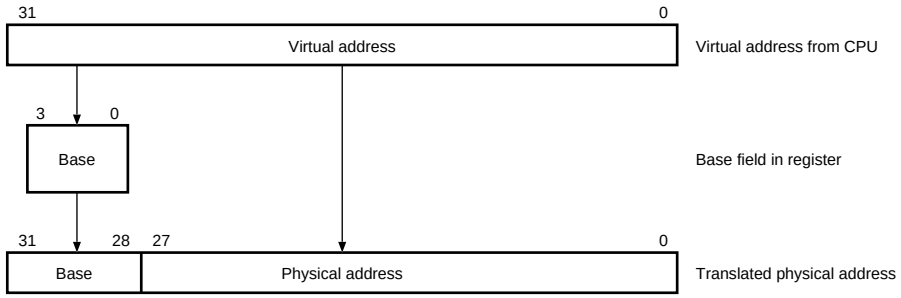


Figure 7.4: *Linear segment address translation*

7.3.4 Translation lookaside buffer

The Translation Lookaside Buffer (TLB) is a 64-entry cache that maps a virtual address to a physical address. If a translation cannot be found in the TLB (i.e. a TLB miss), an exception is generated to the CPU and the software is required to load a new translation into the TLB.

7.3.4.1 TLB Memory sets

The TLB comprises four 16-entry memory sets designated 0 to 3. The TLB is thus a four-way set associative memory. This means that a VPN from the CPU can be stored at only one location in each of the four TLB memory sets, and in each set the chosen location is the same. An example of possible locations for different addresses in the TLB is given in the table below.

	Possible positions in TLB			
VPN	Set0	Set0	Set0	Set0
0	0	16	32	48
1	1	17	33	49
2	2	18	34	50
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
14	14	30	46	62
15	15	31	47	63
16	0	16	32	48
17	1	0	33	49

Table 7.2: *Possible virtual address positions in the TLB*

The selection of the TLB set in which a particular VPN will be stored must be configured in software. The location at which the VPN will be placed within the TLB set is determined by a 4-bit index comprising bits 13 to 16 of the incoming virtual address from the CPU.

As an example, the address 0x400001af have bits 13 to 16 set to 0 and can therefore be placed at address 0 in any of the four TLB sets (position 0, 16, 32 or 48 of the TLB). Address 0x400021af have bits 13 to 16 set to 1 and can be placed at address 1 in any of the four sets (position 1, 17, 33 or 49 of the TLB).

In the event of a TLB miss, a 2-bit set selection number is provided by a random number generator. The 2-bit random number is combined with the 4-bit index from the CPU address to form a 6-bit index. This index is stored by the MMU in register rw_mm_tlb_sel, and can be used to choose which of the 64 TLB entries to replace. It is also possible for the software to use another algorithm to select the entry to replace. As shown below, bits 4 and 5 of the index provide the TLB set number and bits 3 to 0 denote the location within the set.

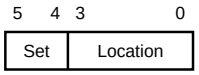


Figure 7.5: *TLB entry select index format*

7.3.4.2 TLB Entries

Entries stored in the TLB are 47 bits in width. They consist of a virtual to physical address translation, a page identification, and control bits. All fields have a corresponding field in the two registers rw_mm_tlb_lo and rw_mm_tlb_hi. Note that the bit positions of the registers are not the same as within the TLB entry.

The format of a TLB entry is shown below.

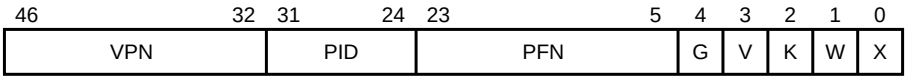


Figure 7.6: *TLB entry format*

VPN The virtual address expressed as a 15-bit Virtual Page Number. The remaining 4 bits are composed from the 4-bit TLB set index.

PID 8-bit page identification

PFN The translated address expressed as a 19-bit Physical Frame Number.

G Global bit. If set, the TLB ignores the page identification matching conditions for a TLB hit.

V Valid bit. If set, it indicates that the TLB entry contains a valid address translation.

K Kernel bit. If set, it prevents access to the page during CPU User Mode.

W Write Enable bit. If cleared, the page is write protected.

X Execute bit. If clerared, the page is protected from execution.

7.3.4.3 TLB Lookup mechanism

When the CPU generates a new address, the four TLB sets are searched for a matching VPN. The lower 4 bits of the VPN part of the address are used to index the four TLB sets while the upper 15 bits are compared with the corresponding VPN fields in each TLB set.

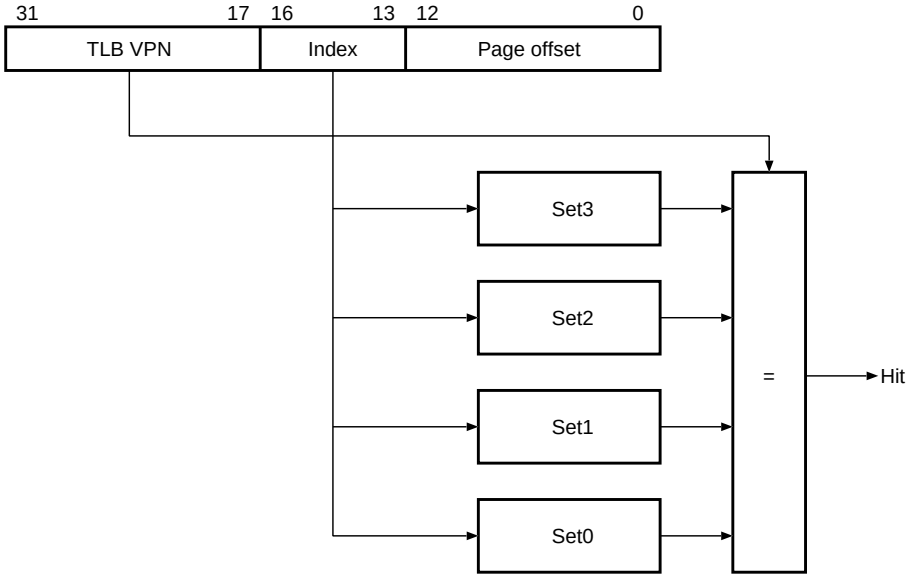


Figure 7.7: *TLB Lookup mechanism*

If a matching entry is found, the PID field is compared to the lower 8 bits of the current PID field stored in a CPU special register. If the global bit is cleared the PID fields must match for the lookup to be treated as a hit in the TLB. If the global bit is set, the matching of the PID fields is ignored and the page is treated as globally available for all possible PID settings.

Only the lower 8 bits of the PID field in the CPU special register are compared to the PID field in the TLB. Remaining bits are ignored by the MMU.

Do not store more than one translation for a page in the TLB. If a lookup results in more than one hit in the TLB, the result will be undefined behavior of the address translation in the MMU.

The valid, kernel, write enable and execute bits are checked and, if all of the conditions below are met, a valid physical address is output to the CPU.

- valid bit is set
- kernel bit is cleared or the CPU is in Kernel mode
- write enable bit is set or the CPU access type is not a write
- execute bit is set or the CPU access type is a data reference

If not all of the above conditions are met an exception is generated to the CPU and the kernel software is responsible for taking the proper action.

7.3.4.4 MMU exceptions

An exception is generated if there is a mismatch in the output from one or more fields of the TLB. In the event of an exception the kernel software is responsible for which action that is to be taken.

Five different types of MMU exceptions can occur:

- Refill - The referenced address does not match any TLB entry, or the CPU PID does not match the TLB index. A valid entry must be loaded by software.
- Write error - During a write operation, reference is made to a page that does not have the write enable bit set in the TLB entry. This exception can be used for write protection and dirty checks.
- Access violation - In User Mode, reference is made to a page with the kernel bit set in the TLB entry. When set, the kernel bit prevents CPU User Mode access to the page.
- Execute violation - An execute reference is made to a page with the execute bit cleared in the TLB entry.
- Invalid page - Reference is made to a page with matching VPN and PID fields in the TLB, but the valid bit is not set. This can be used for reference counting.

The MMU will store information about the exception in the r_mm_cause and rw_mm_tlb_sel registers. The r_mm_cause register holds the following information:

- VPN that generated the exception.
- PID of the application that generated the exception.
- Type of access that triggered the exception.

At a refill exception the rw_mm_tlb_sel register is loaded with a random value to suggest a suitable entry to be replaced. For other types of exceptions the rw_mm_tlb_sel register is loaded with a pointer to the entry that triggered the exception.

The write error, access violation, execute violation and invalid page exceptions can all be disabled in the rw_mm_cfg register. During normal operation the invalid page exception is disabled unless a reference count is in progress. When disabled, an invalid entry in the TLB will not match any address and will, thus, cause a TLB miss.

The nature of a refill exception is such that it does not allow any other exceptions to occur simultaneously. However, a single memory reference may cause combinations of write error, access violation, execute violation and invalid page exceptions at the same time. The order of which multiple exceptions are taken can be found in the CPU documentation 2.

7.4 Software interface

The MMU is configured by a number of registers available as support function registers in the CPU. These registers are read and written to by the CPU without passing through the normal memory hierarchy system.

There are two important rules that have to be followed when using the MMU registers:

- Some of the registers are used to read and update the contents of the TLB. Therefore it is important to never access a virtual memory area that is mapped through the TLB while accessing the MMU registers. This means that the software must run from an area that is not mapped by the TLB.

- The software must not access any virtual memory that is mapped by the TLB in any of the three cycles following a TLB update.

General information about support function registers can be found in the CPU documentation 2. To access the registers, fields and register constants from a C program, a set of macros and data types are defined in [MACROS] and [DEFS].

7.4.1 Support function registers

The registers rw_mm_cfg, rw_mm_kbase_lo and rw_mm_kbase_hi are used to configure the MMU at system start-up.

When an exception occurs, the cause of the exception can be read in the r_mm_cause register. This register contains the VPN, PID, and the type of operation that triggered the exception.

The TLB is controlled by registers rw_mm_tlb_sel, rw_mm_tlb_lo and rw_mm_tlb_hi. All entries in the TLB can be read and written to by the CPU through this set of registers. rw_mm_tlb_sel is used to choose which entry is to be read or written to in the TLB.

TLB entries are more than 32 bits in width, so an entry cannot be read or written to by the CPU in one cycle. When writing to the TLB the CPU must first write the high order part of the TLB entry to register rw_mm_tlb_hi. This contains the same fields as the r_mm_cause register. When writing to rw_mm_tlb_hi the data is stored at the corresponding fields in the r_mm_cause register.

The high order part of the r_mm_cause register is not stored in the TLB until the low order part is written in register rw_mm_tlb_lo. Registers r_mm_cause and rw_mm_tlb_sel are normally updated automatically by the MMU and do not require updating by the software. To write a new translation in the TLB, for example after a TLB miss, the software only has to write the translation into register rw_mm_tlb_lo.

The format of the MMU support function registers is described in 25.37.

7.4.2 Example of virtual memory configuration

Virtual memory configuration is essentially a matter of setting up the ratio of page mapped kernel space to linear mapped kernel space. The kernel/user area will always be a uniform 4Gbyte page mapped area.

For example, to set up the following virtual memory system, the MMU register configurations would resemble those described below.

- 5 x 256MByte segments of linear-mapped kernel space;
- 11 x 256MByte segments of page-mapped kernel space;

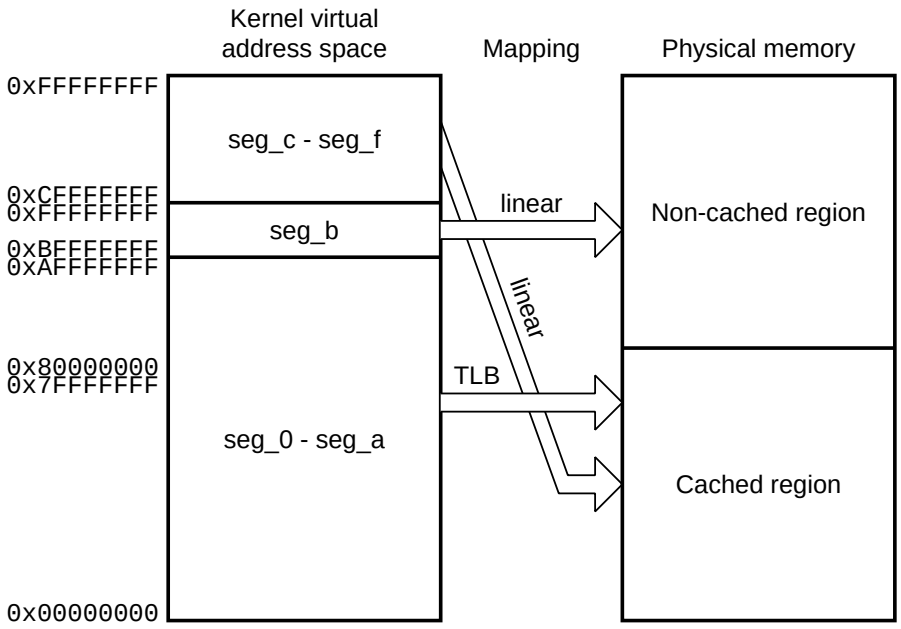


Figure 7.8: *Kernel virtual memory configuration example*

- rw_mm_cfg register settings

```
seg_f = linear
seg_e = linear
seg_d = linear
seg_c = linear
seg_b = linear
seg_a = page
seg_9 = page
seg_8 = page
seg_7 = page
seg_6 = page
```

```
seg_5 = page
seg_4 = page
seg_3 = page
seg_2 = page
seg_1 = page
seg_0 = page
```

- rw_mm_kbase_hi register settings:

```
base_f = 0x7  // linear mapped to 0x70000000
base_e = 0x6  // linear mapped to 0x60000000
base_d = 0x5  // linear mapped to 0x50000000
base_c = 0x4  // linear mapped to 0x40000000
base_b = 0xb  // linear mapped to 0xb0000000
base_a = 0x0  // don't care
base_9 = 0x0  // don't care
base_8 = 0x0  // don't care
```

- rw_mm_kbase_lo register settings:

```
base_7 = 0x0  // don't care
base_6 = 0x0  // don't care
base_5 = 0x0  // don't care
base_4 = 0x0  // don't care
base_3 = 0x0  // don't care
base_2 = 0x0  // don't care
base_1 = 0x0  // don't care
base_0 = 0x0  // don't care
```

7.5 Differences compared to the ETRAX 100LX MMU

The ETRAX FS MMU is very similar to the one used in the ETRAX 100LX chip. The main difference is that the CRIS v32 CPU uses two separate MMUs, one for the instruction cache and one for the data cache. The ETRAX FS CPU also has a more direct connection to the MMU registers by using the support function registers.

The following changes have been made that affect ETRAX FS MMU functionality:

- The PID field of the TLB has been extended from 6 bits to 8 bits.
- An execute bit has been added to the TLB to protect pages from execution.

Chapter 8

Clock generation and reset

8.1 References

Reference	Description
[BOOT]	Boot Methods, chapter 6
[REGS]	General configuration registers, chapter 25.8
[MACROS]	http://developer.axis.com

Table 8.1: *References*

8.2 Overview

The ETRAX FS uses a Phase Locked Loop (PLL) to generate an internal 400 MHz clock from an external 12 MHz clock, supplied at the **clk** input. The 400 MHz clock is divided into eight 100 MHz and two 200 MHz internal clocks, of which all except one can be individually turned on or off. It is also possible to turn off and bypass the PLL, in which case the chip will be clocked directly from the external **clk** pin.

The ETRAX FS requires an active low reset signal on the **rst_n** input. The reset takes effect asynchronously.

8.3 Functional description

8.3.1 Clock generation

8.3.1.1 Input clock

The ETRAX FS requires an external clock signal on the **clk** input. When the PLL is used, the clock signal shall have a stabilized frequency of 12 MHz. With the PLL in bypass mode, the operation is fully static and the clock can have any frequency between 0 and 100 MHz.

8.3.1.2 PLL

The ETRAX FS uses a Phase Locked Loop (PLL) to generate an internal 400 MHz clock from the 12 MHz clock supplied at the **clk** pin. The PLL requires an external 1.5 nF loop filter capacitor to be connected between the **plllpf** pin and ground.

Immediately after reset, the PLL is bypassed and turned off. Depending on selected boot method, see 6, the PLL will be started by the boot code or remain bypassed.

When the PLL is started, the internal clocks are silent for 8192 cycles of the external clock (approximately 683 μs) while the PLL is locking. The external clock must have a stabilized 12 MHz frequency during the whole lock time and as long as the PLL is enabled.

8.3.1.3 PLL bypass mode

The PLL can be turned off by writing to the pll field in the rw_clk_ctrl register. Turning off the PLL will result in that the internal clock is reduced from 400 MHz to 12 MHz, and that the 100 MHz and 200 MHz clocks are reduced to 3 MHz and 6 MHz respectively. The PLL will also be turned off to save additional power.

Entering the PLL bypass mode takes immediate action while turning on the PLL again will take 683 μs due to the PLL lock time.

All internal functionality in the ETRAX FS is available as usual when the PLL is bypassed. However, external devices that are clocked by the ETRAX FS might not operate correctly if the frequency is lowered, and the performance might not be enough to handle all types of external devices correctly.

8.3.1.4 Internal clock distribution and configuration

The internal 400 MHz clock from the PLL is divided into eight 100 MHz and two 200 MHz internal clocks that are distributed to the internal blocks. Some parts in the I/O Processor block can also use the external 12 MHz clock as an internal clock to synchronize communication with external devices.

One 100 MHz clock which supports essential system functionality is always enabled. The other internal 100 MHz and 200 MHz clocks can be individually turned on and off, see table 8.2. After reset, only the CPU clock region is enabled. Depending on which boot mode that is used, different blocks will be turned on by the boot code, see 6.

Internal clock region	Clock frequency	Clock enable
Memory arbiter, internal RAM and ROM, DMA connection logic, pin multiplexer, general I/O, timers, central interrupt logic and the clock and reset configuration registers	100 MHz	Always on
CPU, including caches, MMUs and debug support	200 MHz	Configurable
I/O Processor	200 MHz	Configurable
Ethernet port 0 and internal DMA channels 0 and 1	100 MHz	Configurable
Internal DMA channels 2 and 3	100 MHz	Configurable
Internal DMA channels 4 and 5	100 MHz	Configurable

Internal DMA channels 6 and 7	100 MHz	Configurable
The crypto accelerator and internal DMA channels 8 and 9	100 MHz	Configurable
The external bus interface, including external DMA channels and external bus arbitration	100 MHz	Configurable
Synchronous and asynchronous serial ports, ATA and Ethernet port 1	100 MHz	Configurable

Table 8.2: *Internal clocks*

8.3.1.5 Turning off clocks

The configurable 100 MHz and 200 MHz clocks can be turned off by writing to the rw_clk_ctrl register, see 25.8.

There can be no communication with a block where the clock has been turned off. The internal state of each block is kept while the clock is turned off and if the block is turned on again it will continue as if nothing has happened. The user must make sure that the block is in a safe state before turning off the clock so that no internal or external buses will be blocked.

All configurable clocks except the CPU clock are turned off at the end of the internal reset sequence, and the blocks are left in the reset state. Depending on the boot method, see 6, some clocks will remain turned off after reset while others will be turned on by the boot code.

8.3.2 Reset

8.3.2.1 Reset input

The ETRAX FS requires an active low reset signal to be supplied on the **rst_n** input. During power on, **rst_n** should be kept low until the power supply voltages have stabilized. While powered up, an active low pulse on the **rst_n** input can be used at any time to reset the ETRAX FS.

The reset takes effect asynchronously, and no clock signal is required during reset. However, if the PLL is used, the input clock should be stabilized at 12 MHz no later than two clock cycles after reset has been released.

At power up, the TAP controller (15) also needs to be reset by applying an active low reset signal to the **trst** pin. This can be done either by tying the **rst_n** and **trst** pins together or by tying the **trst** pin to Vss.

8.3.2.2 Boot mode selection

On the positive edge of the reset input, the ETRAX FS samples the boot select inputs **bs0** - **bs6** and starts initialization according to the selected boot method, see 6. The sampled boot select value is available in the r_bootsel register.

8.3.2.3 External reset output

The ETRAX FS provides an external software controlled output on the **phyrst_n** pin. This pin can be used as a delayed reset output for external circuits such as e.g. an Ethernet transceiver. The **phyrst_n** pin goes low immediately at reset. When using one of the network boot methods, the pin is set high after the 25 MHz clock on the **e0phyclk** has been started. With other boot methods, the **phyrst_n** pin remains low until it is changed by user software.

The value on **phyrst_n** can be controlled by writing to the phyrst_n field in rw_pad_ctrl.

8.3.2.4 USB transceiver suspend

The internal USB transceiver, pins **u0vp** and **u0vm**, is enabled after reset. If the transceiver is not used, it can be suspended by writing to the usb_susp field in rw_pad_ctrl.

8.4 Hardware interface

8.4.1 Clock and reset pins

The clock and reset pins are shown in the table below.

Pin name	Direction	Description
clk	input	Clock input
plllpf	-	PLL loop filter
rst_n	input	Reset input, active low
phyrst_n	output	Software controlled reset output
bs0 - bs6	inputs during reset	Boot select pins. These signals are multiplexed with the Real Time Trace functionality, where **bs0 - bs5** are used as outputs.

Table 8.3: *Clock and reset pins*

8.4.2 Clock and reset timing

See figure 8.1.

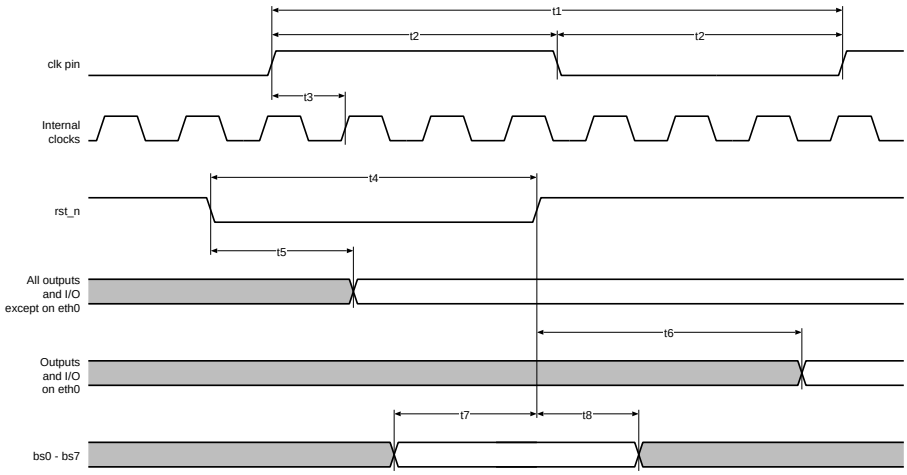


Figure 8.1: *Clock and reset timing*

Parameter number	Explanation	Min	Max	Unit
t1	Input clock cycle time , PLL enabled.	83.3	83.4	ns
t1	Input clock cycle time , PLL bypassed.	10	-	ns
t2	Input clock pulse width, PLL enabled.	33	50	ns
t2	Input clock pulse width, PLL bypassed.	4	-	ns
t3	Input clock to internal clock delay.	7	16	ns
t4	Reset pulse width.	20	-	ns
t5	Reset to output or tri-state delay, except Ethernet interface 0 pins.	-	20	ns
t6	Reset to output or tri-state delay, pins on Ethernet interface 0.	-	18*t1	ns
t7	Boot select setup time to rst_n high	1	-	ns
t8	Boot select hold time after rst_n high	2	-	ns

Table 8.4: *Clock and reset timing*

8.5 Software interface

The reset and clock configuration registers are specified in 25.8. Macros for accessing the registers from a C program are defined in [MACROS].

Chapter 9

Crypto Accelerator

9.1 References

Reference	Description
[DMA]	DMA, chapter 5
[DES]	Data Encryption Standard (FIPS46-3), http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
[DESMODES]	DES Modes of Operation (FIPS81), http://www.itl.nist.gov/fipspubs/fip81.htm
[AES]	AES Advanced Encryption Standard (FIPS197), http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
[MD5]	The MD5 Message Digest Algorithm (RFC1321), http://www.ietf.org/rfc/rfc1321.txt
[SHA1]	Secure Hash Standard (FIPS180-2), http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
[RFC1071]	RFC1071, Computing the Internet Checksum, http://www.ietf.org/rfc/rfc1071.txt
[REGS]	Strcop register description, chapter 25.42
[STRCOP_H]	Strcop C register macros, http://developer.axis.com

Table 9.1: *References*

9.2 Definitions

Term	Description
CA	Abbreviation for Cryptography (Crypto) Accelerator.
dword	A dword is 32 bits long.
word	A word is 16 bits long.
DMA out	DMA out means DMA out from memory into the CA, The terminology comes from the CPU's point of view, not the CAs.
DMA in	DMA in is DMA from the CA to the memory. The terminology comes from the CPU's point of view, not the CAs.
EOP, WAIT	The DMA data descriptor control bits eop and wait. In the text they are referenced using capitals, like EOP and WAIT.

Table 9.2: *Definitions*

9.3 Overview

The Cryptography Accelerator (CA) is connected to two DMA channels, one in each direction, and operates on data streamed through it. It does not need to be a one-to-one correspondence between incoming and outgoing bytes, but there can be only one input data stream and one output data stream concurrently. CA operations are chosen by encoding a configuration in the DMA data descriptor meta data fields, which automatically synchronize the configuration between the CA and the data it operates on.

Since almost all configuration comes from the DMA data descriptors, there is only one configuration register associated with the CA. The configuration register it does have is mainly for debugging modes and some configuration possibilities that normally do not need to be changed very often.

System overview diagram:

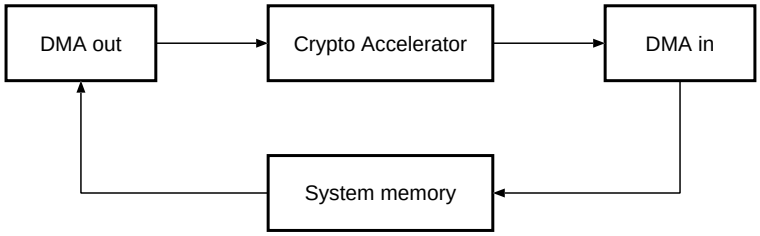


Figure 9.1: *System overview*

The CA supports three different cipher algorithms, two hash algorithms and IP checksumming. The CA is very versatile and there are numerous configuration possibilities. The CA supports running one cipher and one hash algorithm at the same time as the data is being IP checksummed. The modules implementing each algorithm can be configured to either process data in parallel or in series with the other enabled modules. For instance, it is possible to configure the CA to AES encrypt plaintext data, SHA-1 hash the ciphertext and at the same time IP checksum the plaintext.

Below is a summary of the CA's modules and some basic performance figures:

Module	Description	Performance
AES	Advanced Encryption Standard, supports ECB/CBC modes with 128/192/256-bit keys.	$>$ 200 Mbps
DES/3DES	Data Encryption Standard, supports ECB/CBC modes with 56-bit keys for DES and up to 168-bit keys for 3DES.	540/230 Mbps
SHA-1	Secure Hash Algorithm is a hashing algorithm producing 160-bit message digests.	640 Mbps
MD-5	Message Digest is a hashing algorithm producing 128-bit digests.	800 Mbps

IP checksum	IP checksum calculates the ones-complement checksum used in IP.	3.2 Gbps
Keystore	Keystore is a general register used for downloading keys to be used by the ciphers.	N/A

Table 9.3: *CA submodules*

9.4 Functional description

The CA operates mainly according to a configuration made in DMA data descriptor meta data fields. The meta data configuration determines how the central crossbar of the CA is set up. This means that the out channel data descriptors determine where data is being dropped inside the CA and the in channel descriptors determine from where inside the CA data is being fetched.

Internally, the CA consists of three parts:

1. A DMA frontend, containing DMA interface logic and configuration decoding
2. The cipher/hash modules
3. A crossbar, connecting the modules and the DMA channels according to the chosen configuration

The following diagram shows the crossbar, the modules, and the associated data paths:

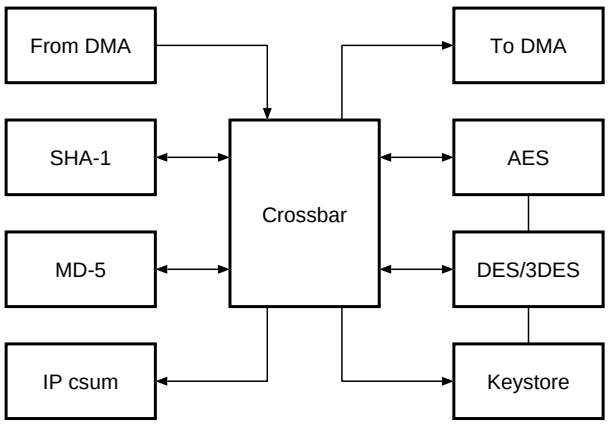


Figure 9.2: *System overview*

In addition to the data paths between the crossbar and the modules, there is a connection from the keystore supplying the cipher modules with keydata.

Since a crossbar is used, the modules can be connected either in parallel or in series. For example, the DMA input can be connected to both the AES and the SHA-1 submodules for simultaneous encryption and hashing or to the AES module followed by SHA-1 module for hashing of the encrypted data. See the examples section for more details.

As the incoming data can be routed through several modules in series, care must be taken not to switch the configuration while data is still passing through the CA. If an out channel descriptor has the WAIT bit set, the CA pauses the DMA at the end of the buffer associated with that descriptor. So, whenever a change is made to the configuration that might affect running data, the WAIT bit should be set. If in doubt, the WAIT bit can always be set even though some cycles will be wasted between descriptors if they do not contain any configuration changes. The CA resumes DMA operation when the last dword of the buffer associated with that descriptor has been processed and output.

A configuration change (involving the WAIT bit) can be carried out at almost any instance without setting EOP. However there is one case where extra care is needed. When running algorithms in series, configuration is only allowed to change between the first algorithm's blocks. As an example of this, assume that CA is configured to cipher data and then hash the cipher. In this case configuration is only allowed to change at the end of a cipher block (the cipher is the first algorithm in the chain), meaning that data sizes must be a multiple of the first algorithm's blocksize.

Some of the modules, specifically the checksum and hash modules, need to know when to stop processing the input and to start producing an output. This is specified using the EOP bit in the incoming DMA descriptors. After receiving the last dword of a buffer associated with a descriptor with that bit set, the modules stop accepting input data until the result is read out.

Notice that an EOP does not imply a WAIT automatically. For example, if many blocks are to be digested separately, they can be put sequentially in a DMA chain with EOP in each descriptor. Moreover, since there are no configuration changes, no WAIT is necessary. The other modules ignore any EOP flag, but still pass it through to succeeding modules.

The EOP bit in the DMA in channel data descriptors should not be set by software. Instead the EOP bit will be set by the CA when it has received an EOP bit from the DMA out channel and has finished processing its current packet. The EOP will force the DMA in channel to load new meta data as it moves on to the next descriptor.

9.4.1 Byte order, memory layout and block sizes

In general, all the modules expect the same byte order in memory as is given in the algorithm specifications and handle any big/little endian conversions due to DMA fetches automatically.

For example, if a key is specified in the standard as:

```
128-bit key:   95 0C DE B5 6F B2 DA CD 0A 5F DD 6E 0B B0 C2 24
```

then it should simply be placed in memory at consecutive byte addresses as:

```
unsigned char aeskey128[16] = {
   0x95,   0x0C,   0xDE,   0xB5,   0x6F,   0xB2,   0xDA,   0xCD,
   0x0A,   0x5F,   0xDD,   0x6E,   0x0B,   0xB0,   0xC2,   0x24
};
```

Any deviations from this example are specified under the separate usage instructions below.

The input data sizes for all modules, except for checksum, must be a multiple of an underlying block size, for example 16 bytes or 64 bytes. The data may be divided into any number of memory buffers as long as the first data descriptor points at a buffer containing at least 4 bytes of data. The input data sizes are described in detail in the sections for the individual modules. The modules will not work as expected if an input packet does not contain the correct number of bytes.

One notable exception is when running DMA loopback (i.e. memory to memory DMA). In this case, any block size down to individual bytes is accepted provided that no module with any other requirement is also connected to the CA DMA input and that the first memory buffer contains at least 4 bytes.

9.4.2 ECB/CBC modes and IV's

The ciphers can be run in ECB or CBC mode. The ECB mode stands for Electronic CodeBook mode and implies that an input block encrypts to the same output block given the same key under all circumstances (i.e. like a big old codebook). An attacker could conceivably build a dictionary of useful plaintext ciphertext pairs when eavesdropping on an ECB encrypted conversation.

To improve security, there is the Chained Block Cipher mode (CBC). When encrypting more than one block with this mode, the previous resulting cipher block is XORed with the next input block causing a dependency on previous input which removes the possibility of building a dictionary since a single input does not cause the same output every time. Similarly, when decrypting, the previous input ciphertext is XORed with the output plaintext of the next block.

Because the CBC mode needs a previous block to XOR with, it is necessary to specify what the very first block should use. This is called the IV, standing for Initial Vector, and when running a CA cipher in CBC mode, the first block in a packet is the IV, in both encryption and decryption modes.

When using the CBC mode on a cipher along with other parallel stream operations like a hash or a checksum, it is necessary to download the IV using a separate descriptor with a configuration set to only load it into the cipher module. This means that unless the IV itself should be hashed, the descriptor should not have the EOP bit set and no hash or checksum units enabled. This way the IV will be part of the incoming datastream and stored inside the cipher unit but it will not be processed in any way. After the IV has been stored, the next descriptor will be loaded and can enable one of the hash units and the IP checksum unit if desired. An example of such a configuration can be seen in section 9.6.7.

A typical software example of how the meta_out field of the descriptor containing the IV can be adjusted to only download the IV is shown below. The configuration for the actual computation is already set up in meta_out, but is stripped to download the IV for the initial descriptor into the cipher only:

```
struct strcop_meta_out mout  = meta_out;
```

```
mout.hashsrc                = src_none;
mout.csumsrc                = src_none;
```

9.4.3 DES/3DES specific usage

DES works on 64-bit blocks, meaning that the data packets must be a multiple of 64 bits (8 bytes) in length. The key length is always 56 bits for DES and 168 bits for 3DES (one 56-bit key for each 3DES round), and both ECB and CBC modes are supported.

However, because DES keys are normally specified with a parity bit for every seven key bits, the 56-bit keys are really 64-bit keys with embedded parity bits. Likewise, the three 56-bit keys (168 bits) for 3DES are really three 64-bit keys (192 bits). It is the parity-embedded versions that should be downloaded into the keystore even though the CA does not perform any consistency checks on the key using the parity bits.

There is no difference in speed between encryption and decryption, and the key is the same in both cases. However 3DES is, because it consists of three sequential DES operations, three times slower than DES.

When using 3DES, the software can choose the pattern of encryption/decryption for the three DES operations using the rw_cfg register. However, in practice only the default E.D.E. configuration is needed.

As a result, it is not specifiable inside the meta data configuration because it will, in most cases, stay as the default. When decryption is specified in the meta data configuration, the operations are the inverse (for example the default decryption sequence will be D.E.D.).

9.4.4 AES specific usage

AES works on 128-bit blocks, meaning that the data packets must be a multiple of 128 bits (16 bytes) in length. The key lengths are 128, 192 or 256 bits, so the keystore needs to be loaded with the appropriate key before AES processing. The speed differences between the key lengths are 20 % slower for 192 bits and 40 % slower for 256 bits compared to the 128-bit length. Both ECB and CBC modes are supported.

For decryption, the downloaded key needs to be preprocessed to match the last round-key because the key generation runs backwards compared to the encryption case. This needs to be done in software before downloading the key. Please refer to [AES] for a pseudo-code example of how this is done. This means that decryption when using completely new keys incurs a slight penalty compared to the similar encryption case, but for the more common case with session keys, the preprocessing only needs to be done once per session.

9.4.5 SHA-1 specific usage

The SHA-1 module calculates the message digest on the incoming data but needs to be fed data in multiples of 16 dwords (64 bytes). If the data to be hashed is not an integral multiple of this block size, the software needs to apply any necessary padding before sending it to the CA. Usually this can be done easily by attaching an extra DMA data

descriptor after the real data descriptor. Please refer to [SHA1] for an example of how this is done.

The message digest calculated by the SHA-1 module is 5 dwords (20 bytes) long and can be read after an EOP arrives at the input. Until it is read, the module will not accept any more data from the input.

Also, see the section on Hash IV's, 9.4.7.

9.4.6 MD-5 specific usage

The MD-5 module calculates the message digest on the incoming data but needs to be fed data in multiples of 16 dwords (64 bytes). If the data to be hashed is not an integral multiple of this block size, the software needs to apply any necessary padding before sending it to the CA. Usually this can be done easily by attaching an extra DMA data descriptor after the real data descriptor. Please refer to [MD5] for an example of how this is done.

The message digest calculated by the MD5 module is 4 dwords (16 bytes) long and can be read after an EOP arrives at the input. Until it is read, the module will not accept any more data from the input.

Also, see the section on Hash IV's, 9.4.7.

9.4.7 Hash IV's

The hashes can also run in an IV mode, where the internal state of the message digest algorithm is downloaded as the first dwords in the block. This is needed when a partial hash should be continued. The first part is hashed and the partial hash read out as normal. Then when the next part is to be hashed, the partial hash value is downloaded using a separate descriptor in front of the data body similar to the way CBC IV's are handled in the ciphers.

However, if there is no intervening hash calculation between the parts to be hashed, it is not necessary to explicitly upload and download the hash IV since the modules keep their internal state while other modules are configured to run.

9.4.8 IP-checksum specific usage

The IP checksum is calculated on the incoming data, and the result consists of a 16-bit word that has to be read out after an incoming EOP, and before any new data is accepted. The size of the incoming data must be a multiple of 2 bytes.

The endianness of the resulting word can be configured in the configuration register (that is, not by the meta data configuration) in order to give more flexibility for the software application. Sometimes the result might be used by the CPU, in which case the endianness should stay default (host endian). However, sometimes the result might be written directly into a network header, in which case it needs to have network endianness.

9.5 Software interface

In the following sections a detailed description of how to configure and use the CA is given.

9.5.1 DMA descriptor controlled configuration

Before any operation is initiated, software needs to setup the CA's DMA channels to use 32 bit wide transfers.

The bit fields and constants referred to in the following two sub-sections are available as structures and enumerations in the [STRCOP_H] C include file. Please refer to section 9.6 for further information.

Both the DMA out and the DMA in descriptors use source fields to specify where to read and write data inside the CA. The source definitions are the same for both channels and are specified in the table below:

Nbr	Description	SW alias
0	No source	src_none
1	DMA out (from memory)	src_dma
2	DES/3DES	src_des
3	SHA-1	src_sha1
4	Checksum	src_csum
5	AES	src_aes
6	MD-5	src_md5
7	Reserved (do not use)	src_res

Table 9.4: *DMA channel meta data field sources*

9.5.1.1 DMA out channel meta data

The following table describes the CA usage of the DMA out meta data field (16 bits total):

Bit	Name	Description
15	cbcmode	Choose ECB (0) or CBC (1) mode for ciphers
14	dlkey	When 1, the destination will be the keystore
13	decrypt	Choose encryption (0) or decryption (1) for ciphers
12	hashmode	Select explicit IV mode (1) or not (0) for hashes. During key download, bits 13 and 12 together define the keysize as 00=64 bits, 01=128, 10=192, 11=256.
11	hashconf	Select hash: 0=SHA-1, 1=MD-5
10-8	hashsrc	Select source of hash (see table above)
7-6	ciphconf	Select cipher: 0=DES, 1=3DES, 2=AES
5-3	ciphsrc	Select source of cipher (see table above)
2-0	csumsrc	Select source of checksum (see table above)

Table 9.5: *DMA out channel meta data field*

The three source fields ciphsrc, hashsrc and csumsrc defines how the internal crossbar is configured, streaming data from one sub-module to another. Possible values for these fields are defined in table 9.4. The other fields should be self explanatory, please refer to section 9.6 for examples of how to configure the out channel descriptor meta data field.

9.5.1.2 DMA in channel meta data

The following figure describes the CA usage of the DMA in channel meta data field:

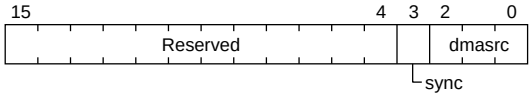


Figure 9.3: *CA usage of the DMA meta data field*

The dmasrc field selects the source of the DMA in channel as specified in table 9.4. Memory-to-memory DMA is possible by simply choosing the DMA out as the source for the DMA in.

The sync field is used to synchronize data transfers at the out channel with the in channel. When the sync field is set, it indicates that after the current in channel descriptor a change of DMA in channel configuration might take place. The bit should be set whenever it is necessary to disable the CA from writing data to the DMA in channel. Examples might be when the in channel source is changed or between two packets. Failing to configure this bit can result in data produced by the out channel ending up in the wrong in channel data buffer. If synchronization is not needed, the CA can be configured to ignore the synchronization bit. By setting the ignore_sync bit in the rw_cfg register the CA will not try to synchronize the in channel with the out channel. For example, if processing many packets after each other with the same algorithm, you do not need to synchronize the data transfers. In this case the performance can be optimized by setting the ignore_sync bit.

9.5.2 Register controlled configuration

The CA only has one configuration register, the rw_cfg register. The register contains 6 fields. Some examples of configuration that is used with the register are, enabling the CA, configuring each round in the 3DES algorithm to decrypt or encrypt data and endianness used by the IP checksum. For a full description of the configuration register, please refer to 25.42.

Briefly, this is a typical code example for initial configuration of the CA:

```
 // Configure the CA to use 3DES EDE mode and // use
little-endian result for the IP checksum.

   reg_strcop_rw_cfg rw_cfg;

   rw_cfg.en = 1;
```

```
rw_cfg.ignore_sync = 0;
rw_cfg.ipend = regk_strcop_little;
rw_cfg.td1 = regk_strcop_e;
rw_cfg.td2 = regk_strcop_d;
rw_cfg.td3 = regk_strcop_e;

REG_WR( strcop, regi_sl0, rw_cfg, rw_cfg);
```

9.5.3 Downloading keys into the keystore

The keystore is a 256-bit wide shift register. If the DMA out channel loads a data descriptor with the Download Key (dlkey) field set, all data pointed at by that descriptor will be stored in the shift register and used as key later on. The cipher modules that require less than 256 bits use the last bits downloaded, so if a 128-bit key is needed, 128 bits should be downloaded and no software padding is required.

Also, when the dlkey bit is set in a DMA data descriptor, the other configuration bits are still active but are normally not used. What the DMA writes to the CA goes to the keystore, and to all other modules that have the DMA as source. The common usage is to set all modules sources to 0 (No Source) while downloading a key. The size of the downloaded key has to be specified in the configuration bits as well, and must be either 64, 128, 192 or 256 bits.

The keystore keeps the key until it is overwritten with another key, so the software can switch CA configurations and return to a cipher without downloading the key again.

The descriptor used for downloading the key must have the EOP bit set.

It is not possible to upload a key from the keystore.

The example below shows a typical meta data configuration when downloading a key of any supported size, including the multiplexing of the keysize bits onto the decrypt and hashmode bits in the bitfield structure.

```
int keysize = (key length_in_bits / 64) - 1;

struct strcop_meta_out mout = { dlkey : 1,
                                decrypt: (keysize >> 1) & 1,
                                hashmode: keysize & 1 };
```

9.6 Configuration examples

Below, some typical configurations are shown along with an overview of how the data will flow. In all examples, the buffers associated with the DMA bulk data descriptors for input and output can be split into any number of smaller buffers. This will of course require that a new descriptor is created for each buffer and that the correct configuration is kept in all of them. The last descriptor must have the appropriate EOP or WAIT bits set.

Downloading the key is only shown once in a separate example, and it is assumed in

the other examples that the appropriate key is already loaded into the keystore.

9.6.1 Data descriptor definitions

Below are two tables with abbreviations used by the descriptor lists described in the coming sections:

Name	Description
len	Size of buffer used by descriptor. Not part of a real descriptor, only used in the examples below
ctrl	Control field. Can be set to any combination of EOP and WAIT: E = EOP, W = WAIT.
cssrc	Checksum data source. Configures where to fetch data for the IP checksum unit
cisrc	Cipher data source. Configures where to fetch data for the cipher unit currently in use.
cicfg	Cipher configuration. Chooses which cipher algorithm to use.
hssrc	Hash data source. Configures where to fetch data for the hash unit currently in use.
hscfg	Cipher configuration. Chooses which hash algorithm to use.
hsmode	Hash mode. Which initial digest to use: IV = (Initial Vector) = default digest, !IV = user defined
dec	Decrypt. Configures the CA to decrypt or encrypt.
dlkey	Download key to keystore.
cbc	Selects if a cipher algorithm shall be run in CBC or ECB mode.

Table 9.6: *out channel descriptor field abbreviations*

Name	Description
len	Size of buffer used by descriptor.
ctrl	Control field. Can be set to EOP.
dmasrc	DMA data source. Configures which source to use with the DMA in channel.
sync	Change source. When set, this field marks that the next descriptor will change the in channel source.

Table 9.7: *in channel descriptor field abbreviations*

When running chained operations such as AES encryption with SHA-1 hashing of the ciphertext, the descriptors have been named in accordance with table 9.8

Name	Full name
P	Plaintext
C	Cipher
D	Hash digest
IV	Initial vector
CP	Cipher padding
HP	Hash padding
BULK	Bulk data

Table 9.8: *Descriptor name abbreviations*

9.6.2 Downloading a key

The following figure shows downloading a 256-bit key into the keystore. The same procedure can be used for any other key length as well, with different lengths in the descriptor being the only difference.

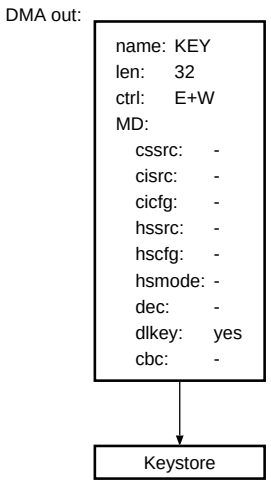


Figure 9.4: *Key download*

9.6.3 DES CBC encryption

The figure below shows the encryption of an 8000-byte block using ordinary DES in CBC mode with a user defined initial vector. A 64-bit key is used from the keystore.

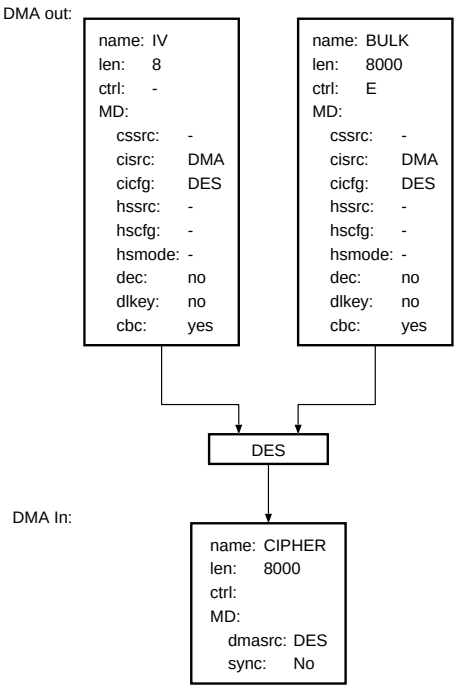


Figure 9.5: *DES CBC encryption*

9.6.4 SHA-1 hashing

The figure below shows calculating the 20-byte message digest using SHA-1 for a 128 byte block (pre-padded).

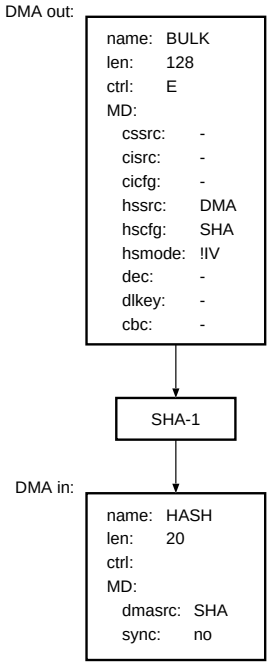


Figure 9.6: *SHA-1 hashing*

9.6.5 AES-192 ECB encryption with SHA-1 hashing of the ciphertext

The figure below shows the encryption of 1552 bytes of plain text using AES in ECB mode with a 192-bit key in the keystore, and with SHA-1 hashing of the resulting ciphertext. In this example, the bulk of the data consists of 1536 bytes that are processed by both the AES and SHA units. Next, the last 16 bytes of actual data are processed in the same way. To disconnect the AES unit, a zero-length descriptor must be inserted. Finally, 48 bytes of hash padding is added in order to end the SHA processing at a SHA-block boundary.

As can be seen in the figure below, the BULK and CP descriptors have exactly the same configuration and could have been merged together into one descriptor. However, in this example they are separated to show that the last SHA block consists of 16 bytes of actual data and 48 bytes of hash padding.

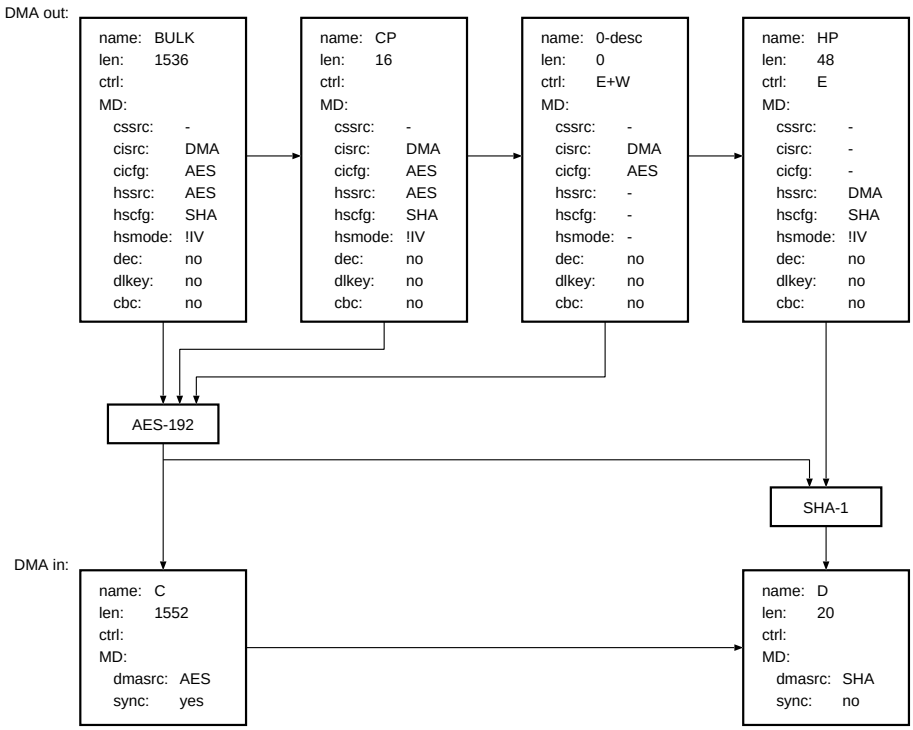


Figure 9.7: *AES-192 ECB encryption*

9.6.6 AES-192 CBC decryption with SHA-1 hashing of the ciphertext

The figure below shows the decryption of a 9008-byte ciphertext using AES in CBC mode with a 192-bit key in the keystore, and with SHA-1 hashing of the incoming ciphertext.

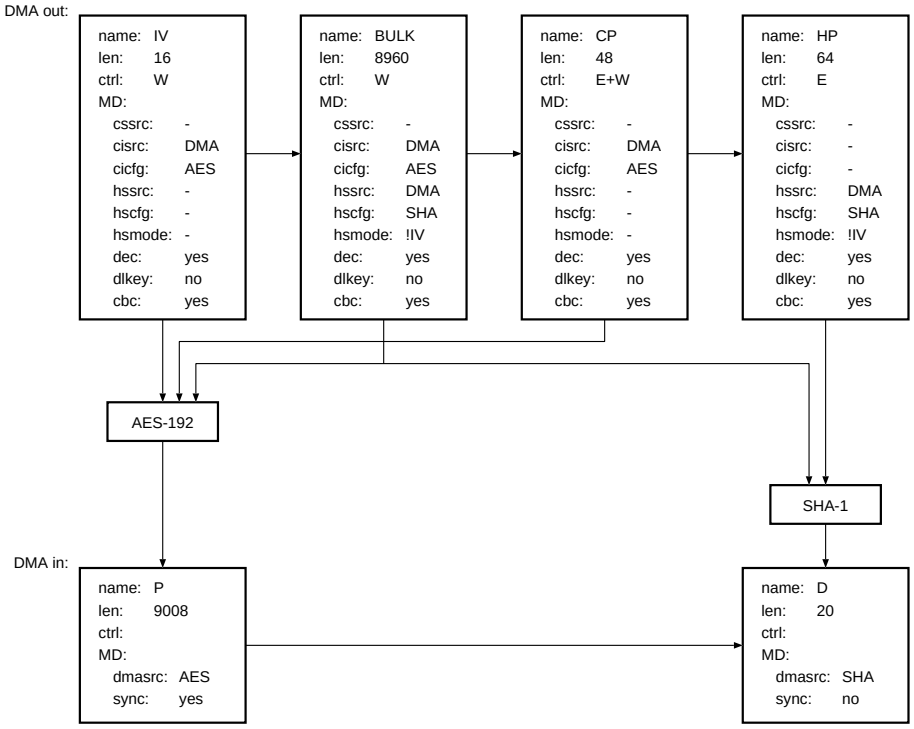


Figure 9.8: *AES-192 CBC decryption*

9.6.7 AES-256 CBC encryption with MD-5 hashing of the plaintext

The CBC mode requires an IV to be downloaded first. The IV should not reach the hash module so it requires a separate descriptor. This descriptor should not have the EOP set. The reason for having a CP descriptor that has a zero length buffer associated with it, is to disconnect the hash unit when the EOP bit is reached. If the EOP bit had been set in the BULK descriptor instead, the hash calculation would end before processing the hash padding.

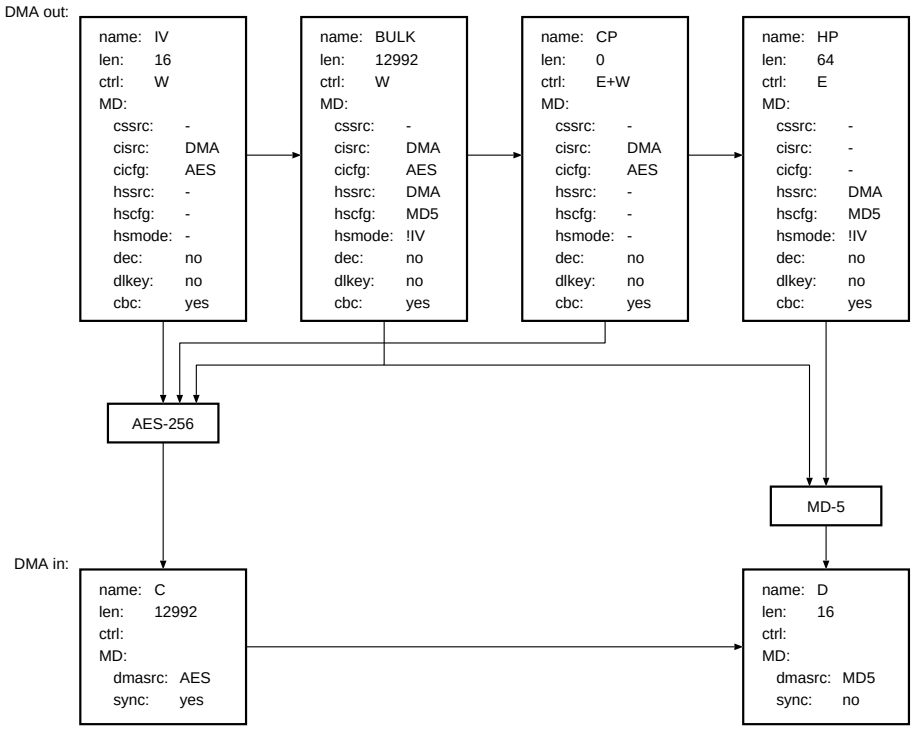


Figure 9.9: *AES-256 CBC encryption*

9.6.8 Memory-to-memory copying with parallel IP checksumming

This example shows coping a block of memory while checksumming it using the IP checksum module.

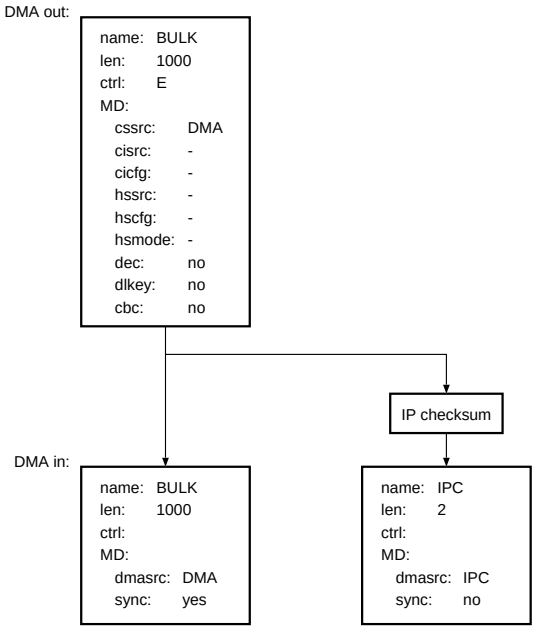


Figure 9.10: *Memory-to-memory copying*

9.6.9 3DES ECB decryption in DED mode

This figure shows 3DES in ECB mode, decrypting 1000 bytes using a Decryption/Encryption/Decryption sequence.

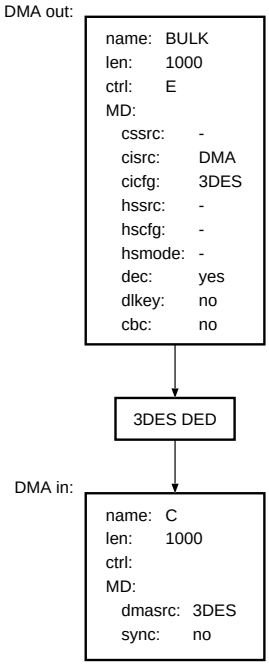


Figure 9.11: *3DES ECB decryption*

9.7 Performance Issues

Provided that no DMA or memory bottlenecks exist, the following table lists the maximum throughputs achieved when the CA modules operate by themselves (without any parallel invocations).

Module	Throughput	Longwords per chunk
DES	64 MByte/s	2
3DES	27 MByte/s	2
AES	32 MByte/s	4
SHA-1	76 MByte/s	16
MD-5	96 MByte/s	16
IP csum	384 MByte/s	1

Table 9.9: *Maximum throughputs*

When using parallel invocation, see for example section 9.6.7 above where both AES and MD-5 are run in parallel, the modules have to wait for each other to complete each chunk before the DMA can push more data to them. A similar situation arises when modules are run in serial, like in section 9.6.5. So in compound situations, the throughput is mostly dependant on the slowest module involved.

In practice, the memory will probably be a bottleneck depending on the memory configuration used, at least for the faster modules like the IP checksumming and hashes. Therefore, it is a good idea to make use of the internal RAM in the chip as a temporary buffer store. For example, if an Ethernet packet is to be checksummed and sent with the source data residing in the external RAM, it should be copied to the internal RAM by the CA at the same time as it is checksummed. It can then be sent by the Ethernet DMA from the internal RAM. This avoids dual accesses to the external RAM and the extra internal RAM accesses are almost for free in comparison.

Chapter 10

DMA Connection

10.1 References

[REGS]	Registers, chapter 25.43
[MACROS]	http://developer.axis.com

Table 10.1: *References*

10.2 Functional description

The internal DMA is connected to the different I/O interface modules in the ETRAX FS through the DMA connection module.

Before a DMA channel is used together with an I/O interface, the connection must be set up in the rw_cfg register, see 25.43. All connections through the DMA connection module are disabled after a system reset. The following configuration alternatives are available:

DMA channel	Direction	I/O interface alternatives
dma0	out	Ethernet interface 0 (eth0)
dma1	in	Ethernet interface 0 (eth0)
dma2	out	External DMA channel 2 (ext2), I/O processor DMC 0 (iop0), ATA interface (ata), Asynchronous serial port 2 (ser2)
dma3	in	External DMA channel 3 (ext3), I/O processor DMC 0 (iop0), ATA interface (ata), Asynchronous serial port 2 (ser2)
dma4	out	I/O processor DMC 1 (iop1), Asynchronous serial port 1 (ser1), Synchronous serial port 0 (sser0)
dma5	in	I/O processor DMC 1 (iop1), Asynchronous serial port 1 (ser1), Synchronous serial port 0 (sser0)

dma6	out	External DMA channel 0 (ext0), Asynchronous serial port 0 (ser0), Synchronous serial port 1 (sser1), Ethernet interface 1 (eth1)
dma7	in	External DMA channel 1 (ext1), Asynchronous serial port 0 (ser0), Synchronous serial port 1 (sser1), Ethernet interface 1 (eth1)
dma8	out	External DMA channel 2 (ext2), Crypto accelerator (strcop), Asynchronous serial port 3 (ser3)
dma9	in	External DMA channel 3 (ext3), Crypto accelerator (strcop), Asynchronous serial port 3 (ser3)

Table 10.2: *DMA channel to I/O interface connection alternatives*

10.3 Software interface

The configuration register for the DMA connection module is specified in 25.43. A set of macros for accessing the registers from a C program is available in [MACROS].

Chapter 11

Internal Memory

11.1 References

Reference	Description
[BOOT]	Boot methods, chapter 6
[DEBUG]	Stubless debugging, chapter 17

11.2 Definitions

11.3 Functional Description

11.3.1 General

The internal memory consists of 128 kbytes of RAM and 8 kbytes of ROM. The memory is 256 bits wide and has a cycle time of 20 ns, giving a memory bandwidth of 1.6 Gbytes/s.

Internal memory is accessible from the CPU/cache system, the DMA, the I/O processor and through the slave mode part of the bus interface.

The total address range assigned to internal memory in the ETRAX FS is 0x38000000-0x3fffffff. The memory can also be used with non-cached addresses in the range 0xb8000000-0xbfffffff.

The RAM address range is 0x38000000-0x3801ffff, but the RAM is also accessible at each 128 k increment of the address, up to 0x3bffffff. The ROM range is 0x3c000000-0x3c001fff but it is also accessible at each 8 k address increment up to 0x3fffffff.

The contents of the internal RAM after reset is undefined.

11.3.2 ROM content

The ROM contains code for the different boot methods, see 6, and for the ”guru mode” on chip debug, see 17.

The ROM also contains a vendor ID code at address 0x3c000000.

Chapter 12

Interrupts

12.1 References

Reference	Description
[REGS]	Interrupt controller registers, chapter 25.14

Table 12.1: *References*

12.2 Overview

Interrupts are generated by the different on-chip subsystems, or through external interrupt request pins. There are three types of interrupts:

- Maskable vectorized interrupts
- Non maskable interrupts (NMI)
- Guru mode exceptions

The vectors for the maskable interrupts are all generated internally on-chip by the interrupt controller. Guru mode exceptions have the highest priority, followed by NMIs. Vectorized interrupts have the lowest priority. There is no internal priority between different vectorized interrupts. They all share the same priority. For more information of the different interrupt types see 2.1.10.

12.3 Functional Description

12.3.1 Interrupt masks

Maskable interrupts generated by internal subsystems are masked in two levels. The first level is located in each subsystem that generates interrupts. This mask has individual mask bits for each interrupt source within the subsystem. After the mask, the

individual interrupt bits of a subsystem are combined (logically OR-ed) to form a single interrupt vector request.

The second mask level is a vector mask with one mask bit for each interrupt vector. There is generally one vector for each subsystem. The vector mask is located in the interrupt controller.

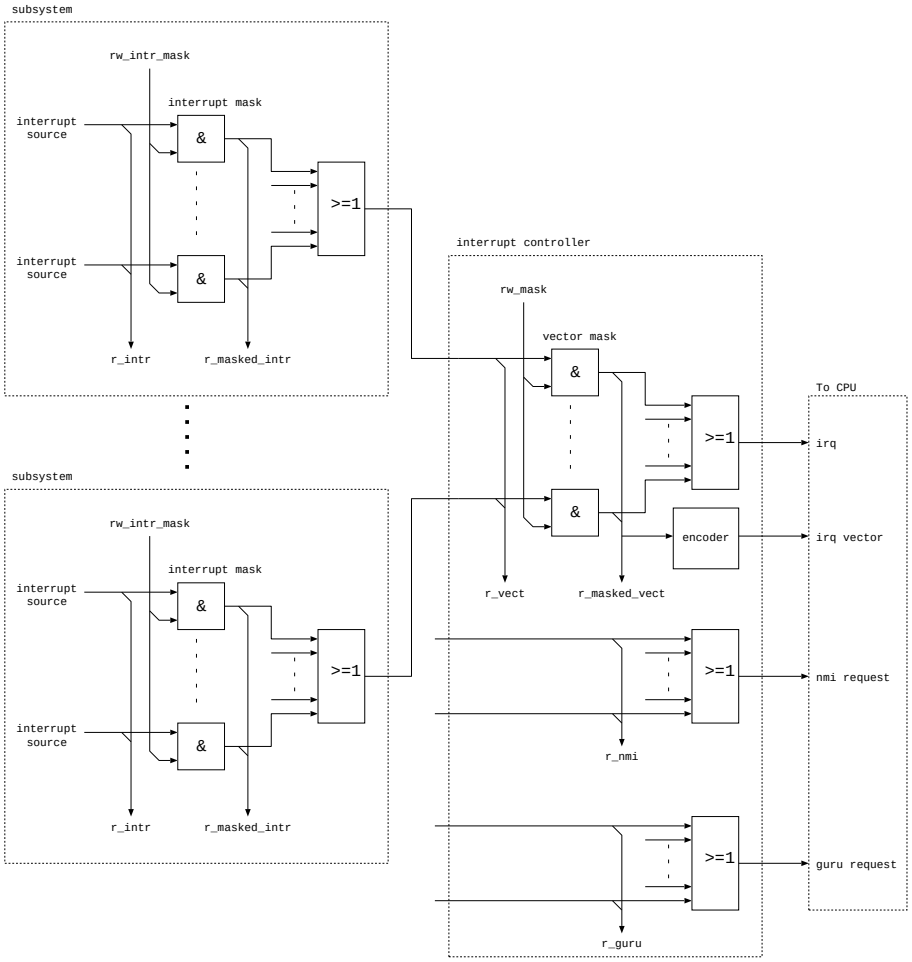


Figure 12.1: *Interrupt masks*

It is recommended that the software driver for each subsystem should control its own mask bits at the subsystem interrupt masking level, while general system functions control the vector mask. For this reason, there is one mask bit on each level, even for those vectors that have only one interrupt.

The interrupt on the **irq_n** pin does not have any first level mask, since external peripherals typically have their own interrupt masking capabilities.

12.3.2 Interrupt status

12.3.2.1 Individual interrupts

The status of individual interrupts can be read in registers inside each subsystem, both before and after they have been gated with the interrupt mask. In each subsystem, register r_intr contains the interrupt bits prior to masking and register r_masked_intr contains the interrupt bits after masking. Register rw_intr_mask contains the interrupt mask bits. In general it holds that (using C syntax):

```
r_masked_intr = r_intr & rw_intr_mask
```

12.3.2.2 Interrupt vectors

The status of each interrupt vector can be read in registers inside the interrupt controller (see 25.14), both before and after they have been gated with the vector mask. Register r_vect contains the interrupt vector bits prior to masking and register r_masked_vect contains the interrupt vector bits after masking. Register rw_mask contains the interrupt vector mask bits. In general it holds that (using C syntax):

```
r_masked_vect = r_vect & r_mask
```

12.3.2.3 Non maskable interrupts

The cause of a non maskable interrupt (NMI) may be read in the register r_nmi in the interrupt controller. The register contains one bit for each NMI source.

12.3.2.4 Guru mode exceptions

The cause of a guru mode exception may be read in the register r_guru in the interrupt controller. The register contains one bit for each guru mode exception source.

12.3.3 Vector generation

The interrupt controller generates an interrupt request and a vector number to the CPU based on the masked interrupt vector bits. If only one bit is set, a vector corresponding to that bit is generated. If two or more bits are set, a special multiple vectors interrupt vector is generated. It is then up to the interrupt handling software to decide in what order to serve the active interrupts.

12.3.4 Interrupt vector numbers

The following interrupt vector numbers are defined:

Name	Vector Nr	Description
mult	0x30	multiple interrupt vectors
memarb	0x31	memory arbiter breakpoints
gen_io	0x32	general I/O
iop0	0x33	I/O processor port 0
iop1	0x34	I/O processor port 1
iop2	0x35	I/O processor port 2
iop3	0x36	I/O processor port 3
dma0	0x37	dma channel 0
dma1	0x38	dma channel 1
dma2	0x39	dma channel 2
dma3	0x3a	dma channel 3
dma4	0x3b	dma channel 4
dma5	0x3c	dma channel 5
dma6	0x3d	dma channel 6
dma7	0x3e	dma channel 7
dma8	0x3f	dma channel 8
dma9	0x40	dma channel 9
ata	0x41	ATA interface
sser0	0x42	Synchronous serial port 0
sser1	0x43	Synchronous serial port 1
ser0	0x44	Serial port 0
ser1	0x45	Serial port 1
ser2	0x46	Serial port 2
ser3	0x47	Serial port 3
eth0	0x49	Ethernet port 0
eth1	0x4a	Ethernet port 1
timer	0x4b	Timers
bif_arb	0x4c	Bus interface arbiter
bif_dma	0x4d	Bus interface DMA
ext	0x4e	External IRQ pin

12.3.5 Interrupt acknowledge

Once active, an interrupt stays active until acknowledged by software. Each interrupt shall be acknowledged at the source of the interrupt. E.g. a serial port interrupt is acknowledged through mode registers in the serial port causing the interrupt. In most subsystems interrupts are acknowledged by writing to the rw_ack_intr register.

12.3.6 Non maskable interrupts

A non maskable interrupt (NMI) can be generated by following sources:

Name	Description
ext	External NMI pin
watchdog	Watchdog timer

12.3.7 Guru mode exceptions

A guru mode exception can be generated by the following (non CPU internal) sources:

Name	Description
jtag	JTAG debug interface

Chapter 13

I/O Processor

13.1 References

Reference	Description
[DMA_MDS]	DMA, chapter 5
[IOPASM]	Assembler tool for MPU and SPU, http://developer.axis.com

Table 13.1: *References*

Reference	Description
[STRMUX]	DMA Connection, chapter 10
[CRC_PAR_REGS]	Mode registers, chapter 25.15
[CRC_SER_IN_REGS]	Mode registers, chapter 25.32
[CRC_SER_OUT_REGS]	Mode registers, chapter 25.33
[DMC_IN_REGS]	Mode registers, chapter 25.16
[DMC_OUT_REGS]	Mode registers, chapter 25.17
[FIFO_IN_REGS]	Mode registers, chapter 25.19
[FIFO_IN_XTRA_REGS]	Mode registers, chapter 25.21
[FIFO_OUT_REGS]	Mode registers, chapter 25.18
[FIFO_OUT_XTRA_REGS]	Mode registers, chapter 25.20
[MPU_REGS]	Mode registers, chapter 25.22
[SAP_IN_REGS]	Mode registers, chapter 25.23
[SAP_OUT_REGS]	Mode registers, chapter 25.24
[SPU_REGS]	Mode registers, chapter 25.25
[SW_CFG_REGS]	Mode registers, chapter 25.26
[SW_CPU_REGS]	Mode registers, chapter 25.27
[SW_MPU_REGS]	Mode registers, chapter 25.22
[SW_SPU_REGS]	Mode registers, chapter 25.29
[TRIGGER_REGS]	Mode registers, chapter 25.31
[TIMER_REGS]	Mode registers, chapter 25.30

Table 13.2: *References to register descriptions*

13.2 Definitions

CRC	Cyclic Redundancy Check
DMC	DMA Communicator
CPU	The main CPU of the ETRAX FS
MC	Memory Controller
MPU	Master Processing Unit
SAP	Synchronization and Asynchronous Paths
SPU	Slave Processing Unit
GIO	General I/O
HAB	Hardware Accelerator Block (FIFO, Ser CRC, Par CRC, DMC_in, DMC_out)
ALU	Arithmetic Logic Unit
FSM	Finite State Machine

Table 13.3: *Definitions*

13.3 Overview

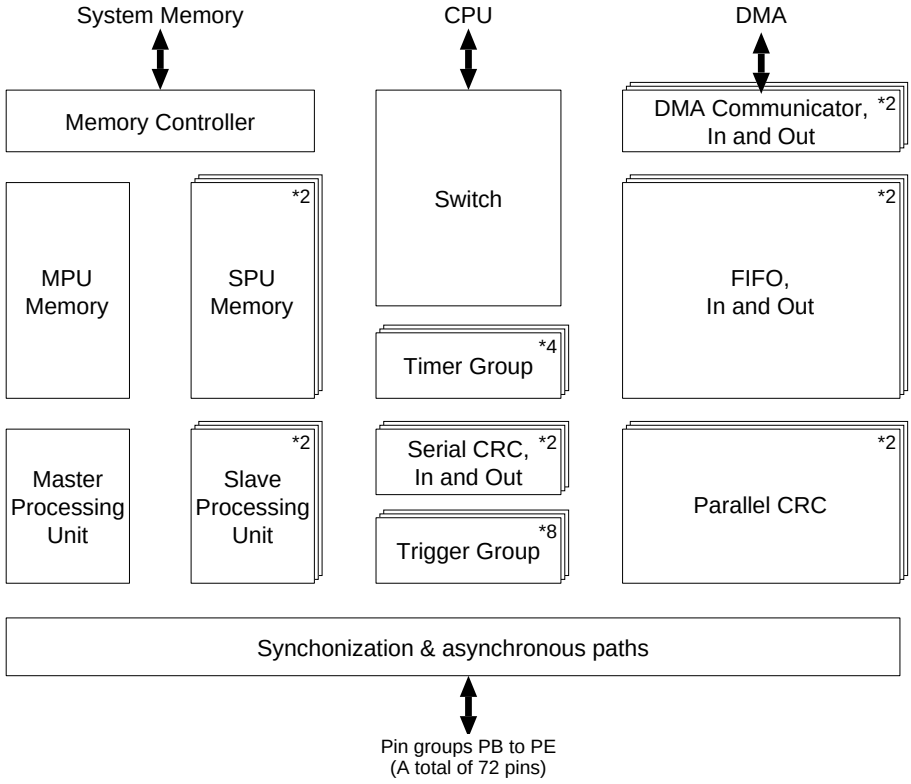


Figure 13.1: *Block diagram over the I/O Processor*

As it can be seen in figure 13.1, the I/O Processor is not a processor in a conventional sense. From a software perspective it should be considered as a collection of HABs, which are connected to each other in a chip-specific manner. However, the flow of data is configurable with regards to which hardware blocks the data should traverse on its way to or from a peripheral device (external hardware).

13.3.1 The concept of ownership

In this text ownership is defined as the ability to read and write the register interface of a HAB. The ownership hierarchy of the I/O Processor is described in table 13.4. Each row lists one of the I/O Processor Modules in the first column, followed by the modules that can own it in the second column:

Module	Possible Owners
CPU	
MPU	CPU
SPU0	CPU, MPU, SPU1
SPU1	CPU, MPU, SPU0
HABs[1]	CPU, MPU, SPU0, SPU1

Table 13.4: *Module Ownership*

The CPU can own any module, while the MPU can own any module except the CPU. SPU0 and SPU1 can own each other and any HAB.

Generally, each module can have one owner. However, in addition to a primary owner, the FIFO can also have a secondary owner. The secondary owner implements a subset of the owned FIFO's primary register interface in order to read and write data, and to read FIFO status.

13.3.2 MPU Characteristics

The MPU is a traditional processor in the sense that it has interrupt handling capabilities. The CPU is able to force the MPU to execute instructions using the register interface.

13.3.3 SPU Characteristics

The Slave Processing Unit (SPU) is a special processor. It differs from an ordinary CPU in the following ways:

- Does not have any interrupt handling capabilities
- The SPU can execute state-machine code in a special mode called FSM-mode

[1] HABs = FIFO, Ser CRC, Par CRC, MC, DMC_in, DMC_out, Timers and Triggers.

Beside these two differences, the SPU is still able to execute normal sequential code just as any other processor. The FSM-mode of the SPU can be described in the following way:

- At most eight state transitions can be specified in each state
- Each state can trigger a transition by using a combination of up to four inputs or the internal timer in the SPU
- Up to four outputs can be changed as a result of a state transition
- An optional sequential instruction can be executed in each state
- Up to four events can be specified and used to handle exceptions that are asynchronous as seen from a protocol perspective (i.e., error conditions)

13.3.4 The Memory Controller (MC)

The I/O Processor's interface towards the ETRAX FS system memory consists of the memory controller (MC). MC allows the following operations to be carried out:

- Copying data from the system memory[2] to the SPU memories
- Writing data from a register to the SPU memories
- Transferring data to and from an MC register to the system memory [2]

13.3.5 The Switch

The main responsibility of the Switch is to manage ownership and to configure the individual connections between the modules of the I/O Processor. It contains logic for:

- Register access within the I/O Processor
- Interrupt multiplexing to CPU
- Interrupt multiplexing to MPU
- Pin multiplexing, creating internal buses
- Multiplexing of signals between I/O Processor modules
- Creating data and control paths
- The CPU, MPU and SPU are all able to control whether the output of an I/O pin should be set to high or low.

[2]Due to limitations in the ETRAX FS memory arbiter the MC can not access internal mode registers.

13.3.6 SAP

The Synchronization and Asynchronous Paths (SAP) module handles the synchronization of incoming data from general I/O pins and buses, and controls the output clocking of I/O signals.

13.3.7 Trigger

The Trigger can detect the leading or trailing edge of incoming data from an I/O pin. When an edge is detected an interrupt to the MPU or CPU can be generated. The Trigger can also enable or disable other Triggers and Timers.

13.3.8 Timer

All timers are divided into groups. A timer group generates clocks and strobe signals. The strobe signals are useful when generating timeouts or enabling or disabling Timers and Triggers.

Timers and triggers can be combined in order to create more advanced functions. For further reference, see chapters 13.8 and 13.9.

13.3.9 The parallel data path

The data buses for parallel data paths are 1-4 bytes wide. The parallel data path is created by the interconnection of the following modules: Parallel CRC, FIFO, DMC In and DMC Out. These modules can also transmit or receive data through the register interface using the SPU, MPU or CPU.

Module	Description
DMA Communicator (DMC)	DMC controls the interface to DMA.
FIFO	Buffers the data.
Parallel CRC	The parallel CRC calculates CRC for either in- or out-channel. It can not calculate CRC for bot in- and out-channel at the same time.

Table 13.5: *Parallel data path modules*

Bus width	With CRC	Without CRC
8 bits	100 MByte/s	100 MByte/s
16 bits	200 MByte/s	200 MByte/s
32 bits	200 MByte/s	400 MByte/s

Table 13.6: *Maximum speed on the parallel data path*

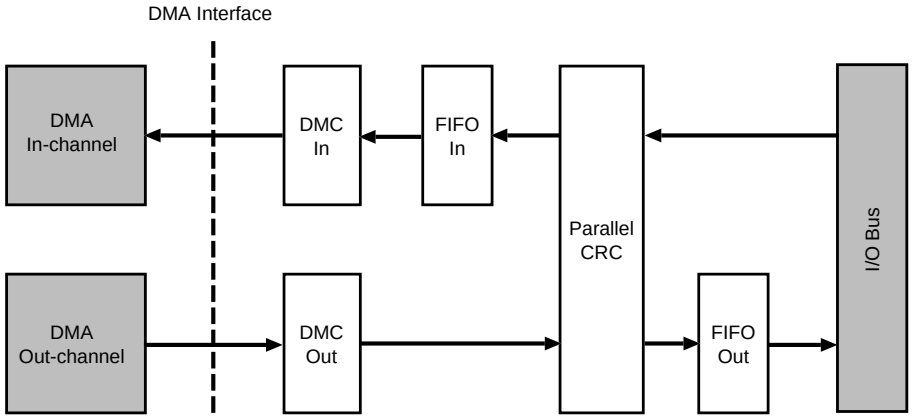


Figure 13.2: *Block diagram of the parallel data path*

13.4 Master Processing Unit

The Master Processing Unit (MPU) is a part of the I/O Processor, which can be used to control the different I/O Processor submodules. It will handle simple protocols and interrupt requests from protocol implementations with real time properties. The MPU runs at a frequency of 200 MHz and executes one instruction each clock cycle, except for the memory instructions which take an additional clock cycle to execute.

The MPU instructions are either 32- or 64-bit wide. The size of the MPU memory in ETRAX FS is 4096 bytes (512x64). The 64-bit wide memory makes it possible to read a 64-bit instruction (aligned in memory) in one clock cycle.

13.4.1 Architectural description

13.4.1.1 Registers

There are three register types in the MPU: mode registers, general registers, and special registers. The mode registers are registers found in the register banks of the I/O Processor. The general registers and some of the special registers can also be accessed by the CPU through mode registers.

The MPU contains sixteen 32-bit general registers (R0 to R15). The corresponding mode registers are the sixteen instances of rw_r. When an instruction which writes to R0-R15 is executed, the corresponding bit is set in the special register WSTS (r_wr_stat). The bit can be cleared with the following instruction (N is an integer containing the register index):

```
ANDQ WSTS, ~(1 << N), WSTS
```

The MPU architecture also defines 32 special registers (P0-P31). Of these, only 25 are implemented. All special registers are 12 bits wide, except for the WSTS register, which is 16 bits wide. The special registers and their corresponding mode registers are found in table 13.7.

Mnemonic	Reg. no.	Description	Mode register
REGA	P1	Register address storage (lowest two bits are always 0)	
PC	P2	Program counter	r_pc
WSTS	P3	Register write status	r_wr_stat
IRP	P5	Interrupt return pointer	
SRP	P6	Subroutine return pointer	
T0	P8	Memory address for thread 0	rw_thread
T1	P9	Memory address for thread 1	rw_thread
T2	P10	Memory address for thread 2	rw_thread
T3	P11	Memory address for thread 3	rw_thread
I0	P16	Memory address for interrupt request 0	rw_intr
I1	P17	Memory address for interrupt request 1	rw_intr
I2	P18	Memory address for interrupt request 2	rw_intr
I3	P19	Memory address for interrupt request 3	rw_intr
I4	P20	Memory address for interrupt request 4	rw_intr
I5	P21	Memory address for interrupt request 5	rw_intr
I6	P22	Memory address for interrupt request 6	rw_intr
I7	P23	Memory address for interrupt request 7	rw_intr
I8	P24	Memory address for interrupt request 8	rw_intr
I9	P25	Memory address for interrupt request 9	rw_intr
I10	P26	Memory address for interrupt request 10	rw_intr
I11	P27	Memory address for interrupt request 11	rw_intr
I12	P28	Memory address for interrupt request 12	rw_intr
I13	P29	Memory address for interrupt request 13	rw_intr
I14	P30	Memory address for interrupt request 14	rw_intr
I15	P31	Memory address for interrupt request 15	rw_intr

Table 13.7: *MPU, Special registers*

13.4.1.2 Data organization in memory

Instructions are stored in memory with the least significant byte at the lowest address (”little endian”). The MPU has a 64-bit wide data bus to the memory.

Instructions can be aligned to either a 32-bit or a 64-bit boundary. If a 64-bit wide instruction crosses a 64-bit boundary, the MPU will split the instruction access into two separate accesses. So, the use of unaligned 64-bit instructions will degrade performance.

- **Example:**

```
:
MOVEQ 0x1, R0
MOVEQ 0x2, R1
ADDX R0, 0x12345678, R2
ADDQ R2, R1, R3
MOVEX 0x12345678, R3
SUB R3, R2, R2
:
```

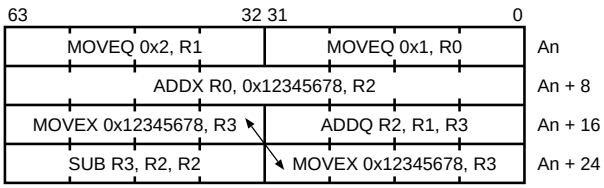


Figure 13.3: *MPU, Unaligned instruction in memory*

Note, in this example, that the MOVEX instruction is unaligned and split to address An+20 and An+24. The MPU must stall (a NOP instruction is executed) before the MOVEX instruction is executed.

13.4.1.3 Branches, jumps and subroutines

The branch and jump instructions of the MPU are delayed branch instructions. This means that the instruction following directly after a branch instruction will always be executed, even if the branch is taken.

- **Example:**

```
        :
        MOVEQ    4, R0
        NOP
LOOP:   BNZ      R0, LOOP
```

```
SUBQ     R0, 1, R0    ; Delay slot instruction, executed
                      ; even if the branch is taken
:
```

The branch to LOOP will be executed four times, and register R0 decremented by 1 after each turn. After leaving the loop, R0 will have the value -1.

There are some restrictions as to which instructions can be placed in the delay slot. Valid instructions for the delay slots are all instructions except:

- BA/BAR/BBC/BBS/BNZ/BMI/BPL/BZ
- JIR/JSR/JNT
- RET/RETI
- 64-bit instructions which are not aligned in memory.

A BBC/BBS/BMI/BNZ/BPL/BZ instruction can not use the destination register of the immediately preceding instruction as the conditional register, since the updated or old value of the register may be referenced depending on whether an interrupt occurred between the two instructions or not. When no interrupts are active, the BBC/BBS/BMI/BNZ/BPL/BZ will always reference the old register value.

- **Example:**

```
:
MOVEQ    5, R0
SUBQ     R0, 5, R0
BZ       R0, ZERO
:
```

If an interrupt routine, which does not alter the value of R0, is started after the SUBQ instruction, the value of R0 used by the BZ instruction will equal 0. Otherwise, the value of R0 will equal 5.

A NOP instruction should be added by the programmer after the SUBQ instruction. Then there will be no doubt that the value of R0 equals 0 when BZ is executed.

When executing the JSR instruction, the return address is stored in a special register called the Subroutine Return Pointer (SRP). A subroutine should end with the RET instruction and the instruction in the delay slot after RET. The RET instruction will copy the value of the SRP register to the program counter. The subroutine return pointer can be altered during the execution of the subroutine.

13.4.1.4 Interrupts

The MPU interrupts are generated by the other internal I/O Processor modules. There is a total of 16 interrupts. The I/O Processor Switch configures which interrupts that will be used. The addresses for the interrupt routines are set by writing to the mode registers (rw_intr) or by executing `MOVEQ addr, In`, where In (n = 0..15) is one of the special registers I0-I15 explained in section 13.4.1.1.

If more than one interrupt is generated at the same time, the interrupt with the highest priority will be executed. I0 has the highest priority and I15 has the lowest priority. There is no support for nested interrupts. All interrupt requests are ignored when executing an interrupt routine.

The MPU can also jump to an interrupt routine by executing the JIR instruction. The interrupt return pointer can not be put on a stack and therefore the JIR instruction should not be executed in an interrupt routine.

The interrupt routine must end with RETI and the RETI delay slot. The first instruction in an interrupt routine should not be RR, and the last instruction, in the delay slot after RETI, should not be RW/RWQ/RWX. When entering or exiting an interrupt routine and the rules above are not followed, a register read could be executed directly after a register write. Executing a register read directly after a register write is not allowed.

Interrupts request can be disabled with the DI instruction and enabled with the EI instruction.

The MPU can generate an interrupt request to the CPU by writing to a special register in the I/O Processor Switch.

13.4.1.5 The MPU executes instructions from the CPU

The CPU can make the MPU execute a single instruction by writing the instruction to the MPU's instruction register, rw_instr. The instruction will be executed immediately, except for some special cases:

- When the MPU executes LW/SW/SWX instructions. If the MPU is running a sequence containing only of LW/SW/SWX instructions, the instruction will not be executed until after the last LW/SW/SWX instruction of the sequence is executed.
- When the MPU executes an interrupt routine. If the MPU is running an interrupt routine, the instruction will not be executed until the interrupt routine is finished.
- When the MPU executes an unaligned 64-bit instruction. The instruction will not be executed until the complete 64-bit instruction has been executed.
- When the MPU executes a delay slot. The instruction will not be executed until after the instruction in the delay slot has been executed.
- After the disable interrupt (DI) instruction was executed. The instruction is not executed until after an EI instruction has been executed.

The instruction from the instruction register can always be executed when the MPU is disabled.

Before the CPU writes to the instruction register, it should check if the r_stat.instr_reg_busy field is set to no. Otherwise it might overwrite an instruction which has not yet been executed.

The CPU can make the MPU enter an interrupt routine by writing the JIR instruction to the instruction register. Executing BA will change the program counter. Using the rw_instr register is also useful when the CPU wants to access registers in Hardware Accelerator Blocks (HABs) owned by the MPU.

If the MPU code contains unaligned instructions which can be interrupted by the execution of the instruction register, then the LW/SW/SWX instruction must not be written to the instruction register.

13.4.1.6 Register write and read

The MPU can read or write mode registers in other I/O Processor modules which are owned by the MPU. A register write instruction can not be directly followed by a register read instruction.

13.4.1.7 Memory instructions

The MPU is capable of loading and storing data in its own memory. The LW instruction loads 32-bit of memory data to a register and the SW/SWX instruction stores a 32-bit value to the MPU memory.

If the LW/SW/SWX instruction can be interrupted by an interrupt request, there must be a NOP or branch instruction at the memory address preceding the LW/SW/SWX instruction. In a sequence of several LW/SW/SWX instructions which are executed after each other, the NOP instruction is needed only before the first LW/SW/SWX instruction of that sequence.

13.4.1.8 Threads

A thread is a variant of a subroutine. Threads can be interrupted by both instructions written to the instruction register and by interrupt requests. Threads are useful when splitting the execution time of non critical timing code in the MPU.

The MPU has four threads. All four thread addresses will point to address 0x0000 when the MPU is reset. Writing to special register T0-T3 will change the address of a thread. It is also possible to change the addresses by writing to the rw_thread registers.

An ordinary subroutine is started by the JSR instruction and is ended with RET. Regarding threads, a single instruction, JNT (Jump Next Thread), is used when entering and exiting a thread. JNT ends the execution of the current thread, and at the same time starts the execution of the next thread. When JNT is executed, the program counter will be updated with the contents of the special register Tn (where n = 0..3) pointed to by the thread pointer. When the next JNT instruction is executed, the thread pointer points to the next thread address (n = n + 1 modulo 4).

A thread can not be disabled. It will always end with JNT and the delay slot of JNT. The simplest thread routine will only jump to the next thread. In other words, it will only contain a JNT instruction and a NOP instruction.

- **Example:**

  All thread addresses point to WAIT at startup. The WAIT loop will be an endless loop until one of the thread addresses are changed.[3]

```
WAIT:    JNT              ; Wait for a thread address to change to
         NOP              ; THREAD0 or THREAD1.

         :

THREAD0:                  ; This thread is executed when the thread
         :                ; pointer points to it and the JNT is executed.
         JNT
         MOVEQ WAIT, T0   ; Thread address 0 is set to WAIT

         :

THREAD1:
         :
         JNT
         MOVEQ WAIT, T1   ; Thread address 1 is set to WAIT
```

Thread addresses can be changed at any time. If a thread address is changed and the thread routine is never executed before the thread address is changed a second time, the first address change will be lost and it will never take place.

- **Example:**

  This program will execute in the following order[4]:

  START->WAIT->THREAD0->WAIT->THREAD2->THREAD3->WAIT

```
START:   MOVEQ THREAD0, T0
         MOVEQ THREAD1, T1  ; THREAD1 routine is never executed because
                            ; Thread address 1 is changed in THREAD0.
         MOVEQ THREAD2, T2
         MOVEQ THREAD3, T3

WAIT:    JNT
         NOP

         :

THREAD0:
```

[3] The MOVEQ instructions are executed in the delay slot of JNT.

[4] THREAD1 is never executed. When the program has returned to WAIT it will loop until one thread address is changed.

```
        :
        MOVEQ WAIT, T1     ; Thread address 1 is set to WAIT
        :
        JNT
        MOVEQ WAIT, T0     ; Thread address 0 is set to WAIT

THREAD1:
        :
        JNT
        MOVEQ WAIT, T1     ; Thread address 1 is set to WAIT

THREAD2:
        :
        JNT
        MOVEQ WAIT, T2     ; Thread address 2 is set to WAIT

THREAD3:
        :
        JNT
        MOVEQ WAIT, T3     ; Thread address 3 is set to WAIT
```

13.4.2 Instruction set description

13.4.2.1 Definitions

addr	Instruction address. This field points to a 32 bit entity in the MPU memory. To get the byte address the field is shifted left by two steps (addr << 2).
b	An 8 bit wide immediate value.
i	Immediate value which is either 16 or 32 bits wide.
regaddr	Address to a register in one of the I/O Processor register banks. This field points to a 32 bit entity in the mode register memory space. To get the byte address the field is shifted left by two steps (regaddr << 2).
Rd, Rn and Rs	Index of a general register (R0-R15) or a special register (P0-P31).

Table 13.8: *MPU, Instruction set definitions*

13.4.2.2 Instructions in alphabetical order

13.4.2.2.1 ADD - Add

Assembler syntax:

```
ADD Rs, Rn, Rd
```

Description: The source register is added to the contents of a general register or a special register, and the result is stored in the destination register. When either D, S or N is set to 1, Rd, Rs or Rn respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs + Rn;
```

Instruction format:

```
31     25    20    15    10 9 8 7       0
+------+-----+-----+-----+-+-+-+--------+
|010000|  Rd |  Rs |  Rn |D|S|N|10001100|
+------+-----+-----+-----+-+-+-+--------+
```

D	S	N	General registers	Special registers
0	0	0	Rd Rs Rn	None
0	0	1	Rd Rs	Rn
0	1	0	Rd Rn	Rs
0	1	1	Rd	Rs Rn
1	0	0	Rs Rn	Rd
1	0	1	Rs	Rd Rn
1	1	0	Rn	Rd Rs
1	1	1	None	Rd Rs Rn

13.4.2.2.2 ADDQ - ADD Quick

Assembler syntax:

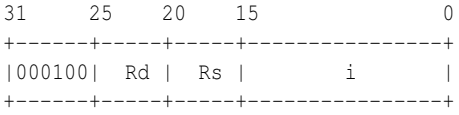
```
ADDQ Rs, i, Rd
```

Description: A 16-bit immediate value is zero extended to 32 bits and added to the contents of a general register, and the result is stored in the destination register. Both Rs and Rd are general registers.

Operation:

```
Rd = Rs + i;
```

Instruction format:



13.4.2.2.3 ADDX - ADD Extended

Assembler syntax:

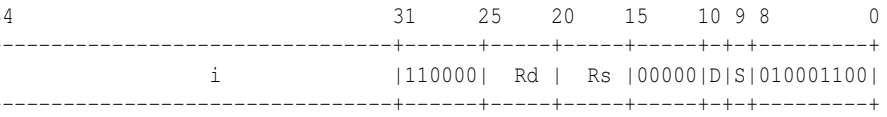
```
ADDX Rs, i, Rd
```

Description: A 32-bit unsigned immediate value is added to the contents of a general register or a special register, and the result is stored in the destination register. When either D or S is set to 1, Rd or Rs respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs + i;
```

Instruction format:



D	S	General registers	Special registers
0	0	Rd Rs	None
0	1	Rd	Rs
1	0	Rs	Rd
1	1	None	Rd Rs

13.4.2.2.4 AND - Logical AND

Assembler syntax:

```
AND Rs, Rn, Rd
```

Description: A bitwise AND is performed between the source register and the contents of a general register or a special register. The result is stored in the destination register. When either D, S or N is set to 1, Rd, Rs or Rn respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs & Rn;
```

Instruction format:

```
31     25    20    15    10 9 8 7       0
+------+-----+-----+-----+-+-+-+--------+
|010000|  Rd |  Rs |  Rn |D|S|N|10001010|
+------+-----+-----+-----+-+-+-+--------+
```

D	S	N	General registers	Special registers
0	0	0	Rd Rs Rn	None
0	0	1	Rd Rs	Rn
0	1	0	Rd Rn	Rs
0	1	1	Rd	Rs Rn
1	0	0	Rs Rn	Rd
1	0	1	Rs	Rd Rn
1	1	0	Rn	Rd Rs
1	1	1	None	Rd Rs Rn

13.4.2.2.5 ANDQ - Logical AND Quick

Assembler syntax:

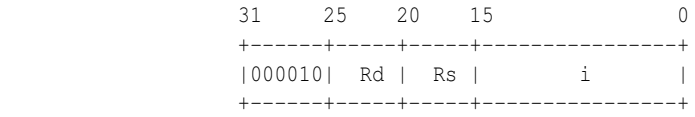
```
ANDQ Rs, i, Rd
```

Description: The 16-bit immediate value is zero extended to 32 bits and then a bitwise AND is performed between it and the contents of a general register. The result is stored in the destination register. Both Rs and Rd are general registers.

Operation:

```
Rd = Rs & i;
```

Instruction format:



13.4.2.2.6 ANDX - Logical AND Extended

Assembler syntax:

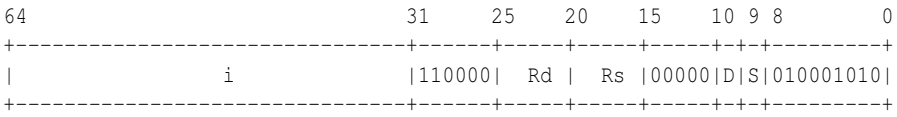
```
ANDX Rs, i, Rd
```

Description: A bitwise AND is performed between a 32-bit immediate value and the contents of a general register or a special register. The result is stored in the destination register. When either D or S is set to 1, Rd or Rs respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs & i;
```

Instruction format:



D	S	General registers	Special registers
0	0	Rd Rs	None
0	1	Rd	Rs
1	0	Rs	Rd
1	1	None	Rd Rs

13.4.2.2.7 BA - Branch Always

Assembler syntax:

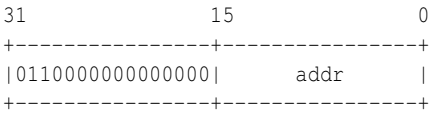
```
BA addr
```

Description: The program counter (PC) is loaded with the contents of addr. The BA instruction is a delayed branch instruction, with one delay slot.

Operation:

```
PC = (addr << 2);
```

Instruction format:



13.4.2.2.8 BAR - Branch Always Register

Assembler syntax:

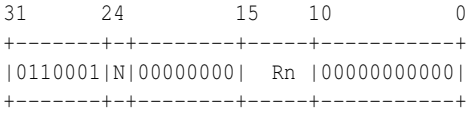
```
BAR Rn
```

Description: The program counter (PC) is loaded with the contents of the source register. The BAR instruction is a delayed branch instruction, with one delay slot. When N is set to 1, Rn becomes a special register rather than a general register.

Operation:

```
PC = Rn;
```

Instruction format:



13.4.2.2.9 BBC - Branch Bit Clear

Assembler syntax:

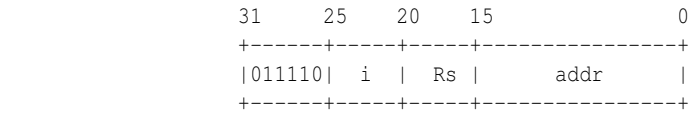
```
BBC Rs, i, addr
```

Description: The program counter (PC) is loaded with the contents of addr if bit i of the source register is cleared. The BBC instruction is a delayed branch instruction, with one delay slot.

Operation:

```
if ((Rs & (1 << i)) == 0) {
   PC = (addr << 2);
}
```

Instruction format:



13.4.2.2.10 BBS - Branch Bit Set

Assembler syntax:

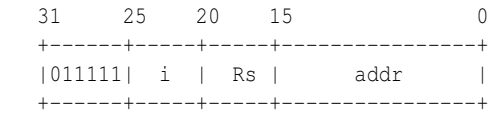
```
BBS Rs, i, addr
```

Description: The program counter (PC) is loaded with the contents of addr if bit i of the source register is set. The BBS instruction is a delayed branch instruction, with one delay slot.

Operation:

```
if ((Rs & (1 << i)) != 0) {
   PC = (addr << 2);
}
```

Instruction format:



13.4.2.2.11 BMI - Branch Minus

Syntax:

```
BMI Rs, addr
```

Description: The program counter (PC) is loaded with the contents of addr if the contents of the source register is less than zero.[5] The BMI instruction is a delayed branch instruction, with one delay slot.

Operation:

```
if ((Rs & 0x80000000) != 0) {
   PC = (addr << 2);
}
```

Instruction format:

```
  31          20    15               0
  +-----------+-----+----------------+
  |01111111111|  Rs |      addr      |
  +-----------+-----+----------------+
```

[5] BMI Rs, addr is the same as BBS Rs, 31, addr

13.4.2.2.12 BNZ - Branch Not Zero

Assembler syntax:

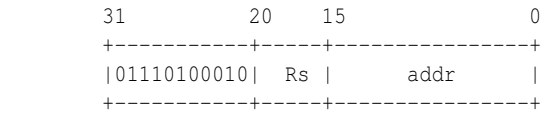
```
BNZ Rs, addr
```

Description: The program counter (PC) is loaded with the contents of addr if the contents of the source register is not equal to zero. The BNZ instruction is a delayed branch instruction, with one delay slot.

Operation:

```
if (Rs != 0) {
   PC = (addr << 2);
}
```

Instruction format:



13.4.2.2.13 BPL - Branch Plus

Assembler syntax:

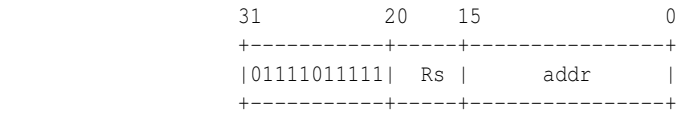
```
BPL Rs, addr
```

Description: The program counter (PC) is loaded with the contents of addr if the contents of the source register is greater than or equal to zero.[6] The BPL instruction is a delayed branch instruction, with one delay slot.

Operation:

```
if ((Rs & 0x80000000) == 0) {
   PC = (addr << 2);
}
```

Instruction format:



[6] BPL Rs, addr is the same as BBC Rs, 31, addr.

13.4.2.2.14 BZ - Branch Zero

Assembler syntax:

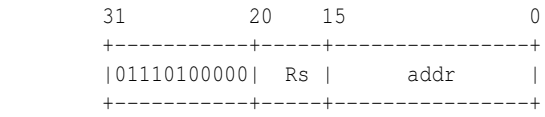
```
BZ Rs, addr
```

Description: The program counter (PC) is loaded with the contents of addr if the contents of the source register equals zero. The BZ instruction is a delayed branch instruction, with one delay slot.

Operation:

```
if (Rs == 0) {
   PC = (addr << 2);
}
```

Instruction format:



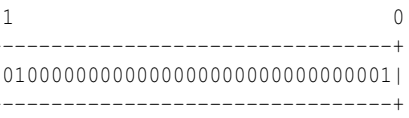
13.4.2.2.15 DI - Disable Interrupts

Assembler syntax:

```
DI
```

Description: Disable interrupts and the execution of instructions from the the instruction register.[7]

Instruction format:



[7] An interrupt routine can be executed immediately after the execution of the DI instruction. But there can be no interrupt after the first instruction following DI.

13.4.2.2.16 EI - Enable Interrupts

Syntax:

```
EI
```

Description: Enable interrupts and the execution of instructions from the the instruction register.[8]

Instruction format:

```
31                              0
+-------------------------------+
|01000000000000000000000000000011|
+-------------------------------+
```

[8] The execution of the first instruction of an interrupt routine, after enabling the interrupts, executes no earlier than after the first instruction following the EI instruction.

13.4.2.2.17 HALT - Halt the MPU

Syntax:

```
HALT
```

Description: Disables the MPU. The CPU can enable the MPU by writing to rw_ctrl.en.

Instruction format:

```
31                                0
+--------------------------------+
|01000000000000000000000000000010|
+--------------------------------+
```

13.4.2.2.18 JIR - Jump to Interrupt Routine (Address is an immediate)

Assembler syntax:

```
JIR addr
```

Description: The interrupt return pointer (IRP) is loaded with the contents of the program counter (PC) for the instruction to be executed after the interrupt routine.[9] PC is then loaded with the contents of the address operand. The JIR instruction is a delayed branch instruction, with one delay slot.

Operation:

```
IRP = Next PC;
PC = (addr << 2);
```

Instruction format:

```
31               15               0
+----------------+----------------+
|0110000000100000|      addr      |
+----------------+----------------+
```

[9] The IRP can not be put on a stack. If JIR is executed in an interrupt routine, the value of IRP will be overwritten. The JIR instruction should therefore never be executed in an interrupt routine.

13.4.2.2.19 JIR - Jump to Interrupt Routine (Address is a register)

Assembler syntax:

```
JIR Rn
```

Description: The interrupt return pointer (IRP) is loaded with the contents of the program counter (PC) for the instruction to be executed after the interrupt routine.[10] PC is then loaded with the contents of Rn. When N is set to 1, Rn becomes a special register rather than a general register. The JIR instruction is a delayed branch instruction, with one delay slot.

Operation:

```
IRP = Next PC;
PC = Rn;
```

Instruction format:

```
31      24         15    10          0
+-------+-+--------+-----+-----------+
|0110001|N|00100000|  Rn |00000000000|
+-------+-+--------+-----+-----------+
```

[10] The IRP can not be put on a stack. If JIR is executed in an interrupt routine, the value of IRP will be overwritten. The JIR instruction should therefore never be executed in an interrupt routine.

13.4.2.2.20 JNT - Jump Next Thread

Assembler syntax:

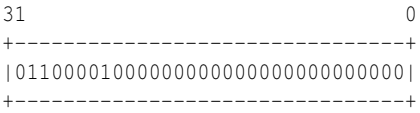
```
JNT
```

Description: Jump to the thread address which the thread pointer (TP) points at. The thread pointer is then updated and will point at the next thread address. The JNT instruction is a delayed jump instruction, with one delay slot.

Operation:

```
PC = ThreadAddr[TP];
TP = TP + 1;
```

Instruction format:



13.4.2.2.21 JSR - Jump to Subroutine (Address is an immediate)

Assembler syntax:

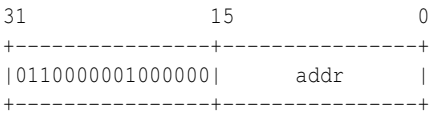
```
JSR addr
```

Description: The subroutine return pointer (SRP) is loaded with the contents of the program counter (PC) for the instruction to be executed after the subroutine.[11] PC is then loaded with the contents of the address operand. The JSR instruction is a delayed branch instruction, with one delay slot.

Operation:

```
SRP = Next PC;
PC = (addr << 2);
```

Instruction format:



[11] The SRP can not be put on a stack. If JSR is executed using the register interface while already running a subroutine, the first SRP value will be lost. The JSR instruction should therefore never be executed using the register interface.

13.4.2.2.22 JSR - Jump to Subroutine (Address is a register)

Assembler syntax:

```
JSR Rn
```

Description: The subroutine return pointer (SRP) is loaded with the contents of the program counter (PC) for the instruction to be executed after the subroutine.[12] PC is then loaded with the contents of Rn. When N is set to 1, Rn becomes a special register rather than a general register. The JSR instruction is a delayed branch instruction, with one delay slot.

Operation:

```
SRP = Next PC;
PC = Rn;
```

Instruction format:

```
  31      24         15    10          0
  +-------+-+--------+-----+-----------+
  |0110001|N|01000000|  Rn |00000000000|
  +-------+-+--------+-----+-----------+
```

[12] The SRP can not be put on a stack. If JSR is executed using the register interface while already running a subroutine, the first SRP value will be lost. The JSR instruction should therefore never be executed using the register interface.

13.4.2.2.23 LSL - Logical Shift Left

Assembler syntax:

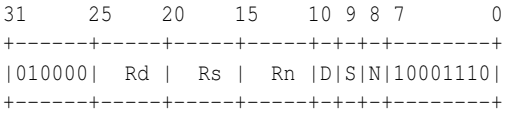
```
LSL Rs, Rn, Rd
```

Description: The source register is left shifted the number of steps specified by a general register or a special register and zero-filled.[13] When either D, S or N is set to 1, Rd, Rs or Rn respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs << Rn;
```

Instruction format:



D	S	N	General registers	Special registers
0	0	0	Rd Rs Rn	None
0	0	1	Rd Rs	Rn
0	1	0	Rd Rn	Rs
0	1	1	Rd	Rs Rn
1	0	0	Rs Rn	Rd
1	0	1	Rs	Rd Rn
1	1	0	Rn	Rd Rs
1	1	1	None	Rd Rs Rn

[13] A shift with 32 bits or more will give a zero result.

13.4.2.2.24 LSLQ - Logical Shift Left Quick

Assembler syntax:

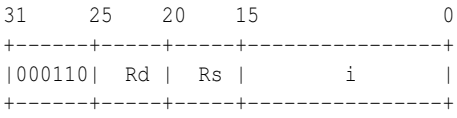
```
LSLQ Rs, i, Rd
```

Description: The source register is left shifted the number of steps specified by the 16-bit immediate value and zero-filled.[14]

Operation:

```
Rd = Rs << i;
```

Instruction format:



[14] A shift with 32 bits or more will give a zero result.

13.4.2.2.25 LSR - Logical Shift Right

Assembler syntax:

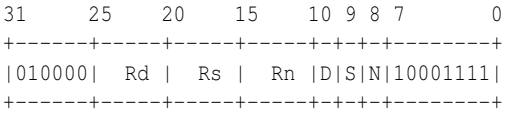
```
LSR Rs, Rn, Rd
```

Description: The source register is right shifted the number of steps specified by a general register or a special register and zero-filled.[15] When either D, S or N is set to 1, Rd, Rs or Rn respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs >> Rn;
```

Instruction format:



D	S	N	General registers	Special registers
0	0	0	Rd Rs Rn	None
0	0	1	Rd Rs	Rn
0	1	0	Rd Rn	Rs
0	1	1	Rd	Rs Rn
1	0	0	Rs Rn	Rd
1	0	1	Rs	Rd Rn
1	1	0	Rn	Rd Rs
1	1	1	None	Rd Rs Rn

[15] A shift with 32 bits or more will give a zero result.

13.4.2.2.26 LSRQ - Logical Shift Right Quick

Assembler syntax:

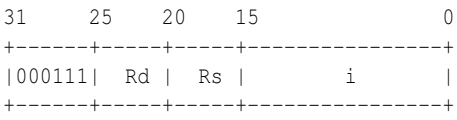
```
LSRQ Rs, i, Rd
```

Description: The source register is right shifted the number of steps specified by the 16-bit immediate value and zero-filled.[16]

Operation:

```
Rd = Rs >> i;
```

Instruction format:



[16] A shift with 32 bits or more will give a zero result.

13.4.2.2.27 LW - Load 32-bit data to register (Address is an immediate)

Assembler syntax:

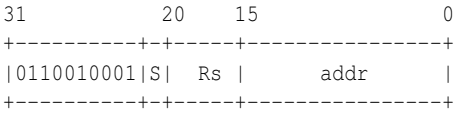
```
LW addr, Rs
```

Description: The destination register is loaded with the contents of memory data at address addr in the MPU memory. The MPU requires an extra clock cycle to execute the LW instruction. When S is set to 1, Rs becomes a special register rather than a general register.

Operation:

```
Rs = mem[addr << 2];
```

Instruction format:



13.4.2.2.28 LW - Load 32-bit data to register (Address is a register)

Assembler syntax:

```
LW Rn, b, Rs
```

It can also be written as the following when b=0:

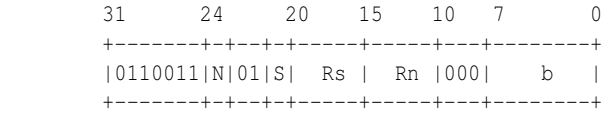
```
LW Rn, Rs
```

Description: The destination register is loaded with the contents of memory data at the address stored in Rn. The contents of Rn is changed by adding an 8-bit unsigned immediate value, which is zero extended to 32 bits. The MPU will insert a delay after the LW instruction has been executed. When either N or S is set to 1, Rn or Rs respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rs = mem[Rn];
Rn = Rn + b;
```

Instruction format:



S	N	General registers	Special registers
0	0	Rn Rs	None
0	1	Rn	Rs
1	0	Rs	Rn
1	1	None	Rn Rs

13.4.2.2.29 MOVE - Move to Register

Assembler syntax:

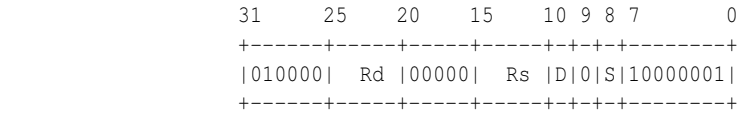
```
MOVE Rs, Rd
```

Description: Move data from source register to the destination register. When either D or S is set to 1, Rd or Rs respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs;
```

Instruction format:



D	S	General registers	Special registers
0	0	Rd Rs	None
0	1	Rd	Rs
1	0	Rs	Rd
1	1	None	Rd Rs

13.4.2.2.30 MOVEQ - Move Quick

Assembler syntax:

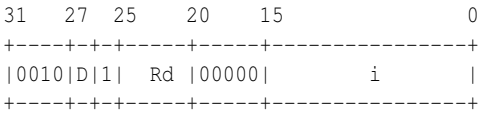
```
MOVEQ i, Rd
```

Description: The 16-bit immediate value is zero extended to 32 bits and then moved to the destination register. When D is set to 1, Rd becomes a special register rather than a general register.

Operation:

```
Rd = i;
```

Instruction format:



13.4.2.2.31 MOVEX - Move Extended

Assembler syntax:

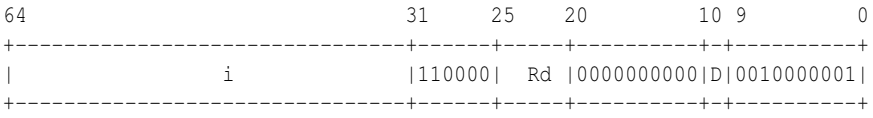
```
MOVEX i, Rd
```

Description: Move the 32-bit immediate value to the destination register. When D is set to 1, Rd becomes a special register rather than a general register.

Operation:

```
Rd = i;
```

Instruction format:



13.4.2.2.32 NOP - No Operation

Assembler syntax:

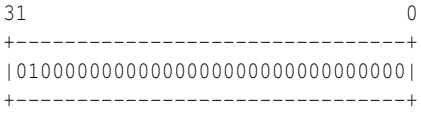
```
NOP
```

Description: No operation

Operation:

```
;
```

Instruction format:



13.4.2.2.33 NOT - Logical Complement

Assembler syntax:

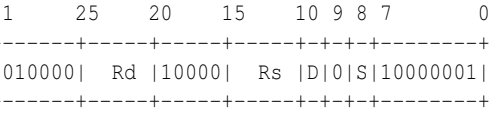
```
NOT Rs, Rd
```

Description: The contents of the source register is bitwise inverted (1's complement). When either D or S is set to 1, Rd or Rs respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = ~Rs;
```

Instruction format:



D	S	General registers	Special registers
0	0	Rd Rs	None
0	1	Rd	Rs
1	0	Rs	Rd
1	1	None	Rd Rs

13.4.2.2.34 OR - Logical OR

Assembler syntax:

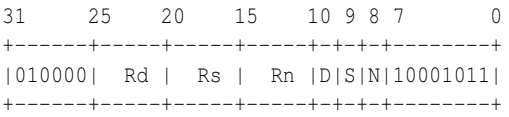
```
OR Rs, Rn, Rd
```

Description: A bitwise OR is performed between the source register and the contents of a general register or a special register. The result is stored in the destination register. When either D, S or N is set to 1, Rd, Rs or Rn respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs | Rn;
```

Instruction format:



D	S	N	General registers	Special registers
0	0	0	Rd Rs Rn	None
0	0	1	Rd Rs	Rn
0	1	0	Rd Rn	Rs
0	1	1	Rd	Rs Rn
1	0	0	Rs Rn	Rd
1	0	1	Rs	Rd Rn
1	1	0	Rn	Rd Rs
1	1	1	None	Rd Rs Rn

13.4.2.2.35 ORQ - Logical OR Quick

Assembler syntax:

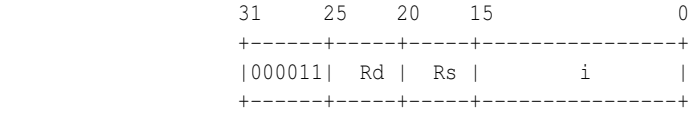
```
ORQ Rs, i, Rd
```

Description: The 16-bit immediate value is zero extended to 32 bits, and then a bitwise OR is performed between it and the contents of a general register. The result is stored in the destination register. Both Rs and Rd are general registers.

Operation:

```
Rd = Rs | i;
```

Instruction format:



13.4.2.2.36 ORX - Logical OR Extended

Assembler syntax:

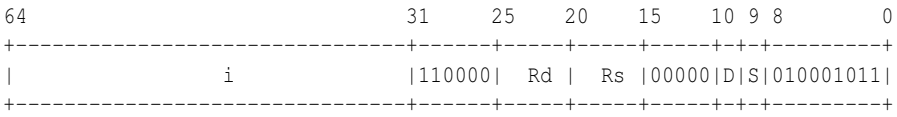
```
ORX Rs, i, Rd
```

Description: A bitwise OR is performed between a 32-bit immediate value and the contents of a general register or a special register. The result is stored in the destination register. When either D or S is set to 1, Rd or Rs respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs | i;
```

Instruction format:



D	S	General registers	Special registers
0	0	Rd Rs	None
0	1	Rd	Rs
1	0	Rs	Rd
1	1	None	Rd Rs

13.4.2.2.37 RET - Return from Subroutine

Assembler syntax:

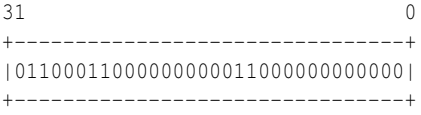
```
RET
```

Description: The contents of the subroutine return pointer (SRP) is loaded to PC.[17] The RET instruction is a delayed jump instruction, with one delay slot.

Operation:

```
PC = SRP;
```

Instruction format:



[17] RET is the same as BAR SRP.

13.4.2.2.38 RETI - Return from Interrupt

Assembler syntax:

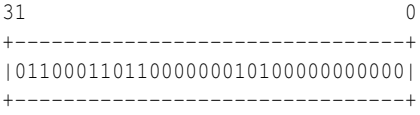
```
RETI
```

Description: The contents of the interrupt return pointer (IRP) is loaded to PC. The RETI instruction is a delayed jump instruction, with one delay slot.

Operation:

```
PC = IRP;
```

Instruction format:



13.4.2.2.39 RR - Register Read (Address is an immediate)

Assembler syntax:

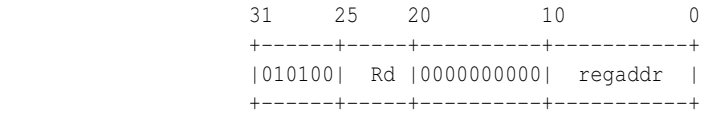
```
RR regaddr, Rd
```

Description: Move data from register at address regaddr using the register interface. The data is stored in the destination register.[18]

Operation:

```
Rd = REGIF[regaddr << 2];
```

Instruction format:



[18] The destination of the RR instruction can not be used by the instruction directly after the RR instruction, since the updated or old value of the register may be referenced depending on whether an interrupt occurred between the two instructions or not.

13.4.2.2.40 RR - Register Read (Address is a register)

Assembler syntax:

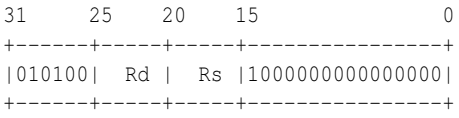
```
RR Rs, Rd
```

Description: Move data from register at address stored in Rs, using the register interface. The data is stored in the destination register.[19] Rs is a special register.

Operation:

```
Rd = REGIF[Rs];
```

Instruction format:



[19] An RR instruction can not use the destination register of the immediately preceding instruction as the address register. Neither can the destination of the RR instruction be used by the instruction directly after the RR instruction, since the updated or old value of the register may be referenced depending on whether an interrupt occurred between the two instructions or not.

13.4.2.2.41 RW - Register Write (Address is an immediate)

Assembler syntax:

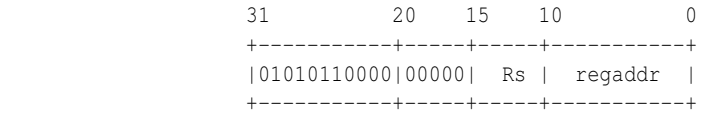
```
RW Rs, regaddr
```

Description: Move data from the source register using the register interface. The data is stored in the register at address regaddr.

Operation:

```
REGIF[regaddr << 2] = Rs;
```

Instruction format:



13.4.2.2.42 RW - Register Write (Address is a register)

Assembler syntax:

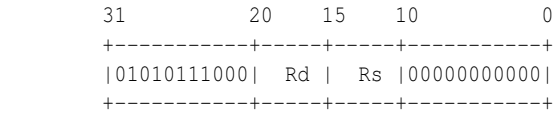
```
RW Rs, Rd
```

Description: Move data from the source register using the register interface. The data is stored in the register at the address stored in Rd. Rd is a special register.

Operation:

```
REGIF[Rd] = Rs;
```

Instruction format:



13.4.2.2.43 RWQ - Register Write Quick (Address is an immediate)

Assembler syntax:

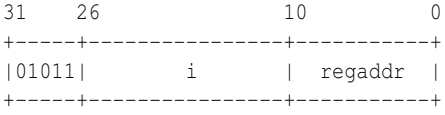
```
RWQ i, regaddr
```

Description: The 16-bit immediate value is zero extended and then moved to the register at address regaddr using the register interface.

Operation:

```
REGIF[regaddr << 2] = i;
```

Instruction format:



13.4.2.2.44 RWQ - Register Write Quick (Address is a register)

Assembler syntax:

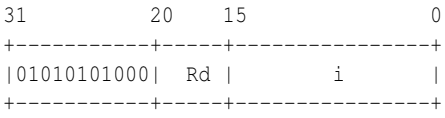
```
RWQ i, Rd
```

Description: The 16-bit immediate value is zero extended and then moved to the register at the address stored in Rd using the register interface. Rd is a special register.

Operation:

```
REGIF[Rd] = i;
```

Instruction format:



13.4.2.2.45 RWX - Register Write Extended (Address is an immediate)

Assembler syntax:

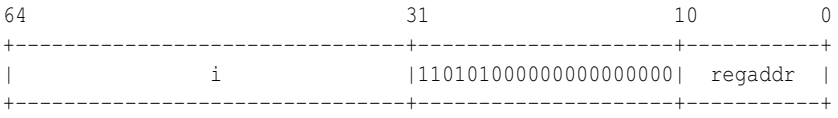
```
RWX i, regaddr
```

Description: Move the 32-bit immediate value to the register at address regaddr using the register interface.

Operation:

```
REGIF[regaddr << 2] = i;
```

Instruction format:



13.4.2.2.46 RWX - Register Write Extended (Address is a register)

Assembler syntax:

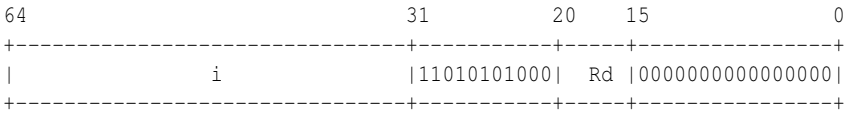
```
RWX i, Rd
```

Description: Move the 32-bit immediate value to the register at address stored in the destination register using the register interface. Rd is a special register.

Operation:

```
REGIF[Rd] = i;
```

Instruction format:



13.4.2.2.47 SUB - Subtract

Assembler syntax:

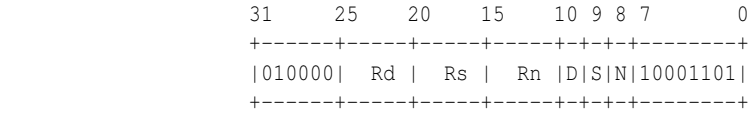
```
SUB Rs, Rn, Rd
```

Description: The general register is subtracted from the contents of the source register, and the result is stored in the destination register. When either D, S or N is set to 1, Rd, Rs or Rn respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs - Rn;
```

Instruction format:



D	S	N	General registers	Special registers
0	0	0	Rd Rs Rn	None
0	0	1	Rd Rs	Rn
0	1	0	Rd Rn	Rs
0	1	1	Rd	Rs Rn
1	0	0	Rs Rn	Rd
1	0	1	Rs	Rd Rn
1	1	0	Rn	Rd Rs
1	1	1	None	Rd Rs Rn

13.4.2.2.48 SUBQ - Subtract Quick

Assembler syntax:

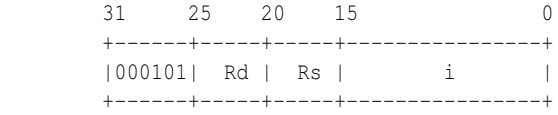
```
SUBQ Rs, i, Rd
```

Description: A 16-bit immediate value is zero extended to 32-bits and then subtracted from the contents of the source register. The result is stored in the destination register. Both Rs and Rd are general registers.

Operation:

```
Rd = Rs - i;
```

Instruction format:



13.4.2.2.49 SUBX - Subtract Extended

Assembler syntax:

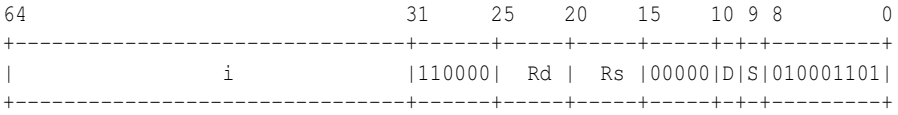
```
SUBX Rs, i, Rd
```

Description: A 32-bit immediate value is subtracted from the contents of the source register, and the result is stored in the destination register. When either D or S is set to 1, Rd or Rs respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs - i;
```

Instruction format:



D	S	General registers	Special registers
0	0	Rd Rs	None
0	1	Rd	Rs
1	0	Rs	Rd
1	1	None	Rd Rs

13.4.2.2.50 SW - Store 32-bit data to memory (Address is an immediate)

Assembler syntax:

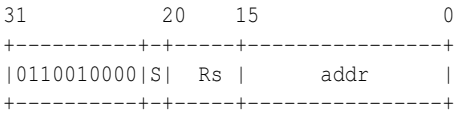
```
SW Rs, addr
```

Description: The contents of the source register will be stored in the MPU memory at address addr. The MPU requires an extra clock cycle to execute the SW instruction. When S is set to 1, Rs becomes a special register rather than a general register.

Operation:

```
mem[addr << 2] = Rs;
```

Instruction format:



13.4.2.2.51 SW - Store 32-bit data to memory (Address is a register)

Assembler syntax:

```
SW Rs, b, Rn
```

It can also be written as the following when b=0:

```
SW Rs, Rn
```

Description: The contents of the source register will be stored in the MPU memory at the address stored in Rn. The contents of Rn is changed by adding an 8-bit unsigned immediate value, which is zero extended to 32 bits, to the register. The MPU requires an extra clock cycle to execute the SW instruction. When either N or S is set to 1, Rn or Rs respectively, becomes a special register rather than a general register. See table below.

Operation:

```
mem[Rn] = Rs;
Rn = Rn + b;
```

Instruction format:

```
31      24     20    15    10   7        0
+-------+-+--+-+-----+-----+---+--------+
|0110011|N|00|S|  Rs |  Rn |000|    b   |
+-------+-+--+-+-----+-----+---+--------+
```

N	S	General registers	Special registers
0	0	Rn Rs	None
0	1	Rn	Rs
1	0	Rs	Rn
1	1	None	Rn Rs

13.4.2.2.52 SWX - Store 32-bit data to memory (Address is an immediate)

Assembler syntax:

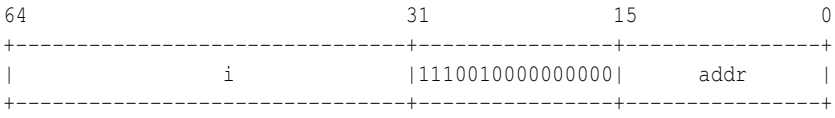
```
SWX i, addr
```

Description: Store a 32-bit immediate value to MPU memory at address addr. The MPU requires an extra clock cycle to execute the SWX instruction.

Operation:

```
mem[addr << 2] = i;
```

Instruction format:



13.4.2.2.53 SWX - Store 32-bit data to memory (Address is a register)

Assembler syntax:

```
SWX i, b, Rn
```

It can also be written as the following when b=0:

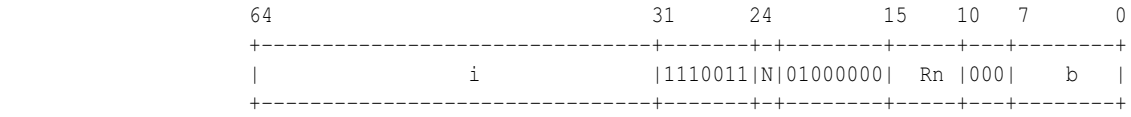
```
SWX i, Rn
```

Description: Store a 32-bit immediate value to MPU memory at the address which is stored in Rn. The contents of Rn is changed by adding an 8-bit unsigned immediate value, which is zero extended to 32 bits. The MPU requires an extra clock cycle to execute the SWX instruction. When N is set to 1, Rn becomes a special register rather than a general register.

Operation:

```
mem[Rn] = i;
Rn = Rn + b;
```

Instruction format:



13.4.2.2.54 XOR - Logical Exclusive OR

Assembler syntax:

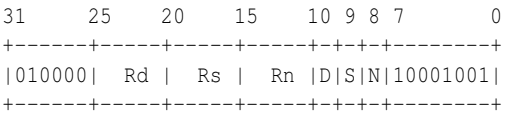
```
XOR Rs, Rn, Rd
```

Description: A bitwise XOR is performed between the source register and the contents of a general register or a special register. The result is stored in the destination register. When either D, S or N is set to 1, Rd, Rs or Rn respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs ^ Rn;
```

Instruction format:



D	S	N	General registers	Special registers
0	0	0	Rd Rs Rn	None
0	0	1	Rd Rs	Rn
0	1	0	Rd Rn	Rs
0	1	1	Rd	Rs Rn
1	0	0	Rs Rn	Rd
1	0	1	Rs	Rd Rn
1	1	0	Rn	Rd Rs
1	1	1	None	Rd Rs Rn

13.4.2.2.55 XOR - Register Exclusive OR

Assembler syntax:

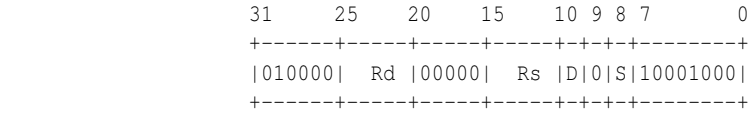
```
XOR Rs, Rd
```

Description: An XOR is performed between each of the source register bits. The one bit result is stored in the destination register. When either D or S is set to 1, Rd or Rs respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs[0] ^ Rs[1] ... ^ Rs[31];
```

Instruction format:



D	S	General registers	Special registers
0	0	Rd Rs	None
0	1	Rd	Rs
1	0	Rs	Rd
1	1	None	Rd Rs

13.4.2.2.56 XORQ - Logical Exclusive OR Quick

Assembler syntax:

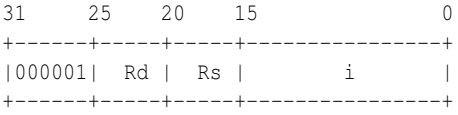
```
XORQ Rs, i, Rd
```

Description: The 16-bit immediate value is zero extended to 32 bits and then XOR is performed between it and the contents of a general register. The result is stored in the destination register. Both Rs and Rd are general registers.

Operation:

```
Rd = Rs ^ i;
```

Instruction format:



13.4.2.2.57 XORX - Logical Exclusive OR Extended

Assembler syntax:

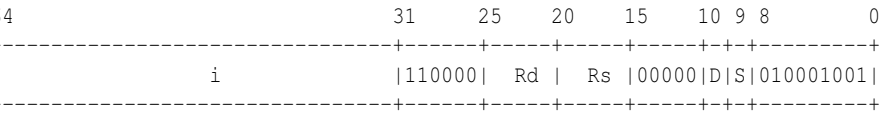
```
XORX Rs, i, Rd
```

Description: A bitwise XOR is performed between a 32-bit immediate value and the contents of a general register or a special register. The result is stored in the destination register. When either D or S is set to 1, Rd or Rs respectively, becomes a special register rather than a general register. See table below.

Operation:

```
Rd = Rs ^ i;
```

Instruction format:



D	S	General registers	Special registers
0	0	Rd Rs	None
0	1	Rd	Rs
1	0	Rs	Rd
1	1	None	Rd Rs

13.5 Slave Processing Unit

The Slave Processing Unit (SPU) is a micro code driven processor used for time critical protocols or time critical parts within a protocol. The SPU contains several registers which all can be accessed from the owner (CPU, MPU or another SPU).

The SPU memory is filled from the Memory Controller. The size of each SPU memory in ETRAX FS is 4096 bytes (128x256).

The SPU has two 32 bit wide I/O buses and 32 general I/O (GIO) signals.

In the SPU there are two different modes, one sequential (SEQ) mode and one mode allowing the SPU to execute code representing a Finite State Machine (FSM). The latter mode will be referred to as FSM mode. It is possible to switch between SEQ and FSM mode in runtime. See 13.5.1.5.

The sequential mode has the following characteristics:

- Instruction size is 32 bits.
- Instructions are:
  - Common ALU operations
  - Register operations (e.g., setting and reading registers)
  - Branch instructions

The FSM mode has the following characteristics:

- Setting output signals and next state.
- A sequential instruction can be performed in each FSM state.
- ALU flags are sampled on every entrance to a state and can be used as FSM inputs. Note that a sequential instruction can not affect the flags in the current state.
- Includes a state timer. The timer is used to change state after a specified time. The start value to the state timer can be specified in the state code. It is also possible to use a register from the SPU register bank (R0 - R15) to specify the time.

13.5.1 Architectural description

The SPUs main purpose is handling I/O protocols. This can, depending on the protocol, be done entirely by the SPU or by letting the SPU control other I/O Processor modules.

13.5.1.1 Enabling and disabling SPU modes from owner

As described earlier the SPU has two modes of operation, SEQ and FSM mode. It is possible to switch between these modes in runtime. The SPU itself can do so by using

instructions. The owner of the SPU uses the following methods to enable or disable the SPU:

To enable sequential mode:[20]

1. Update rw_seq_pc.addr
2. Write fsm = no and en = no to rw_ctrl
3. Write fsm = no and en = yes to rw_ctrl

To enable FSM mode: [20]

1. Update rw_fsm_pc.addr
2. Write fsm = yes and en = no to rw_ctrl
3. Write fsm = yes and en = yes to rw_ctrl

To disable any mode:

- Write en = no to rw_ctrl

13.5.1.2 Registers

For register descriptions see 25.25. All registers except for IMMHI can be accessed from the SPU owner.

13.5.1.2.1 General registers

The SPU contains sixteen 32-bit wide general registers (R0-R15). The general registers can be accessed by the owner using the vector register rw_r.

When the owner (CPU, MPU or other SPU) writes to R0-R15 the corresponding bit is set in WSTS (rs_wr_stat). This bit can be cleared with an ALU operation. The example below shows a method for clearing bit 0.[21]

```
ANDQ WSTS, 0xfffe, WSTS
```

13.5.1.2.2 Special registers

The SPU architecture also defines 16 special registers (P0-P15). The special registers are:

Mnemonic	Reg. no.	Description	Mode register

[20] Two separate writes to rw_ctrl must be done.

[21] If owner writes to R0 at the same time as the above instruction is performed, bit 0 will remain set.

SEQPC	P0	Sequential mode PC	rw_seq_pc.addr
FSMPC	P1	FSM mode PC	rw_fsm_pc.addr
REGA	P2	Register access	rw_reg_access
FSM3_0	P3	FSM inputs register for input 3 - 0	rw_fsm_inputs3_0
FSM7_4	P4	FSM inputs register for input 7 - 4	rw_fsm_inputs7_4
GOUT	P5	General I/O out register	rw_gio_out
B0OUT	P6	Bus 0 out register	rw_bus0_out
B1OUT	P7	Bus 1 out register	rw_bus1_out
GIN	P8	General I/O in register	r_gio_in
B0IN	P9	Bus 0 in register	r_bus0_in
B1IN	P10	Bus 1 in register	r_bus1_in
STATIN	P11	Status in register	r_stat_in
TRIGGER	P12	Trigger in status register, read only	r_trigger_in
IMMHI	P12	Immediate hi value, write only	
WSTS	P13	Owner register write attention	r_wr_stat rs_wr_stat
SPECS	P14	Special signals	r_special_stat
INDEX	P15	Indexed register using bits of bus0	r_reg_indexed_by_bus0_in

Table 13.26: *SPU, Special registers*

SEQ_PC and FSM_PC

These registers contain the program counter (PC) for the two different modes of operation.

Both registers store the address in bytes:

- SEQ_PC is 32 bit aligned, the SPU can access this as SEQPC (P0)
- FSM_PC is 64 bit aligned, the SPU can access this as FSMPC (P1)

The SEQ_PC register is also used for setting memory address when the SPU memory is filled with code via the Memory Controller (MC). While doing this the SPU must be disabled and set to sequential mode.

Indexed register

The INDEX (P15) register uses bus0_in[3:0] as an index, selecting one of the R0 - R15 registers, i.e., P15 = R[bus_in0[3:0]].

13.5.1.2.3 Event registers

To support FSM events there are 13 event registers. For an explanation on FSM events see section 13.5.1.14.

Mnemonic	Reg. no.	Description	Mode register
ECFG0	E0	Event 0 configuration	rw_event_cfg
ECFG1	E1	Event 1 configuration	rw_event_cfg
ECFG2	E2	Event 2 configuration	rw_event_cfg
ECFG3	E3	Event 3 configuration	rw_event_cfg

EMASK0	E4	Event 0 mask	rw_event_mask
EMASK1	E5	Event 1 mask	rw_event_mask
EMASK2	E6	Event 2 mask	rw_event_mask
EMASK3	E7	Event 3 mask	rw_event_mask
EVAL0	E8	Event 0 value	rw_event_val
EVAL1	E9	Event 1 value	rw_event_val
EVAL2	E10	Event 2 value	rw_event_val
EVAL3	E11	Event 3 value	rw_event_val
EADDR	E12	Event return address	rw_event_ret.addr

Table 13.27: *SPU, Event registers*

13.5.1.3 Instruction formats

All sequential instructions are 32 bits wide.

FSM state instructions are composed of:

- One FSM instruction, 32-bit - Mandatory
- One SEQ instruction, 32-bit - Optional [22] [23]
- One Timer instruction, 32-bit - Optional [24]
- 0-8 Transition instructions, 24-bit each - Optional [25]

13.5.1.4 Branches

The branch instructions in the SPU sequential mode all have one delay slot. In other words, the instruction directly after the branch instruction will always be executed.

If a register used as the conditional register by either a BBC, BBS, BMI, BNZ, BPL, or BZ instruction is the destination register of the immediately preceding instruction, the branch instruction will reference the register before it is updated.

- **Example:**

```
MOVEQ 4, R0
SUBQ R0, 4, R0
BNZ R0, addr
```

[22] All SPU sequential instructions except branches and start FSM (i.e., except for BA, BAR, BBC, BBS, BMI, BNZ, BPL, BZ, FSM, FSMQ, and HALT) can be used.

[23] In FSM mode the sequential instructions can use a 32-bit immediate value, by using the bits used by the timer instruction.

[24] No timer instruction can be used if an instruction holding a 32-bit immediate value is used as SEQ instruction.

[25] If both a SEQ instruction and a Timer instruction (or a SEQ instruction with 32-bit immediate) is used in a state, a maximum of six Transition instructions can be used. If only one of the SEQ instruction or the Timer instruction is used, a maximum of eight Transition instructions can be used.

The branch will be taken, since BNZ sees the old value (4) of R0.

All instructions are valid in a delay slot except for:

Branch instructions (BA/BAR/BBC/BBS/BMI/BNZ/BPL/BZ/FSM/FSMQ)

Possible branch source operands for the branch instructions are R0 - R15, P5(GOUT), P8(GIN), P11(STATIN), P12(TRIGGER), P13(WSTS) or P14(SPECS).

13.5.1.5 Switching between SEQ and FSM mode

It is possible to switch between SEQ and FSM mode in runtime, using the FSM instruction when going from SEQ to FSM mode, and by using the go_seq bit in the state encoding when going from FSM to SEQ mode. See sections 13.5.1.5.1 and 13.5.1.5.2.

13.5.1.5.1 From Sequential mode to FSM mode

To change from SEQ mode to FSM mode the sequential instructions FSM or FSMQ can be used. These instructions are treated as branches and have a delay slot.

If FSM is used, the value of special register FSM_PC can not be the result from the previous instruction.

- **Example:**

```
MOVEQ 8, FSM_PC
SUBQ FSM_PC, 4, FSM_PC
FSM
```

In this example the SPU will enter FSM-mode at address 0x8, since the FSM instruction uses the old value (8) of FSM_PC.

SEQ_PC is updated to point to the instruction following the FSM/FSMQ delay slot, so when returning from FSM mode the SEQ program will continue.

- **Example:**

```
ADDQ 1, R0, R0
FSMQ state_4
ADDQ 2, R0, R0 ; Delay slot instruction
SUBQ R1, 6, R2
```

This example will increase the value of R0 with three, then enter FSM mode at state_4. When returning from FSM mode (without specifying or changing SEQ_PC) the SUBQ instruction will be performed.

13.5.1.5.2 From FSM mode to Sequential mode

To change from FSM mode to SEQ mode the go_seq bit in the FSM instruction is used. The operation has one delay slot (as for all FSM state changes) and takes 10 ns to perform. If the seq_only bit is also set in the FSM instruction the only address is a SEQ address.

- **Example:**

```
state_1: go_seq
         only 0x24
```

  In this example the SPU will enter SEQ-mode at address 0x24 10 ns after entering state_1.

If go_seq is used without the 'only' tag the SEQ address is taken from SEQ_PC. SEQ_PC takes 15 ns to be updated (from FSM mode) so previous state might not be used to update SEQ_PC.

- **Example:**

```
state_2: MOVEQ 8, SEQ_PC
         only state_3

state_3: MOVEQ 4, SEQ_PC
         only state_4

state_4: go_seq
```

  In this example the SPU will enter SEQ-mode at address 0x8, since the go_seq uses the old value (8) of SEQ_PC.

13.5.1.6 Register operations

The SPU can access registers in other I/O Processor modules (if configured as the module owner in the Switch). The register instructions are:

RR - Register read

RW - Register write

RWQ - Register write with 16 bit immediate [26]

RRM - Register Read Masked

RRMQ - Register Read Masked with 16 bit immediate [27]

[26] The upper 16 bits of register are taken from special register REGA bits 31 - 16.
[27] The upper 16 bits will always be zero. This operation only affects the z-flag.

RRMH - Register Read Masked with 16 bit immediate [27]

In the above instructions the address can be an immediate or come from special register REGA (P2). The result of a read operation is delayed until after the instruction following the read instruction.

It is not possible to have a RR (or RRM, RRMH, RRMQ) instruction directly after a RW (or RWQ) instruction.

If a branch instruction uses the read result a NOP instruction (or similar) is required, since the read result is delayed.

- **Example:**

```
RR 124, R3
BZ R3, label
```

In the above code, the BZ instruction uses the previous value of R3, not the value of register 124. In order to use the result from the RR instruction to the BZ instruction the following can be used:

```
RR 124, R3
NOP
BZ R3, label
```

13.5.1.7 32 bit ALU operations using IMMHI

To support 32 bit immediate instructions in the SPU the IMMHI special register can be used. This register is used to write the 16 upper bits of an immediate operation. All quick instructions except for `RWQ`.

- **Example:**

```
MOVEQ 0x1234, IMMHI
ADDQ 0x1, R0, R2
```

This example will add 0x12340001 to R0 and store the result in R2.

The IMMHI register can only be used in the instruction following directly after it.

- **Example:**

```
MOVEQ 0x1234, IMMHI
NOP
ADDQ 0x1, R0, R2
```

This example will add 0x1 to R0 and store the result in R2.

13.5.1.8 ALU mask operations

All ALU operations with no immediate can have a mask on Rs and/or Rn. The mask can be any of registers R0-R15. The selected mask operation can be & (AND) or | (OR). Refer to 13.5.2.2 for a complete explanation of the coding in the sequential instruction.

- **Examples:**

```
ADD (R3 & R2), R1, R0

ADD R3, (R1 | R15), R0

ADD (R3 & R2), (R1 & R2), R0

ADD (R3 | R2), (R1 | R2), R0
```

The register used as mask can be the destination register of the previous instruction.

- **Example:**

```
MOVEQ 0xffff, R1
ADD (R3 & R1), R2, R0 ; Uses 0xffff to mask R3
```

13.5.1.9 ALU flags

All ALU operations affect the N, Z, V, C flags:[28] [29]

N - Negative Most significant bit of ALU result operand == one.

```
N = res[31]
```

Z - Zero ALU result == zero.

```
Z = !res[31] & ... & !res[0]
```

V - Overflow ALU operations set V to 0 with the following exceptions:

- Addition (ADD, ADDQ): If the two source operands and the ALU result have different signs (most significant bit) the V flag is set.

```
V = rs[31] & rn[31] & !res[31] | !rs[31] & !rn[31] & res[31]
```

[28] The flags can be used as inputs in FSM mode and also read in r_special_stat (SPECS).
[29] Non ALU operations do not affect the flags i.e., the flags keep their current values.

- Subtraction (SUB, SUBQ): If the two source operands and the ALU result have different signs (most significant bit) the V flag is set.

```
V = !rs[31] & rn[31] & !res[31] | rs[31] & !rn[31] & res[31]
```

C - Carry ALU operations set C to 0 with the following exceptions:

- Addition (ADD, ADDQ):

```
C = (rn[31] & !res[31]) | (rs[31] & !res[31])
```

- Subtraction (SUB, SUBQ):

```
C = (rn[31] & res[31]) | (!rs[31] & res[31])
```

- Shift left (LSL, SSL):

```
C = rs[31]
```

- Shift right (LSR, SSR):

```
C = rs[0]
```

13.5.1.10 Inputs

The SPU has 32 general inputs and two 32-bit wide input buses. All inputs can be read by GIN (P8), B0IN (P9) and B1IN (P10). Inputs to the FSM mode are selected by FSM3_0 (P3) and FSM7_4 (P4) and also in each state.

13.5.1.11 Outputs

The SPU has two 32-bit wide output buses whose values can be set by writing to B0OUT (P6) and B1OUT (P7). The SPU can also control 32-bit general outputs by writing to GOUT (P5).

When the SPU runs in FSM mode, general outputs GOUT[7:0] can only be modified from state outputs, but the FSM mode could use SEQ instructions to set or clear other output signals using GOUT. The SPU owner can control outputs by using registers in 25.25 (even GOUT[7:0] in FSM mode).

13.5.1.12 State transitions

All state transitions takes 10 ns but the state output signals are set after 5 ns. If no state transitions match, the FSM will remain in the current state. If more than one state transition is true, the first one is selected. For example:

- **Example:**

```
FSM_1:  ?_0_0_0    : 0_0_0_1 : FSM_2
        ?_0_0_1    : 1_0_0_1 : FSM_3
        1_0_0_0    : 0_1_0_1 : FSM_4
```

If the input combination is 1_0_0_0 FSM_2 will be the next state.

13.5.1.13 FSM mode inputs

The inputs to the FSM mode can be:

- ALU flags

  0 c-flag
  1 v-flag
  2 z-flag
  3 n-flag

- General registers bit 0
  bit[0] from general registers R0 - R15
- SPU GIO_IN signals
- SPU TRIGGER_IN signals
- SPU STATUS_IN signals
- WSTS ”Register attention” signals
- Bus xor signals

  0 xor on (bus0[31:0], R2[0]) i.e.,

  ```
  bus0[31] ^ ... ^ bus0[0] ^ R2[0]
  ```

  1 xor on (bus1[31:0], R3[0]) i.e.,

  ```
  bus1[31] ^ ... ^ bus1[0] ^ R3[0]
  ```

  2 xor on (bus0[31:0] masked with R0[31:0], R2[0]) i.e.,

  ```
  (bus0[31] & R0[31]) ^ ... ^ (bus0[0] & R0[0]) ^ R2[0]
  ```

  3 xor on (bus1[31:0] masked with R1[31:0], R3[0]) i.e.,

  ```
  (bus1[31] & R1[31]) ^ ... ^ (bus1[0] & R1[0]) ^ R3[0]
  ```

· SPU GIO_OUT signals gio_out[7:0]

Eight inputs are selected by registers rw_fsm_inputs3_0 and rw_fsm_inputs7_4. These eight inputs are delayed one clock cycle and will thus be 5 ns delayed.

Each FSM state selects four of the above signals as inputs for that state.

13.5.1.14 FSM events

In addition to the four selected inputs to each FSM state there are four events that can trigger the FSM to change state. The events are configured using the Event registers listed in section 13.5.1.2.3. The FSM instruction holds fields for enable and/or disable events and controls the update of them. As for the FSM inputs all input signals to the events are delayed 5 ns.

The FSM event works like this:

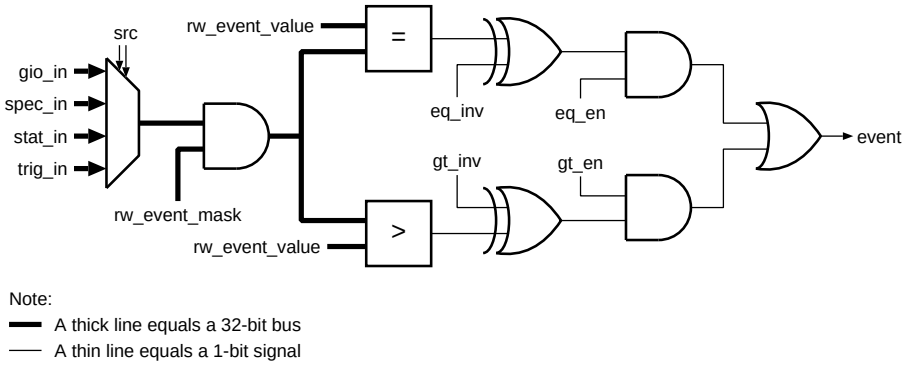


Figure 13.4: *SPU, FSM event*

If more than one enabled event triggers, event0 has the highest priority and event3 the lowest.

The event_value can be updated with the masked input signal. This is controlled by the do_event_upd and event_upd_mask fields of the FSM instruction, see table 13.32.

When an enabled event triggers, the FSM moves to the state defined by that events cfg register. The current state address is stored in register rw_event_ret. This register can be used in an FSM only instruction using event register EADDR (E12).

13.5.1.14.1 Configuring an event

Events are configured using the event registers ECFG0 - ECFG3 (or by the SPU owner by using the vector register rw_event_cfg).

There are four possible event sources which are listed below:

1. gio_in: The 32-bit gio_in connected to the SPU

2. wsts_gioout_spec: This 32-bit bus consists of: register write status, gio_out[7:0], fsm_selectable inputs, xor, and events. See the list below:

   [31:24] WSTS[7:0] for event 0 & 2, WSTS[15:8] for event 1 & 3

   [23:16] SPU gio_out[7:0]

   [15:8] FSM selectable inputs [7:0]

   [7] XOR, xor_bus1m_r3_0

   [6] XOR, xor_bus0m_r2_0

   [5] XOR, xor_bus1_r3_0

   [4] XOR, xor_bus0_r2_0

   [3:0] EVENT [3:0]

3. stat_in: The 32-bit stat_in connected to the SPU (r_stat_in)

4. trig: 32 trigger outputs (r_trigger_in)

Event masks are configured using the event registers EMASK0 - EMASK3 (or by the SPU owner by using the vector register rw_event_mask). Event values are configured using the event registers EVAL0 - EVAL3 (or by the SPU owner by using the vector register rw_event_val).[30]

13.5.1.15 Breakpoints

The SPU can have four hardware breakpoints. Breakpoints are configured by register vector rw_brp[3:0]. When a breakpoint is enabled it will halt the SPU when the address (either rw_fsm_pc or rw_seq_pc selected by rw_brp.fsm field) matches the rw_brp.addr field.

13.5.1.16 Trace registers

To support some elementary debugging of SPU programs some internal state information can be read out by using the r_trace and r_fsm_trace registers. These registers are updated while the SPU is enabled but stops updating when the SPU is disabled. The SPU disable can occur on halt instructions or by breakpoints.

13.5.2 Instruction set description

13.5.2.1 Definitions

i	Immediate value which is either 5, 14 or 16 bits wide
addr	Instruction address

[30] The only (seq) instruction that can be used to write to the event registers, see table 13.27, is the MOVE instruction.

regaddr	Address to a register in one of the I/O Processor register banks. This field points to a 32 bit entity in the mode register memory space. To get the byte address the field is shifted left by two steps (regaddr << 2).
Rd, Rn and Rs	Index of a general register (R0-R15) or a special register (P0-P15)
Ed and Es	Index of a event register (E0-E12)

13.5.2.2 ALU mask fields

All non immediate ALU instructions support masking of Rs and/or Rn. This is controlled by the ctr and mask fields in the instructions.

- seq_instr[10:7] = mask[3:0], selects mask register 0x0 = r0, 0xf = r15
- seq_instr[6:4] = ctr[2:0]

ctr[2]	operation
0	AND
1	OR

Table 13.29: *SPU, ALU mask ctr[2] field*

ctr[1]	operation
0	Don't mask Rs
1	Mask Rs

Table 13.30: *SPU, ALU mask ctr[1] field*

ctr[0]	operation
0	Don't mask Rn
1	Mask Rn

Table 13.31: *SPU, ALU mask ctr[0] field*

- **Examples:**

```
ADD (R3 & R2), R1, R0
mask = 0x2, ctr = 0x2

ADD R3, (R1 | R15), R0
mask = 0xf, ctr = 0x5

ADD (R3 & R2), (R1 & R2), R0
mask = 0x2, ctr = 0x3

ADD (R3 | R2), (R1 | R2), R0
mask = 0x2, ctr = 0x7
```

13.5.2.3 Sequential instructions in alphabetical order

13.5.2.3.1 ADD - Add

Assembler Syntax: `ADD Rs, Rn, Rd`

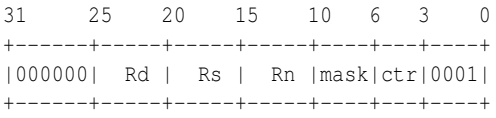
Description: The source register is added to the contents of the `Rn` register, and the result is stored in the destination register.

Rs and/or Rn can be masked with a general register, refer to section 13.5.2.2.

Operation: [31]

```
Rd = Rs + Rn;
```

Instruction Format:



[31] Rs and/or Rn can be masked.

13.5.2.3.2 ADDQ - Add Quick

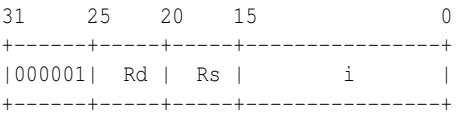
Assembler Syntax: `ADDQ Rs, i, Rd`

Description: A 16-bit immediate value is zero extended to 32 bits and added to the contents of the source register, and the result is stored in the destination register.

Operation:

```
Rd = Rs + i;
```

Instruction Format:



13.5.2.3.3 AND - Logical AND

Assembler Syntax: `AND Rs, Rn, Rd`

Description: A bitwise AND is performed between the source register and the contents of the `Rn` register. The result is stored in the destination register.

Rs and/or Rn can be masked with a general register, refer to section 13.5.2.2.

Operation: [32]

```
Rd = Rs & Rn;
```

Instruction Format:

```
31     25    20    15    10   6   3    0
+------+-----+-----+-----+----+---+----+
|000000|  Rd |  Rs |  Rn |mask|ctr|0010|
+------+-----+-----+-----+----+---+----+
```

[32] Rs and/or Rn can be masked.

13.5.2.3.4 ANDQ - Logical AND Quick

Assembler Syntax: `ANDQ Rs, i, Rd`

Description: The 16-bit immediate value is zero extended to 32 bits and then a bitwise AND is performed between it and the contents of the source register. The result is stored in the destination register.

Operation:

```
Rd = Rs & i;
```

Instruction Format:

```
31     25    20    15               0
+------+-----+-----+----------------+
|000010|  Rd |  Rs |        i       |
+------+-----+-----+----------------+
```

13.5.2.3.5 ANDQH - Logical AND Quick High

Assembler Syntax: `ANDQH Rs, i, Rd`

Description: The 16-bit immediate value is shifted left 16 steps and then a bitwise AND is performed between it and the contents of the source register. The result is stored in the destination register.

Operation:

```
Rd = ((i << 16) | 0x0000ffff) & Rs;
```

Instruction Format:

```
31     25    20    15               0
+------+-----+-----+----------------+
|000101|  Rd |  Rs |        i       |
+------+-----+-----+----------------+
```

13.5.2.3.6 BA - Branch Always

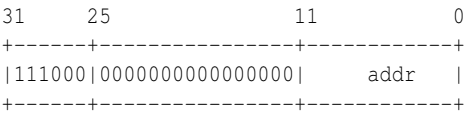
Assembler Syntax: `BA addr`

Description: The program counter (SEQ_PC) is loaded with the contents of addr. The BA instruction is a delayed branch instruction, with one delay slot.

Operation:

```
SEQ_PC = addr;
```

Instruction Format:



13.5.2.3.7 BAR - Branch Always Register

Assembler Syntax: `BAR Rs`

Description: The program counter (SEQ_PC) is loaded with the contents of the source register.[33] The BAR instruction is a delayed branch instruction, with one delay slot.

Operation:

```
SEQ_PC = Rs;
```

Instruction Format:

```
31     25    20    15               0
+------+-----+-----+----------------+
|111100|00000|  Rs |0000000000000000|
+------+-----+-----+----------------+
```

[33] Rs, the source register, can only be a general register R0 - R15.

13.5.2.3.8 BBC - Branch Bit Clear

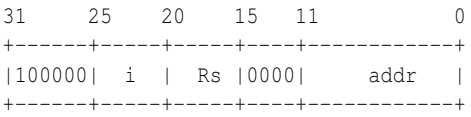
Assembler Syntax: `BBC Rs, i, addr`

Description: The program counter (SEQ_PC) is loaded with the contents of addr if the bit i of the source register[34] is zero. The BBC instruction is a delayed branch instruction, with one delay slot.

Operation:

```
if ((Rs & (1 << i)) == 0) {
  SEQ_PC = addr;
}
```

Instruction Format:



[34] Rs, the source register, can be one of R0 - R15, GOUT, GIN, STATIN, TRIGGER, WSTS or SPECS.

13.5.2.3.9 BBS - Branch Bit Set

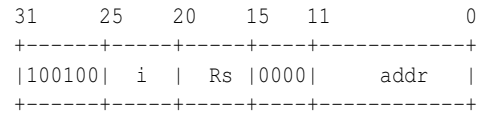
Assembler Syntax: `BBS Rs, i, addr`

Description: The program counter (SEQ_PC) is loaded with the contents of addr if the bit i of the source register[35] is set. The BBS instruction is a delayed branch instruction, with one delay slot.

Operation:

```
if ((Rs & (1 << i)) != 0) {
  SEQ_PC = addr;
}
```

Instruction Format:



[35] Rs, the source register, can be one of R0 - R15, GOUT, GIN, STATIN, TRIGGER, WSTS or SPECS.

13.5.2.3.10 BMI - Branch Minus

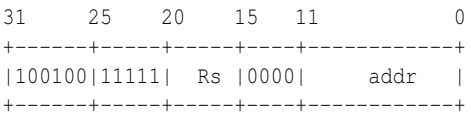
Assembler Syntax: `BMI Rs, addr`

Description: The program counter (SEQ_PC) is loaded with the contents of addr if the contents of the source register[36] is less than zero. The BMI instruction is a delayed branch instruction, with one delay slot.

Operation:

```
if ((Rs & 0x80000000) != 0) {
  SEQ_PC = addr;
}
```

Instruction Format: [37]



[36] Rs, the source register, can be one of R0 - R15, GOUT, GIN, STATIN, TRIGGER, WSTS or SPECS.
[37] BMI Rs, addr is the same as BBS Rs, 31, addr.

13.5.2.3.11 BNZ - Branch Not Zero

Assembler Syntax: `BNZ Rs, addr`

Description: The program counter (SEQ_PC) is loaded with the contents of addr if the contents of the source register[38] is not equal zero. The BNZ instruction is a delayed branch instruction, with one delay slot.

Operation:

```
if (Rs != 0) {
  SEQ_PC = addr;
}
```

Instruction Format:

```
31     25    20    15   11           0
+------+-----+-----+----+------------+
|110000|00000|  Rs |0000|     addr   |
+------+-----+-----+----+------------+
```

[38] Rs, the source register, can be one of R0 - R15, GOUT, GIN, STATIN, TRIGGER, WSTS or SPECS.

13.5.2.3.12 BPL - Branch Plus

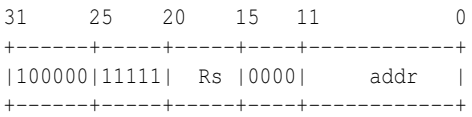
Assembler Syntax: `BPL Rs, addr`

Description: The program counter (SEQ_PC) is loaded with the contents of addr if the contents of the source register[39] is greater than or equal to zero. The BPL instruction is a delayed branch instruction, with one delay slot.

Operation:

```
if ((Rs & 0x80000000) == 0) {
  SEQ_PC = addr;
}
```

Instruction Format: [40]



[39] Rs, the source register, can be one of R0 - R15, GOUT, GIN, STATIN, TRIGGER, WSTS or SPECS.
[40] BPL Rs, addr is the same as BBC Rs, 31, addr.

13.5.2.3.13 BZ - Branch Zero

Assembler Syntax: `BZ Rs, addr`

Description: The program counter (SEQ_PC) is loaded with the contents of addr if the contents of the source register[41] equals zero. The BZ instruction is a delayed branch instruction, with one delay slot.

Operation:

```
if (Rs == 0) {
  SEQ_PC = addr;
}
```

Instruction Format:

```
31     25    20    15   11           0
+------+-----+-----+----+------------+
|110100|00000|  Rs |0000|     addr   |
+------+-----+-----+----+------------+
```

[41] Rs, the source register, can be one of R0 - R15, GOUT, GIN, STATIN, TRIGGER, WSTS or SPECS.

13.5.2.3.14 FSM - Start FSM mode

Assembler Syntax: `FSM`

Description: Change mode from SEQ to FSM. The FSM instruction is a delayed branch instruction, with one delay slot.

Operation:

```
Start FSM mode, at FSM_PC
```

Instruction Format:

```
31     25    20    15               0
+------+-----+-----+----------------+
|101000|00000|10001|0000000000000000|
+------+-----+-----+----------------+
```

13.5.2.3.15 FSMQ - Start FSM mode Quick

Assembler Syntax: `FSMQ addr`

Description: Change mode from SEQ to FSM. The FSMQ instruction is a delayed branch instruction, with one delay slot.

Operation:

```
Load PC with addr and start FSM mode
```

Instruction Format:

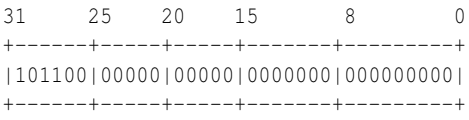
```
31     25    20    15   11           0
+------+-----+-----+----+------------+
|101010|00000|00000|0000|    addr    |
+------+-----+-----+----+------------+
```

13.5.2.3.16 HALT - Halt the SPU

Assembler Syntax: `HALT`

Description: Disables the SPU. The SPU can be re-enabled by the owner, using rw_ctrl.

Instruction Format:



13.5.2.3.17 LSL - Logical Shift Left

Assembler Syntax: `LSL Rs, Rn, Rd`

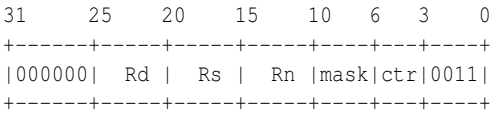
Description: The source register is left shifted the number of steps specified by the `Rn` register and zero-filled.[42]

Rs and/or Rn can be masked with a general register, refer to section 13.5.2.2.

Operation: [43]

```
Rd = Rs << Rn;
```

Instruction Format:



[42] A shift with 32 steps or more will give a zero result.
[43] Rs and/or Rn can be masked.

13.5.2.3.18 LSLQ - Logical Shift Left Quick

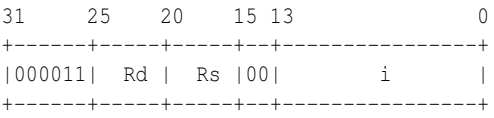
Assembler Syntax: `LSLQ Rs, i, Rd`

Description: The source register is left shifted the number of steps specified by the 14-bit immediate value and zero-filled.[44]

Operation:

```
Rd = Rs << i;
```

Instruction Format:



[44] A shift with 32 steps or more will give a zero result.

13.5.2.3.19 LSR - Logical Shift Right

Assembler Syntax: `LSR Rs, Rn, Rd`

Description: The source register is right shifted the number of steps specified by the `Rn` register and zero-filled.[45]

Rs and/or Rn can be masked with a general register, refer to section 13.5.2.2.

Operation: [46]

```
Rd = Rs >> Rn;
```

Instruction Format:

```
31     25    20    15    10   6   3    0
+------+-----+-----+-----+----+---+----+
|000000|  Rd |  Rs |  Rn |mask|ctr|0100|
+------+-----+-----+-----+----+---+----+
```

[45] A shift with 32 steps or more will give a zero result.
[46] Rs and/or Rn can be masked.

13.5.2.3.20 LSRQ - Logical Shift Right Quick

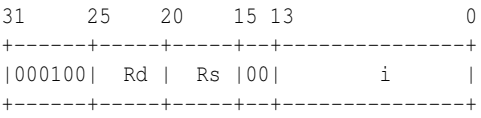
Assembler Syntax: `LSRQ Rs, i, Rd`

Description: The source register is right shifted the number of steps specified by the 14-bit immediate value and zero-filled.[47]

Operation:

```
Rd = Rs >> i;
```

Instruction Format:



[47] A shift with 32 steps or more will give a zero result.

13.5.2.3.21 MOVE - Move to Register

Assembler Syntax: `MOVE Rs, Rd`

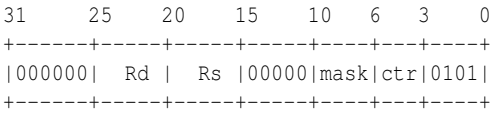
Description: Move data from source register to the destination register.

Rs can be masked with a general register, refer to section 13.5.2.2.

Operation: [48]

```
Rd = Rs;
```

Instruction Format:



[48] Rs can be masked.

13.5.2.3.22 MOVE - Move from Event Register

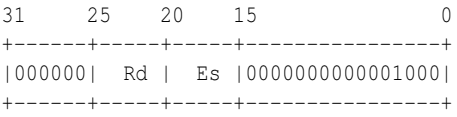
Assembler Syntax: `MOVE Es, Rd`

Description: Move data from special event source register to the destination register.

Operation:

```
Rd = Es;
```

Instruction Format:



13.5.2.3.23 MOVE - Move to Event Register

Assembler Syntax: `MOVE Rs, Ed`

Description: Move data from source register to special event destination register.

Operation:

```
Ed = Rs;
```

Instruction Format:

```
31     25    20    15               0
+------+-----+-----+----------------+
|000000|  Ed |  Rs |0000000000000111|
+------+-----+-----+----------------+
```

13.5.2.3.24 MOVEH - Move High

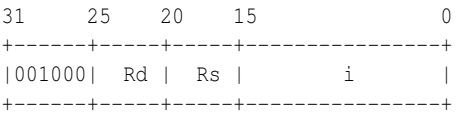
Assembler Syntax: `MOVEH Rs, i, Rd`

Description: The 16-bit immediate value is shifted left 16 steps and combined with the 16 lowest bits from source register and then stored in the destination register.

Operation:

```
Rd = (i << 16) | (0x0000ffff & Rs);
```

Instruction Format:



13.5.2.3.25 MOVEL - Move Low

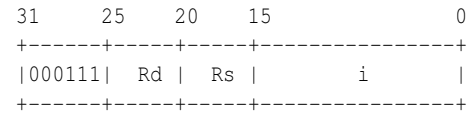
Assembler Syntax: `MOVEL Rs, i, Rd`

Description: The 16-bit immediate value is combined with the 16 highest bits from source register and then stored in the destination register.

Operation:

```
Rd = (Rs & 0xffff0000) | i;
```

Instruction Format:



13.5.2.3.26 MOVEQ - Move Quick

Assembler Syntax: `MOVEQ i, Rd`

Description: The 16-bit immediate value is zero extended to 32 bits and moved to the destination register.

Operation:

```
Rd = i;
```

Instruction Format:

```
31     25    20    15                0
+------+-----+-----+-----------------+
|000110|  Rd |00000|        i        |
+------+-----+-----+-----------------+
```

13.5.2.3.27 NOP - No Operation

Assembler Syntax: `NOP`

Description: No operation.

Operation:

```
;
```

Instruction Format:

```
31                              0
+-------------------------------+
|00000000000000000000000000000000|
+-------------------------------+
```

13.5.2.3.28 NOT - Logical Complement

Assembler Syntax: `NOT Rs, Rd`

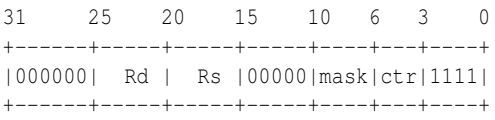
Description: The contents of the source register is bitwise inverted (1's complement). The result is stored in the destination register.

Rs can be masked with a general register, refer to section 13.5.2.2.

Operation: [49]

```
Rd = ~Rs;
```

Instruction Format:



[49] Rs can be masked.

13.5.2.3.29 OR - Logical OR

Assembler Syntax: `OR Rs, Rn, Rd`

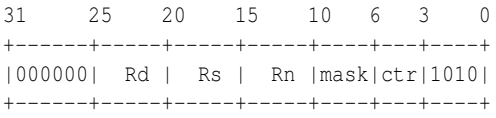
Description: A bitwise OR is performed between the source register and the contents of the `Rn` register. The result is stored in the destination register.

Rs and/or Rn can be masked with a general register, refer to section 13.5.2.2.

Operation: [50]

```
Rd = Rs | Rn;
```

Instruction Format:



[50] Rs and/or Rn can be masked.

13.5.2.3.30 ORQ - Logical OR Quick

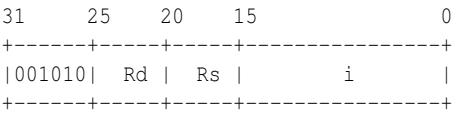
Assembler Syntax: `ORQ Rs, i, Rd`

Description: The 16-bit immediate value is zero extended to 32 bits and then a bitwise OR is performed between it and the contents of the source register. The result is stored in the destination register.

Operation:

```
Rd = Rs | i;
```

Instruction Format:



13.5.2.3.31 RR - Register Read (Address is an immediate)

Assembler Syntax: `RR regaddr, Rd`

Description: Move data from register at address regaddr, using the register interface. The data is stored in the destination register.

Operation:

```
Rd = REGIF[regaddr << 2];
```

Instruction Format:

```
31     25    20         10          0
+------+-----+----------+-----------+
|010000|  Rd |0000000000|  regaddr  |
+------+-----+----------+-----------+
```

13.5.2.3.32 RR - Register Read (Address in REGA register)

Assembler Syntax: `RR REGA, Rd`

Description: Move data from register at address stored in REGA, using the register interface. The data is stored in the destination register.

Operation:

```
Rd = REGIF[REGA];
```

Instruction Format:

```
31     25    20         10          0
+------+-----+----------+-----------+
|010010|  Rd |0000000000|00000000000|
+------+-----+----------+-----------+
```

13.5.2.3.33 RRM - Register Read with Mask (Address is an immediate)

Assembler Syntax: `RRM regaddr, Rs, Rd`

Description: Move data from register at address regaddr, using the register interface. The data is masked with the contents of `Rs` and stored in the destination register.

Operation:

```
Rd = REGIF[regaddr << 2] & Rs;
```

Instruction Format:

```
31     25    20    15    10          0
+------+-----+-----+-----+-----------+
|010001|  Rd |  Rs |00000|  regaddr  |
+------+-----+-----+-----+-----------+
```

13.5.2.3.34 RRM - Register Read with Mask (Address in REGA register)

Assembler Syntax: `RRM REGA, Rs, Rd`

Description: Move data from register at address stored in REGA, using the register interface. The data is masked with the contents of `Rs` and stored in the destination register.

Operation:

```
Rd = REGIF[REGA] & Rs;
```

Instruction Format:

```
31     25    20    15               0
+------+-----+-----+----------------+
|010011|  Rd |  Rs |0000000000000000|
+------+-----+-----+----------------+
```

13.5.2.3.35 RRMH - Register Read with Mask High (Address is an immediate)

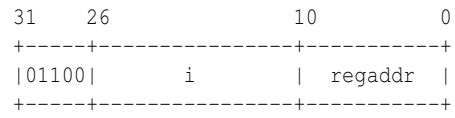
Assembler Syntax: `RRMH regaddr, i`

Description: Read data from register at address regaddr, using the register interface. The data is then masked with the 16-bit immediate which is shifted left 16 times and zero-filled. The RRMH instruction has no destination. The result of the operation only affects the z-flag.

Operation:

```
z-flag = ((REGIF[regaddr << 2] & (i << 16)) == 0);
```

Instruction Format:



13.5.2.3.36 RRMH - Register Read with Mask High (Address in REGA register)

Assembler Syntax: `RRMH REGA, i`

Description: Read data from register at address stored in REGA, using the register interface. The data is then masked with the 16-bit immediate which is shifted left 16 times and zero-filled. The RRMH instruction has no destination. The result of the operation only affects the z-flag.

Operation:

```
z-flag = ((REGIF[REGA] & (i << 16)) == 0);
```

Instruction Format:

```
31    26               10          0
+-----+----------------+-----------+
|01101|       i        |00000000000|
+-----+----------------+-----------+
```

13.5.2.3.37 RRMQ - Register Read with Mask Quick (Address is an immediate)

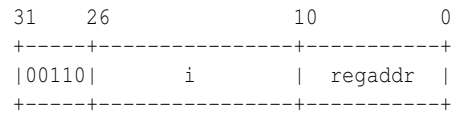
Assembler Syntax: `RRMQ regaddr, i`

Description: Read data from register at address regaddr, using the register interface. The data is then masked with the 16-bit immediate value which is zero extended to 32 bits. The RRMQ instruction has no destination. The result of the operation only affects the z-flag.

Operation:

```
z-flag = ((REGIF[regaddr << 2] & i) == 0);
```

Instruction Format:



13.5.2.3.38 RRMQ - Register Read with Mask Quick (Address in REGA register)

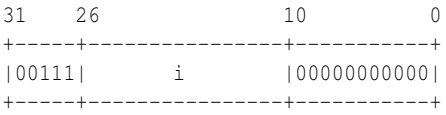
Assembler Syntax: `RRMQ REGA, i`

Description: Read data from register at address stored in REGA, using the register interface. The data is then masked with the 16-bit immediate value which is zero extended to 32 bits. The RRMQ instruction has no destination. The result of the operation only affects the z-flag.

Operation:

```
z-flag = ((REGIF[REGA] & i) == 0);
```

Instruction Format:



13.5.2.3.39 RW - Register Write (Address is an immediate)

Assembler Syntax: `RW Rs, regaddr`

Description: Move data from the source register using the register interface. The data is stored in the register at address regaddr.

Operation:

```
REGIF[regaddr << 2] = Rs;
```

Instruction Format:

```
31     25    20    15    10          0
+------+-----+-----+-----+-----------+
|010100|00000|  Rs |00000|  regaddr  |
+------+-----+-----+-----+-----------+
```

13.5.2.3.40 RW - Register Write (Address in REGA register)

Assembler Syntax: `RW Rs, REGA`

Description: Move data from the source register using the register interface. The data is stored in the register at address stored in REGA.

Operation:

```
REGIF[REGA] = Rs;
```

Instruction Format:

```
31     25    20    15    10          0
+------+-----+-----+-----+-----------+
|010110|00000|  Rs |00000|00000000000|
+------+-----+-----+-----+-----------+
```

13.5.2.3.41 RWQ - Register Write Quick (Address is an immediate)

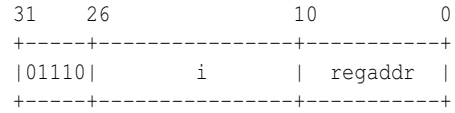
Assembler Syntax: `RWQ i, regaddr`

Description: The 16-bit immediate value is extended to 32 bit with the 16 upper bits from special register REGA and then moved to the register at address regaddr using the register interface.

Operation:

```
REGIF[regaddr << 2] = REGA & 0xffff0000 | i;
```

Instruction Format:



13.5.2.3.42 RWQ - Register Write Quick (Address in REGA register)

Assembler Syntax: `RWQ i, REGA`

Description: regaddr from REGA)

The 16-bit immediate value is extended to 32 bit with the 16 upper bits from special register REGA and then moved to the register at address regaddr using the register interface.

Operation:

```
REGIF[REGA] = REGA & 0xffff0000 | i;
```

Instruction Format:

```
31    26               10          0
+-----+----------------+-----------+
|01111|        i       |00000000000|
+-----+----------------+-----------+
```

13.5.2.3.43 SSL - Set Shift Left

Assembler Syntax: `SSL Rs, Rn, Rd`

Description: The source register is left shifted the number of steps specified by the `Rn` register and the lowest bit is set to one.[51]

Rs and/or Rn can be masked with a general register, refer to section 13.5.2.2.

Operation: [52]

```
Rd = (Rs << Rn) | 0x00000001;
```

Instruction Format:

```
31     25    20    15    10   6   3    0
+------+-----+-----+-----+----+---+----+
|000000|  Rd |  Rs |  Rn |mask|ctr|1101|
+------+-----+-----+-----+----+---+----+
```

[51] A shift with 32 steps or more will give 0x00000001 as result.
[52] Rs and/or Rn can be masked.

13.5.2.3.44 SSLQ - Set Shift Left Quick

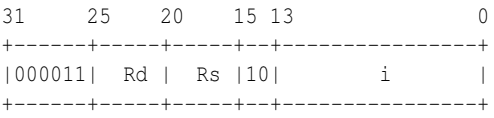
Assembler Syntax: `SSLQ Rs, i, Rd`

Description: The source register is left shifted the number of steps specified by the 14-bit immediate value and the lowest bit is set to one.[53]

Operation:

```
Rd = (Rs << i) | 0x00000001;
```

Instruction Format:



[53] A shift with 32 steps or more will give 0x00000001 as result.

13.5.2.3.45 SSR - Set Shift Right

Assembler Syntax: `SSR Rs, Rn, Rd`

Description: The source register is right shifted the number of steps specified by the `Rn` register and the highest bit is set to one.[54]

Rs and/or Rn can be masked with a general register, refer to section 13.5.2.2.

Operation: [55]

```
Rd = (Rs >> Rn) | 0x80000000;
```

Instruction Format:

```
31     25    20    15    10   6   3    0
+------+-----+-----+-----+----+---+----+
|000000|  Rd |  Rs |  Rn |mask|ctr|1110|
+------+-----+-----+-----+----+---+----+
```

[54] A shift with 32 steps or more will give 0x80000000 as result.
[55] Rs and/or Rn can be masked.

13.5.2.3.46 SSRQ - Set Shift Right Quick

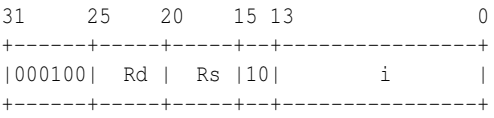
Assembler Syntax: `SSRQ Rs, i, Rd`

Description: The source register is right shifted the number of steps specified by the 14-bit immediate value and the highest bit is set to one.[56]

Operation:

```
Rd = (Rs >> i) | 0x80000000;
```

Instruction Format:



[56] A shift with 32 steps or more will give 0x80000000 as result.

13.5.2.3.47 SUB - Subtract

Assembler Syntax: `SUB Rs, Rn, Rd`

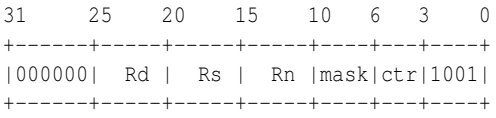
Description: The `Rn` register is subtracted from the contents of the source register, and the result is stored in the destination register.

Rs and/or Rn can be masked with a general register, refer to section 13.5.2.2.

Operation: [57]

```
Rd = Rs - Rn;
```

Instruction Format:



[57] Rs and/or Rn can be masked.

13.5.2.3.48 SUBQ - Subtract Quick

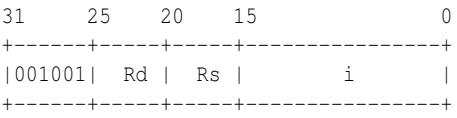
Assembler Syntax: `SUBQ Rs, i, Rd`

Description: A 16-bit immediate value is zero extended to 32 bits and then subtracted from the contents of the source register. The result is stored in the destination register.

Operation:

```
Rd = Rs - i;
```

Instruction Format:



13.5.2.3.49 SWAP - Swap

Assembler Syntax: `SWAP Rs, Rd`

Description: The source register is bitwise swapped i.e., highest bit in the destination register is the lowest bit from the source register etc. The result is stored in the destination register.

Rs can be masked with a general register, refer to section 13.5.2.2.

Operation: [58]

```
Rd[31] = Rs[0] ... Rd[0] = Rs[31]
```

Instruction Format:

```
31     25    20    15 13             0
+------+-----+-----+--+--------------+
|000100|  Rd |  Rs |01|00000000000000|
+------+-----+-----+--+--------------+
```

[58] Rs can be masked.

13.5.2.3.50 SWSRQ - Swap and Shift Right Quick

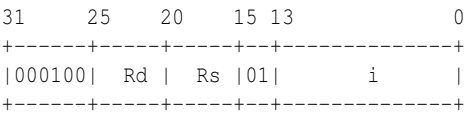
Assembler Syntax: `SWSRQ Rs, i, Rd`

Description: The source register is bitwise swapped i.e., highest bit in the destination register is the lowest bit from the source register etc. The bitwise swapped result is then right shifted the number of steps specified by the 14-bit immediate value and zero-filled.[59]

Operation:

```
SWAP (Rd[31] = Rs[0] ... Rd[0] = Rs[31]) and then shift right
```

Instruction Format:



[59] A shift with 32 steps or more will give a zero result.

13.5.2.3.51 XOR - Logical Exclusive OR

Assembler Syntax: `XOR Rs, Rn, Rd`

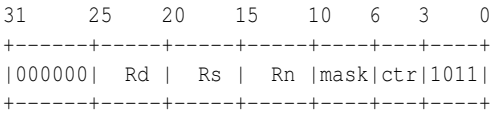
Description: A bitwise XOR is performed between the source register and the contents of the `Rn` register. The result is stored in the destination register.

Rs and/or Rn can be masked with a general register, refer to section 13.5.2.2.

Operation: [60]

```
Rd = Rs ^ Rn;
```

Instruction Format:



[60] Rs and/or Rn can be masked.

13.5.2.3.52 XOR - Register Exclusive OR

Assembler Syntax: `XOR Rs, Rd`

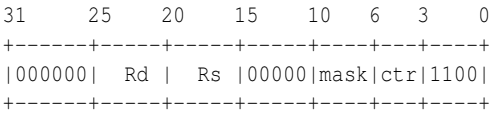
Description: An XOR is performed between all bits of the source register. The one bit result is stored in the destination register.

Rs can be masked with a general register, refer to section 13.5.2.2.

Operation: [61]

```
Rd = Rs[0] ^ Rs[1] ... ^ Rs[31];
```

Instruction Format:



[61] Rs can be masked.

13.5.2.3.53 XORQ - Logical Exclusive OR Quick

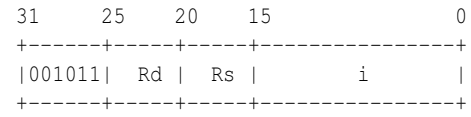
Assembler Syntax: `XORQ Rs, i, Rd`

Description: The 16-bit immediate value is zero extended to 32 bits and then XOR is performed between it and the contents of the source register. The result is stored in the destination register.

Operation:

```
Rd = Rs ^ i;
```

Instruction Format:



13.5.2.4 FSM instructions

All states have an FSM_INSTR used to describe the properties of the state.

Figure 13.5: *SPU, FSM instruction format*

Bit(s)	Field	Description
31	go_seq	Changes the mode from FSM to sequential. If the state has a sequential instruction it will be executed.
30	seq	This state has a sequential instruction.
29	do_seq	If set the sequential instruction will be performed every 200 MHz cycle while the FSM is in this state. Otherwise (if not set) the sequential instruction is performed once, when the state is entered.
28	timer	This state has a timer instruction.[62]
27-24	nbr_trans	This state has nbr_trans (0-8) transition instruction(s). [62]
23	seq_only	If set the FSM will jump directly to next_state, specified in table 13.33. If both go_seq and seq_only are set the address is a sequential mode address. A state with seq_only bit set can only have a sequential instruction. No TRANS_INSTR or TIMER_INSTR can be used.
22	halt	Used to disable the SPU when entering this state.
21	addr_hi	Highest address bit, bit[8], used for all next_state addresses. [63]
20	do_event_upd	If set the selected event values are updated every 200 MHz cycle while the FSM is in this state. Otherwise (if not set) the selected event values are updated when the state is entered.
19-16	event_upd_mask	Update enable mask for the events, if the corresponding bit is set the event is updated. [64] Field do_event_upd controls how the update will be performed.
15-12	event_mask	Enable mask for events in this state. If the corresponding bit is set the event is enabled. [65]
11-8	sel_inputs	Selects the four inputs, fsm_inp[3:0] from inputs selected by rw_fsm_inputs7_4 and rw_fsm_inputs3_0 registers, see table 13.34. If seq_only is set the sel_inputs field is used to select next_state address, see table below.
7-0	sel_outputs	Selects four FSM outputs. The FSM can only affect spu_gio_out[7:0], named o7 to o0. See section 13.5.2.4.2. If seq_only is set the sel_outputs field is used to select next_state address, see table 13.33.

Table 13.32: *SPU, FSM instruction*

[62] If a 32-bit immediate sequential instruction is used (e.g., ADDX) bit 28 and 27 must be set. In this case the TIMER_INSTR is used as immediate value and nbr_trans only uses bits[26:24].

[63] All FSM and timer transitions from this state must go to the same half of the memory.

[64] [19]=event3, [18]=event2, [17]=event1, [16]=event0.

[65] [15]=event3, [14]=event2, [13]=event1, [12]=event0.

Some of the above instruction fields may have other functions depending on some fields.

Description	go_seq	seq_only	sel_inputs				sel_outputs							
			[3]	[2]	[1]	[0]	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Normal mode.	0	0	sel_inputs				sel_outputs							
Only mode[66], FSM address is an immediate	0	1	0	–	fsm_address (byte address >> 3)									
Only mode [66], FSM address is a register	0	1	1	–			next_state address source							
Change to seq. mode, seq_pc is used	1	0	–				–							
Change to seq. mode, seq address is an immediate	1	1	–	seq_address (byte address)										

Table 13.33: *SPU, FSM mode instruction sel_inputs and sel_outputs fields*

13.5.2.4.1 sel_inputs description

sel_inputs [3:0]	inp[3]	inp[2]	inp[1]	inp[0]
0000	fsm_in3	fsm_in2	fsm_in1	fsm_in0
0001	fsm_in5	fsm_in4	fsm_in1	fsm_in0
0010	fsm_in7	fsm_in6	fsm_in1	fsm_in0
0011	fsm_in5	fsm_in4	fsm_in3	fsm_in2
0100	fsm_in7	fsm_in6	fsm_in3	fsm_in2
0101	fsm_in7	fsm_in6	fsm_in5	fsm_in4
0110	fsm_in4	fsm_in2	fsm_in1	fsm_in0
0111	fsm_in5	fsm_in2	fsm_in1	fsm_in0
1000	fsm_in6	fsm_in2	fsm_in1	fsm_in0
1001	fsm_in7	fsm_in2	fsm_in1	fsm_in0
1010	fsm_in4	fsm_in3	fsm_in2	fsm_in0
1011	fsm_in5	fsm_in3	fsm_in2	fsm_in0
1100	fsm_in6	fsm_in3	fsm_in2	fsm_in0
1101	fsm_in7	fsm_in3	fsm_in2	fsm_in0
1110	z-flag	fsm_in2	fsm_in1	fsm_in0
1111	z-flag	fsm_in6	fsm_in5	fsm_in7

Table 13.34: *SPU, FSM mode, instruction field sel_inputs*

13.5.2.4.2 sel_outputs description

The sel_outputs field selects four (4) FSM outputs. The FSM can only affect spu_gio_out[7:0], referred to as o7 to o0. Table 13.35 to 13.38 list how the outputs are selected. Table 13.39 lists four exceptions which can be used to cover output signal combinations not supported in table 13.35 to 13.38.

sel_outputs[7:6]	00	01	10	11

[66] A state with only one transition which is coded in the FSM instruction. The FSM will jump directly to the next_state.

fsm_outp[3]	o2	o3	o6	o7

Table 13.35: *SPU, FSM mode, instruction field sel_outputs[7:6]*

sel_outputs[5:4]	00	01	10	11
fsm_outp[2]	o2	o3	o4	o5

Table 13.36: *SPU, FSM mode, instruction field sel_outputs[5:4]*

sel_outputs[3:2]	00	01	10	11
fsm_outp[1]	o0	o1	o6	o7

Table 13.37: *SPU, FSM mode, instruction field sel_outputs[3:2]*

sel_outputs[1:0]	00	01	10	11
fsm_outp[0]	o0	o1	o4	o5

Table 13.38: *SPU, FSM mode, instruction field sel_outputs[1:0]*

To support combinations not available in the above configuration, the following special sel_outputs listed in the table below can be used.

sel_outputs	0xe0	0xc5	0x53	0x08
fsm_outp[3]	o5	o7	o3	o3
fsm_outp[2]	o4	o6	o2	o2
fsm_outp[1]	o1	o1	o4	o6
fsm_outp[0]	o0	o0	o5	o7

Table 13.39: *SPU, FSM mode, special sel_outputs values*

Note that the sel_outputs field holds the next_state address if seq_only is used and bit[11] is set to 0. If seq_only and bit[11] both are set bit[4:0] selects next_state address.

Bit[4:0]	next_state address source
0 - 15	R0 - R15
16	EADDR, rw_event_ret.addr

13.5.2.4.3 Sequential instruction

Each FSM state can execute one sequential instruction. This is done if the seq bit (bit 30 in FSM_INSTR) is set. If the do_seq bit (bit 29 in FSM_INSTR) is set to one, the SEQ_INSTR will be performed every 5 ns until a jump to another state takes place. Otherwise, (if not set) the SEQ_INSTR will be performed once when entering the state. A 32-bit immediate instruction can be performed (e.g., ADDX, SUBX, MOVEX) by using the bits for the timer instruction, and setting bit 28 and 27 in FSM_INSTR.

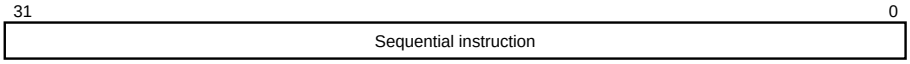


Figure 13.6: *SPU, FSM mode, SEQ instruction format*

13.5.2.4.4 Timer instruction

The timer bit (bit 28) is used to add a state transition controlled by an internal SPU timer. There can only be one TIMER_INSTR in each state.

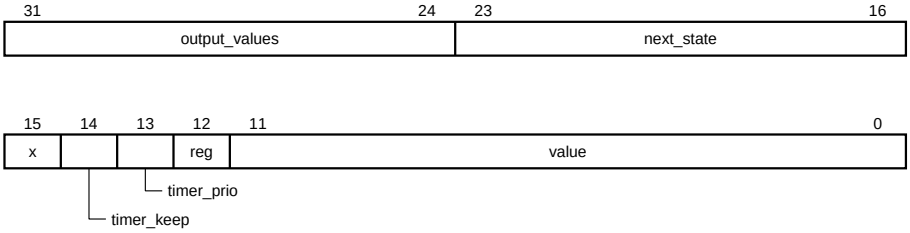


Figure 13.7: *SPU, FSM mode, TIMER_INSTR format*

Bit(s)	Field	Description
31-24	output_values	See the list below.
23-16	next_state	The state address for this state transition.[67]
15		Not used
14	timer_keep	Keep the timer count from previous state.[68]
13	timer_prio	Select if the timer_instr shall have priority over a transition instruction.
12	reg	If set, the four lowest bits in the value field are used to select a general SPU register and load the timer with the selected register value.
11-0	value	Immediate timer value (or register select if reg = 1)

Table 13.41: *SPU, FSM mode, TIMER_INSTR*

The output_values field in TIMER_INSTR is set up as follows:

Bit [31:30] is output action for fsm_outp[3]

Bit [29:28] is output action for fsm_outp[2]

Bit [27:26] is output action for fsm_outp[1]

Bit [25:24] is output action for fsm_outp[0]

The output action values are listed below (for each output):

00 No change (do nothing)

01 No change (do nothing)

10 Set to 0

11 Set to 1

[67] 64-bit aligned absolute address (byte address >> 3). The highest bit in the address is taken from the addr_hi field in the FSM_INSTR.

[68] Only works if the previous state had a timer instruction.

13.5.2.4.5 Transition instruction

All state transitions (except for the ones generated by TIMER_INSTR) are described with a TRANS_INSTR.

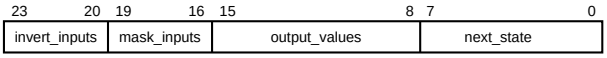


Figure 13.8: *SPU, FSM mode, TRANS_INSTR format*

Bit	Field	Description
23-20	invert_inputs	For invert input bits that are set, the state transition will occur when the corresponding inp[x]=0. For the invert input bits that are not set, the state transition occurs when the corresponding inp[x]=1. See the invert inputs list below.
19-16	mask_inputs	Set a mask input bit if the corresponding inp shall be used to trigger the next state transition. See the mask inputs list below.
15-8	output_values	See the output values list below.
7-0	next_state	The address of the next state transition. [69]

Table 13.42: *SPU, FSM mode, TRANS_INSTR*

- Invert inputs:
  - If bit 23 is set, the transition will occur when inp[3] == 0.
  - If bit 22 is set, the transition will occur when inp[2] == 0.
  - If bit 21 is set, the transition will occur when inp[1] == 0.
  - If bit 20 is set, the transition will occur when inp[0] == 0.

- Mask inputs:
  - Set bit 19 if inp[3] shall be used to trigger the next state transition.
  - Set bit 18 if inp[2] shall be used to trigger the next state transition.
  - Set bit 17 if inp[1] shall be used to trigger the next state transition.
  - Set bit 16 if inp[0] shall be used to trigger the next state transition.

  See the following example:

```
Go to next_state if inp[3] and !inp[1] (inp[2] and inp[0] == don't care)
invert inputs = 0x2
mask inputs   = 0xa
```

- Output values:

  The output_values field in TRANS_INSTR is set up as follows:

[69] 64-bit aligned absolute address (byte address >> 3). The highest bit in the address is taken from the addr_hi field in the FSM_INSTR.

Bit [15:14] is output action for fsm_outp[3]

Bit [13:12] is output action for fsm_outp[2]

Bit [11:10] is output action for fsm_outp[1]

Bit [9:8] is output action for fsm_outp[0]

The output action values are listed below (for each output):

00 No change (do nothing)

01 No change (do nothing)

10 Set to 0

11 Set to 1

13.5.2.4.6 FSM Instructions, memory use

This section shows how FSM code is placed in memory. All FSM states consist of one FSM instruction (referred to as FSM_x where x stands for [FSM_1:FSM_8]).

Each state can also contain one sequential instruction (named SEQ). State transitions from an FSM state are referred to as T_x where x stands for [T_1:T_8]. There can, however, be one timer controlled transition in each state (named TIMER).

- **Example:**

```
FSM_1:  ?_0_0_?    : ?_?_?_? : FSM_2 (T_1)

FSM_2:  seq rw R7, $080 // Put address into mc
        ?_0_0_?    : ?_?_?_? : FSM_1 (T_1)
        ?_1_0_?    : ?_?_?_? : FSM_3 (T_2)

FSM_3:  timer R3   : 0_?_?_1 : FSM_4 (TIMER)

FSM_4:  seq add R1, R3, R5
        ?_0_0_0    : 0_0_0_1 : FSM_1 (T_1)
        ?_0_0_1    : 1_0_0_1 : FSM_2 (T_2)
        ?_0_1_0    : 0_1_0_1 : FSM_3 (T_3)
        ?_0_1_1    : 0_0_0_1 : FSM_4 (T_4)
        ?_1_0_0    : 1_0_0_1 : FSM_5 (T_5)
        ?_1_0_1    : 0_0_0_1 : FSM_6 (T_6)
        ?_1_1_0    : 0_1_0_1 : FSM_1 (T_7)
        ?_1_1_1    : 1_0_0_1 : FSM_2 (T_8)

FSM_5:  timer $12  : 0_?_?_? : FSM_1 (TIMER)
        1_?_0_0    : 1_0_0_1 : FSM_4 (T_1)
        0_1_?_1    : 0_0_0_1 : FSM_7 (T_2)

FSM_6:  ?_0_0_0    : 0_0_0_1 : FSM_1 (T_1)
        ?_0_0_1    : 1_0_0_1 : FSM_2 (T_2)
        ?_0_1_0    : 0_1_0_1 : FSM_3 (T_3)
        ?_0_1_1    : 0_0_0_1 : FSM_4 (T_4)
```

```
FSM_7:  seq or R5, R4, R2
        timer R1    : ?_?_?_? : FSM_8 (TIMER)
        ?_1_0_1     : 1_0_0_1 : FSM_1 (T_1)
        ?_0_0_1     : 1_0_0_1 : FSM_2 (T_2)
        ?_0_1_0     : 0_1_0_1 : FSM_3 (T_3)
        ?_0_1_1     : 0_0_0_1 : FSM_4 (T_4)

FSM_8:  ?_1_0_1     : 1_0_0_1 : FSM_1 (T_1)
```

FSM_PC

255	192	191	128	127	64	63	0	
FSM_3	TIMER	FSM_2	SEQ	T_1	T_2	x	x	FSM_1	T_1	x	An
FSM_4	SEQ	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	An + 32
FSM_6	T_1	T_2	T_3	T_4	FSM_5	TIMER	T_1	T_2	x	x	An + 64
FSM_8	T_1	x	FSM_7	SEQ	TIMER	T_1	T_2	T_3	T_4	An + 96

Note: An (x) indicates that the byte is unused.

Figure 13.9: *SPU, Example of FSM Code in memory*

13.6 Memory Controller

The I/O Processor Memory Controller (MC) handles all memory operations between the system memory and the I/O Processor. The MC is also used when writing to the SPU memories. As seen in figure 13.10, the ETRAX FS I/O Processor contains three internal memories, one for each MPU or SPU instance. The MC can be controlled by the CPU, MPU or an SPU.

As mentioned before the MC is the only unit that can write to the I/O Processor SPU memories. The SPU memories can be read (used for instructions only, not data) by respective SPU connected to it. The address to the memories is set from the SPU connected to it. The MPU memory is entirely controlled by the LW, SW, SWQ, SWX instructions in the MPU.

13.6.1 Functional description

The MC is controlled by registers located in its respective CPU, MPU or SPU switch register bank. See [IOP_SW_CPU_REGS], [IOP_SW_MPU_REGS] and [IOP_SW_SPU_REGS].

MC registers are described below:

rw_mc_ctrl Used to request ownership of MC and to select memory operation.

rw_mc_data 32 bit wide register for data to be written to system memory.

rw_mc_addr Set system memory read/write address [70]. A write to this register actually starts and performs the system memory access.

[70] Due to limitations in the ETRAX FS memory arbiter the MC can not access internal mode registers.

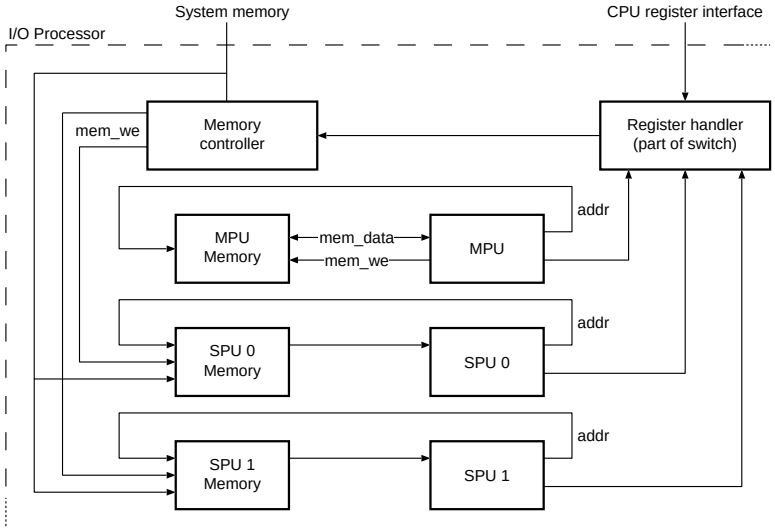


Figure 13.10: *Memory Controller location in the I/O Processor architecture*

r_mc_data and rs_mc_data 32 bit wide register for data read from system memory. When reading rs_mc_data ownership of MC is released.

r_mc_stat Holds MC status such as owner and busy bits.

13.6.1.1 Ownership

The MC is a shared resource and ownership is handled by using registers. Ownership is requested by doing a write to rw_mc_ctrl. This has to be done repeatedly until the corresponding ownership bit is set in the r_mc_stat register. When writing to rw_mc_ctrl it is possible to select whether the ownership shall last for one memory operation or until released. This is set by the keep_owner bit. After ownership is granted it is possible to use the registers rw_mc_data, rw_mc_addr, r_mc_data and rs_mc_data.[71]

13.6.1.1.1 Request ownership

The following steps are used to become owner of the MC:

1. Make a write to rw_mc_ctrl (with correct memory operation and size etc.).
2. Check r_mc_stat to see if you have become owner. If not, repeat step 1.
3. Ownership is granted.

[71] Reading rs_mc_data always releases ownership of MC.

13.6.1.2 Write data from system memory to I/O Processor SPU memory

To copy data from system memory to I/O Processor SPU memory/memories following steps must be taken:

1. Get MC ownership. (When writing rw_mc_ctrl use size = 4, [72] cmd = copy, and set up the write masks to indicate to which I/O Processor memory/memories the write will be performed).
2. Disable the SPU/SPUs connected to the selected memory/memories using rw_ctrl.en in the SPU.
3. Set I/O Processor memory address, using rw_seq_pc register in the SPU. All addresses are byte addresses.
4. Set system memory address in the rw_mc_addr register.[73]
5. Wait for busy_cpu, busy_mpu, busy_spu0 or busy_spu1 bit to go low (for the correct owner) in r_mc_stat.
6. If rw_mc_ctrl.keep_owner is set to no the ownership is released. Otherwise rs_mc_data can be used to release ownership.

13.6.1.3 Read data from system memory to the r_mc_data register

To read data from system memory to the r_mc_data or rs_mc_data register inside the Memory Controller the following steps must be taken:

1. Get MC ownership. (With rw_mc_ctrl.cmd = rd) [74]
2. Set system memory address in the rw_mc_addr register. [73]
3. Wait for busy_cpu, busy_mpu, busy_spu0 or busy_spu1 bit to go low (for the correct owner) in r_mc_stat.
4. Read r_mc_data or rs_mc_data (depending on if you will release ownership).

13.6.1.4 Write data from the rw_mc_data register to system memory

To write data to system memory from the rw_mc_data register inside the Memory Controller the following steps must be taken:

1. Get MC ownership. (With rw_mc_ctrl.cmd = wr) [74]
2. Put data in the rw_mc_data register.
3. Set system memory address in the rw_mc_addr register. [73]

[72] Size must be 4 bytes.
[73] The data must not cross a 256 bit boundary in system memory.
[74] Size can not be more than 4 bytes (32 bits) since the r_mc_data register is 4 bytes wide.

4. Wait for busy_cpu, busy_mpu, busy_spu0 or busy_spu1 bit to go low (for the correct owner) in r_mc_stat.

5. If rw_mc_ctrl.keep_owner is set to no the ownership is released. Otherwise rs_mc_data can be used to release ownership.

13.6.1.5 Write data from the rw_mc_data register to I/O Processor SPU memory

To copy data from the rw_mc_data register to the I/O Processor SPU memory/memories following steps must be taken:

1. Get MC ownership. (When writing rw_mc_ctrl use size = 4, [72] cmd = reg_copy, and set up the write masks to indicate to which I/O Processor memory/memories the write will be performed).

2. Disable the SPU/SPUs connected to the selected memory/memories using rw_ctrl.en in the SPU.

3. Set I/O Processor memory address, using rw_seq_pc register in the SPU. All addresses are byte addresses.

4. Write data to the rw_mc_data register.

5. Wait for busy_cpu, busy_mpu, busy_spu0 or busy_spu1 bit to go low (for the correct owner) in r_mc_stat.

6. If rw_mc_ctrl.keep_owner is set to no the ownership is released. Otherwise rs_mc_data can be used to release ownership.

13.7 Switch

The main objective of the Switch is to connect the various I/O Processor modules with each other. The connections are defined by registers inside the Switch. The Switch has five internal register banks, one for each Processing Unit (CPU, MPU, SPU0 and SPU1) and one for configuration of the Switch. Please see 25.27, 25.22, 25.29 and 25.26 for a complete list and explanation of these registers. The SW_CFG_REGS register bank can be owned by any of the CPU/MPU/SPU0/SPU1[75]. The Switch ownership is controlled by the rw_sw_cfg_owner register in 25.22.

13.7.1 Functional description

In addition to configuring how to connect the I/O Processor modules with each other, the Switch also has other functions. The list below shows the different Switch functions.

[75] After reset the CPU is the owner of the SW_CFG_REGS register bank.

- I/O Processor module connections

  - Creating data and control paths. Controlled by 25.26.

  - Register access within the I/O Processor.

  - Handling the register interface [REGIF] connected to the CPU.

  - Handling ownership of all the other I/O Processor modules. Controlled by 25.26.

- Interrupt multiplexing to CPU

  - All hardware interrupt signals from the different I/O Processor modules can be used as CPU interrupts.

  - The MPU, SPU0 and SPU1 can generate software interrupts to the CPU via their registers in the Switch. 25.22 and 25.29.

  - The CPU uses registers in 25.27 to acknowledge MPU and SPU generated interrupts.

- Interrupt multiplexing to MPU

  - All hardware interrupt signals from the different I/O Processor modules can be combined into 16 MPU interrupts.

  - The SPU0 and SPU1 can generate software interrupts to the MPU via their registers in the Switch. 25.29.

  - The MPU uses registers in 25.22 to acknowledge SPU generated interrupts.

- Pin multiplexing, creating internal buses

  - The I/O Processor has 72 I/O-pins. From these three internal buses are created, two 32 bit wide buses (BUS0 and BUS1) and one 32 bit wide general I/O bus (GIO). Controlled in 25.26.

13.7.1.1 Register access

All I/O Processor modules are controlled by a Processing Unit (e.g., CPU, MPU or SPU). The Processing Unit (PU) in control of an I/O Processor module is called the owner of the module. Ownership is defined by the owner of the Switch in 25.26.

It is the MPU (or the CPU via the MPU) that controls the owner of the Switch using the rw_sw_cfg_owner register. The CPU is the only unit that can be the owner of the MPU. For all other I/O Processor modules the owner can be: CPU, MPU, SPU0 or SPU1. The default owner for all modules is the CPU.

13.7.1.2 Interrupts to CPU from the I/O Processor

There are four interrupt signals from the I/O Processor to the CPU.

In the I/O Processor there are 24 interrupt generating modules which can be controlled by the CPU.

Modules	Reference
4 Timer Groups	tg[3:0]
8 Trigger Groups	trig[7:0]
2 out FIFOs (general register bank)	fifo0[out], fifo1[out]
2 in FIFOs (general register bank)	fifo0[in], fifo1[in]
2 out FIFOs (extra register bank)	fifo_xtra0[out], fifo_xtra1[out]
2 in FIFOs (extra register bank)	fifo_xtra0[in], fifo_xtra1[in]
2 in DMCs	dmc0[in], dmc1[in]
2 out DMCs	dmc0[out], dmc1[out]

Table 13.43: *Switch, CPU interrupt sources*

All of the above modules have interrupt logic including mask and acknowledge registers. Refer to documentation on each I/O Processor module.

In addition to the 24 hardware interrupt sources there are a number of software generated interrupts (from the MPU and SPUs):

- 32 CPU interrupts from MPU software, referred to as mpu[31:0] in table below.
- 16 CPU interrupts from SPU0 software, referred to as spu0[15:0] in table below.
- 16 CPU interrupts from SPU1 software, referred to as spu1[15:0] in table below.

These interrupts are selected through registers in the Switch. Each instance of the MPU and SPU has its own register for generating software interrupts to the CPU. To clear the MPU or SPU software interrupts the CPU uses the rw_ack_intr0 ... rw_ack_intr3 registers inside the Switch CPU register bank.

The different CPU interrupt vectors are created like this:

interrupt vector	interrupt sources
int0	spu1[15:8], spu0[7:0],[76] mpu[15:0] [77]
int1	spu1[7:0],[78] spu0[15:8], mpu[31:16] [79]
int2	tg[1:0], trig[7:0], fifo_xtra0[out,in], fifo0[out,in], dmc0[out,in], spu0[7:0], [76] mpu[7:0] [77]
int3	tg[3:2], trig[7:0], fifo_xtra1[out,in], fifo1[out,in], dmc1[out,in], spu1[7:0], [78] mpu[23:16] [79]

Table 13.44: *Switch, CPU interrupt vectors*

[76] Both rw_ack_intr0[23:16] and rw_ack_intr2[15:8] can clear SPU0 intr[7:0]
[77] Both rw_ack_intr0[7:0] and rw_ack_intr2[7:0] can clear MPU intr[7:0]
[78] Both rw_ack_intr1[31:24] and rw_ack_intr3[15:8] can clear SPU1 intr[7:0]
[79] Both rw_ack_intr1[7:0] and rw_ack_intr3[7:0] can clear MPU intr[23:16]

13.7.1.3 Interrupts to MPU

The MPU interrupt vector has 16 interrupts, intr[15:0]. In case of simultaneous interrupts intr0 has the highest priority and intr15 the lowest priority.

13.7.1.3.1 MPU Interrupts from I/O Processor modules

In the I/O Processor there are 24 interrupt generating modules which can be controlled by the MPU.

Modules	Reference
4 Timer Groups	tg[3:0]
8 Trigger Groups	trig[7:0]
2 out FIFOs (general register bank)	fifo0[out], fifo1[out]
2 in FIFOs (general register bank)	fifo0[in], fifo1[in]
2 out FIFOs (extra register bank)	fifo_xtra0[out], fifo_xtra1[out]
2 in FIFOs (extra register bank)	fifo_xtra0[in], fifo_xtra1[in]
2 in DMCs	dmc0[in], dmc1[in]
2 out DMCs	dmc0[out], dmc1[out]

Table 13.45: *Switch, MPU interrupt sources*

All of the above modules have interrupt logic including mask and acknowledge registers.

In addition to the 24 hardware interrupt sources there are 32 SPU software generated interrupts:

- 16 MPU interrupts from SPU0 software, named spu0[15:0] in table below.
- 16 MPU interrupts from SPU1 software, named spu1[15:0] in table below.

The 16 interrupts are divided into four groups, with four interrupts in each group. How the interrupts in each group are generated is described in the figure 13.11 below. The rw_intr_grp0_mask... rw_intr_grp3_mask registers in 25.22 register bank are used as masks for each interrupt group.

The software interrupts generated by the SPUs are connected individually to the MPU interrupts. To clear SPU software generated interrupts the MPU uses the rw_ack_intr_grp0 ... rw_ack_intr_grp3 registers in the Switch MPU register bank. The MPU can read all masked and non-masked hardware and software interrupts using the r_intr_grp0 ... r_intr_grp3 and the r_masked_intr_grp0 ... r_masked_intr_grp3 registers.

Each one of the 28 hardware interrupts is connected to four different MPU interrupts.

MPU Interrupt	Interrupt sources							
intr[0]	spu0[0]	spu1[0]	trig[0]	trig[4]	tg[0]	fifo_out[0]	fifo_outx[0]	dmc_out[0]
intr[1]	spu0[1]	spu1[1]	trig[1]	trig[5]	tg[1]	fifo_in[0]	fifo_inx[0]	dmc_in[0]

intr[2]	spu0[2]	spu1[2]	trig[2]	trig[6]	tg[2]	fifo_out[1]	fifo_outx[1]	dmc_out[1]
intr[3]	spu0[3]	spu1[3]	trig[3]	trig[7]	tg[3]	fifo_in[1]	fifo_inx[1]	dmc_in[1]

Table 13.46: *Switch, MPU interrupt vectors 0-3, group 0*

Registers for intr[3:0] are located in intr_grp0. intr0 uses bits [7:0], intr1 uses bits [15:8], intr2 uses bits [23:16] and intr3 uses bits [31:24].

MPU Interrupt	Interrupt sources							
intr[4]	spu0[4]	spu1[4]	trig[0]	trig[5]	tg[0]	fifo_in[0]	fifo_inx[0]	dmc_out[0]
intr[5]	spu0[5]	spu1[5]	trig[1]	trig[6]	tg[1]	fifo_out[1]	fifo_outx[0]	dmc_in[0]
intr[6]	spu0[6]	spu1[6]	trig[2]	trig[7]	tg[2]	fifo_in[1]	fifo_inx[1]	dmc_out[1]
intr[7]	spu0[7]	spu1[7]	trig[3]	trig[4]	tg[3]	fifo_out[0]	fifo_outx[1]	dmc_in[1]

Table 13.47: *Switch, MPU interrupt vectors 4-7, group 1*

Registers for intr[7:4] are located in intr_grp1. intr4 uses bits [7:0], intr5 uses bits [15:8], intr6 uses bits [23:16] and intr7 uses bits [31:24].

MPU Interrupt	Interrupt sources							
intr[8]	spu0[8]	spu1[8]	trig[0]	trig[6]	tg[0]	fifo_out[1]	fifo_outx[1]	dmc_out[0]
intr[9]	spu0[9]	spu1[9]	trig[1]	trig[7]	tg[1]	fifo_in[1]	fifo_inx[1]	dmc_in[0]
intr[10]	spu0[10]	spu1[10]	trig[2]	trig[4]	tg[2]	fifo_out[0]	fifo_outx[0]	dmc_out[1]
intr[11]	spu0[11]	spu1[11]	trig[3]	trig[5]	tg[3]	fifo_in[0]	fifo_inx[0]	dmc_in[1]

Table 13.48: *Switch, MPU interrupt vectors 8-11, group 2*

Registers for intr[11:8] are located in intr_grp2. intr8 uses bits [7:0], intr9 uses bits [15:8], intr10 uses bits [23:16] and intr11 uses bits [31:24].

MPU Interrupt	Interrupt sources							
intr[12]	spu0[12]	spu1[12]	trig[0]	trig[7]	tg[0]	fifo_in[1]	fifo_inx[1]	dmc_out[0]
intr[13]	spu0[13]	spu1[13]	trig[1]	trig[4]	tg[1]	fifo_out[0]	fifo_outx[0]	dmc_in[0]
intr[14]	spu0[14]	spu1[14]	trig[2]	trig[5]	tg[2]	fifo_in[0]	fifo_inx[0]	dmc_out[1]
intr[15]	spu0[15]	spu1[15]	trig[3]	trig[6]	tg[3]	fifo_out[1]	fifo_outx[1]	dmc_in[1]

Table 13.49: *Switch, MPU interrupt vectors 12-15, group 3*

Registers for intr[15:12] are located in intr_grp3. intr12 uses bits [7:0], intr13 uses bits [15:8], intr14 uses bits [23:16] and intr15 uses bits [31:24].

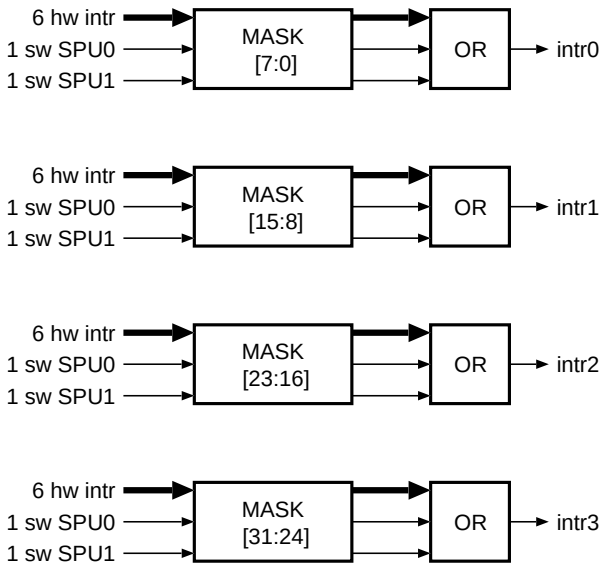


Figure 13.11: *Switch, MPU interrupt generation, group 0*

13.7.1.3.2 MPU Interrupts from CPU software

Using the ordinary sixteen MPU interrupts from CPU software is not possible. To generate a CPU software interrupt to the MPU the CPU can issue an MPU JIR instruction, using the rw_instr register. Refer to section 13.4.2.2.18 for more information.

13.7.1.4 Pin multiplexing

The ETRAX FS I/O Processor has the following structure for pins and buses:

- 72 input/output signals divided into four ports, **pb**, **pc**, **pd** and **pe**.

  Each of the ports contain 18 pins.
- From the 72 input/output signals there are internally created two 32-bit wide buses, BUS0 and BUS1, and one 32-bit wide GIO bus.
- The upper and lower halves of the BUS0 and BUS1 buses can be used as separate internal buses. The I/O Processor controls the direction of each byte of the BUS0 and BUS1 buses separately, but there is no pin by pin direction control.
- The direction of each pin in the GIO bus is individually controlled by the I/O Processor.

The GIO and buses are referred to as:

- BUS0_in[31:0], BUS1_in[31:0], GIO_in[31:0] as inputs.

- BUS0_out[31:0], BUS1_out[31:0], GIO_out[31:0] as outputs.
- BUS0_oe[3:0], BUS1_oe[3:0], (one bit per 8 bit data), GIO_oe[31:0] as output enables.

13.7.1.4.1 Mapping I/O Processor buses onto the pa to pe ports

The mapping from the pins to the three I/O Processor buses is controlled by the Switch owner using the rw_pinmapping register.

Each of the BUS0 and BUS1 buses are split into 8-bit portions, so that each can be mapped onto the **pb** to **pe** ports in two different ways. The GIO bus is split into 4-bit portions. The two mapping alternatives are denoted A and B.

BUS0 portions	Mapping A	Mapping B
BUS0[7:0]	**pb7 - pb0**	**pd7 - pd0**
BUS0[15:8]	**pc7 - pc0**	**pb15 - pb8**
BUS0[23:16]	**pd7 - pd0**	**pe15 - pe8**
BUS0[31:24]	**pe7 - pe0**	**pd15 - pd8**

Table 13.50: *Switch, BUS0 mapping*

BUS1 portions	Mapping A	Mapping B
BUS1[7:0]	**pc7 - pc0**	**pe7 - pe0**
BUS1[15:8]	**pc15 - pc8**	**pe15 - pe8**
BUS1[23:16]	**pb15 - pb8**	**pd15 - pd8**
BUS1[31:24]	**pb7 - pb0**	**pc15 - pc8**

Table 13.51: *Switch, BUS1 mapping*

GIO portions	Mapping A	Mapping B
GIO[3:0]	**pb11 - pb8**	**pd17 - pd16**, **pe17 - pe16**
GIO[7:4]	**pd11 - pd8**	**pb17 - pb16**, **pc17 - pc16**
GIO[11:8]	**pc11 - pc8**	**pe3 - pe0**
GIO[15:12]	**pe11 - pe8**	**pd3 - pd0**
GIO[19:16]	**pb15 - pb12**	**pc3 - pc0**
GIO[23:20]	**pd15 - pd12**	**pc7 - pc4**
GIO[27:24]	**pc15 - pc12**	**pe7 - pe4**
GIO[31:28]	**pe15 - pe12**	**pd7 - pd4**

Table 13.52: *Switch, GIO mapping*

The other way around, the mapping will be:

Port **pb** to **pe**	Mapping to BUS0	Mapping to BUS1	Mapping to GIO
pb7 - pb0	BUS0[7:0]A	BUS1[31:24]A	-
pb15 - pb8	BUS0[15:8]B	BUS1[23:16]A	GIO[19:16,3:0]A
pb17 - pb16	-	-	GIO[7:6]B

pc7 - pc0	BUS0[15:8]A	BUS1[7:0]A	GIO[23:16]B
pc15 - pc8	-	BUS1[15:8]A or BUS1[31:24]B	GIO[27:24,11:8]A
pc17 - pc16	-	-	GIO[5:4]B
pd7 - pd0	BUS0[7:0]B or BUS0[23:16]A	-	GIO[31:28,15:12]B
pd15 - pd8	BUS0[31:24]B	BUS1[23:16]B	GIO[23:20,7:4]A
pd17 - pd16	-	-	GIO[3:2]B
pe7 - pe0	BUS0[31:24]A	BUS1[7:0]B	GIO[27:24,11:8]B
pe15 - pe8	BUS0[23:16]B	BUS1[15:8]B	GIO[31:28,15:12]A
pe17 - pe16	-	-	GIO[1:0]B

Table 13.53: *Switch, Pin mapping*

13.7.1.4.2 Controlling I/O Processor buses

Each PU instance (CPU, MPU, SPU) will have the following registers in the Switch, see 25.27, 25.22 and 25.29.

Register	Function
r_bus0_in	Input BUS0 read register
rw_bus0_clr_mask	Clear register for BUS0 out
rw_bus0_set_mask	Set register for BUS0 out
rw_bus0_oe_clr_mask	Clear register for BUS0 output enable
rw_bus0_oe_set_mask	Set register for BUS0 output enable
r_bus1_in	Input BUS1 read register
rw_bus1_clr_mask	Clear register for BUS1 out
rw_bus1_set_mask	Set register for BUS1 out
rw_bus1_oe_clr_mask	Clear register for BUS1 output enable
rw_bus1_oe_set_mask	Set register for BUS1 output enable
r_gio_in	Input GIO read register
rw_gio_clr_mask	Clear register for GIO out
rw_gio_set_mask	Set register for GIO out
rw_gio_oe_clr_mask	Clear register for GIO output enable
rw_gio_oe_set_mask	Set register for GIO output enable

Table 13.54: *Switch, Registers for buses*

Each of the two buses, BUS0_out and BUS1_out are created as described in figure 13.12 below. The GIO_out and the BUS0_oe, BUS1_oe and GIO_oe are also configured in the same way.

The sources of the output bus are selected by rw_bus_out_cfg in 25.26.

All PUs (MPU, CPU and SPUs) have the possibility to set and clear bytes in the bus, using the rw_bus0_set_mask and rw_bus0_clr_mask registers.

The SPUs also have hi/lo registers for all set and clear registers, see 25.29. This is useful since the SPUs have limited methods for handling 32 bit immediate operations.

The Switch owner (default the CPU) can mask, (force bits of the bus to zero) by using

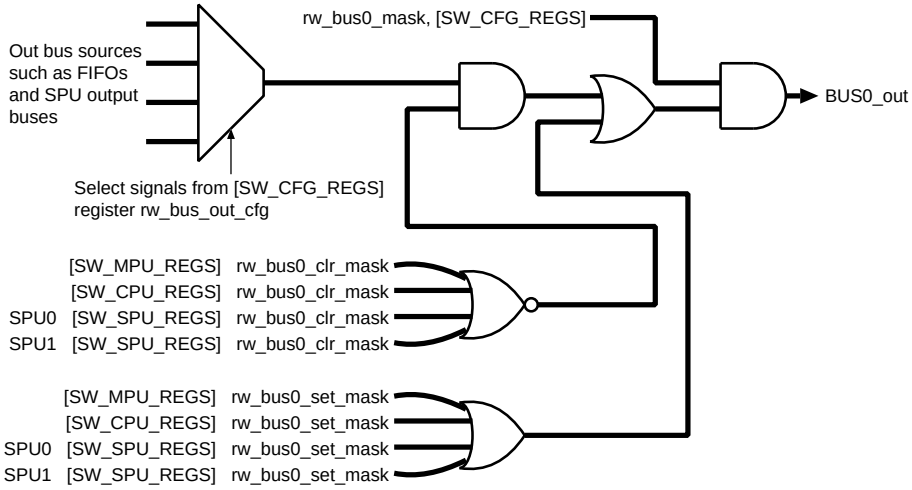


Figure 13.12: *Switch, Creating BUS0_out*

the rw_bus0_mask register. This is needed when the I/O pins are shared with other functions in ETRAX FS, see [PINMUX].

All BUS and GIO output signals are connected to the SAP_OUT module. Finally they all are combined to form the 72 output signals.

13.7.1.4.3 BUS0 out

The source for BUS0_out is controlled by register fields rw_bus_out_cfg.bus0_lo and rw_bus_out_cfg.bus0_hi. As described in the section above all PUs have separate registers for controlling the BUS0_out. BUS0_out is divided into two halves, making it more flexible.

Output enable (OE) for BUS0_out can be configured in register fields rw_bus_out_cfg.bus0_lo_oe and rw_bus_out_cfg.bus0_hi_oe, where lo refers to bits 15:0 and hi to 31:16. OE for BUS0 can be selected to come from the following sources:

- GIO out 0 from SPU0
- GIO out 1 from SPU0
- GIO out 0 from SPU1
- GIO out 1 from SPU1
- Timer group 0, timer 0
- Timer group 1, timer 0
- Timer group 2, timer 0
- Timer group 3, timer 0

13.7.1.4.4 BUS1 out

The source for BUS1_out is controlled by register fields rw_bus_out_cfg.bus1_lo and rw_bus_out_cfg.bus1_hi. As described in section 13.7.1.4.2, all PUs have separate registers for controlling the BUS1_out. BUS1_out is divided into two halves, making it more flexible.

Output enable (OE) for BUS1_out can be configured in field rw_bus_out_cfg.bus1_lo_oe and rw_bus_out_cfg.bus1_hi_oe, where lo refers to bits 15:0 and hi to 31:16. OE for BUS1 can be selected to come from the following sources:

- GIO out 1 from SPU0
- GIO out 2 from SPU0
- GIO out 1 from SPU1
- GIO out 2 from SPU1
- Timer group 0, timer 1
- Timer group 1, timer 1
- Timer group 2, timer 1
- Timer group 3, timer 1

13.7.1.4.5 GIO out bus

The sources for GIO_out and GIO_oe are controlled by the rw_gio_out_grp0_cfg ... rw_gio_out_grp7_cfg registers. Each configuration register affects four GIOs. As for the buses described above all PUs have separate registers for setting and clearing GIO outputs.

13.7.1.5 Connecting I/O Processor modules

The data and control paths in the I/O Processor are configured by registers in the Switch configuration register bank, 25.26. The following sections, beginning with 13.7.1.5.1, describe the different input configurations for each separate I/O Processor module.

13.7.1.5.1 SPU

Each SPU in the ETRAX FS I/O Processor has five 32 bit wide input buses.

Name	Description	SPU register
SPUBUS0_in	Connects the SPU to data sources	r_bus0_in
SPUBUS1_in	Connects the SPU to data sources	r_bus1_in
SPUGIO_in	Connects the SPU to input GIO signals	r_gio_in
Trigger_in	Connects the SPU to output signals from I/O Processor Trigger Group modules	r_trigger_in

Status_in	Connects the SPU to status signals from other I/O Processor modules	r_stat_in

Table 13.55: *Switch, SPU input buses*

The SPU buses SPUBUS0_in and SPUBUS1_in are configured by the rw_spu0_cfg and rw_spu1_cfg registers in 25.26. The SPUGIO_in is always mapped to synchronized GIO_in, and the SPU Trigger_in input port is also permanently mapped to the respective Trigger. Mapping of the Status_in port for each SPU is shown below.

Bits	Source	Register reference
31	MC, owned by SPU0	r_mc_stat.owned_by_spu0
30	MC, busy by SPU0	r_mc_stat.busy_spu0
29	SCRC_IN0, crc_err	r_stat.err
28	SCRC_OUT0, output	n.a.
27	sync_clk_12	n.a.
26:23	SPU1, gio_out[3:0]	n.a.
22	DMC_IN0, cmd_rdy	r_stream_stat.cmd_rdy
21	DMC_IN0, full	r_stream_stat.full
20	DMC_IN0, sth	r_stream_stat.sth
19:16	Timer grp2 strobes	n.a.
15	PCRC0, correct CRC	n.a.
14	DMC_OUT0, cmd_rdy	r_stream_stat.cmd_rdy
13	DMC_OUT0, cmd_rq	r_stream_stat.cmd_rq
12	DMC_OUT0, last	r_stream_stat.last
11	DMC_OUT0, dv	r_stream_stat.dv
10	DMC_OUT0, eop	n.a.
9	DMC_OUT0, dth	r_stream_stat.dth
8	DMC_OUT0, all	r_stream_stat.all_avail
7	FIFO_IN0, rdy	n.a.
6	FIFO_OUT0, all	n.a.
5	FIFO_OUT0, rdy	n.a.
4	FIFO_OUT0, last	n.a.
3:0	Timer grp0 strobes	n.a.

Table 13.56: *Switch, SPU0 Status_in signals*

Bits	Source	Register reference
31	MC, owned by SPU1	r_mc_stat.owned_by_spu1
30	MC, busy by SPU1	r_mc_stat.busy_spu1
29	SCRC_IN1, crc_err	r_stat.err
28	SCRC_OUT1, output	n.a.
27	sync_clk_12	n.a.
26:23	SPU0, gio_out[3:0]	n.a.
22	DMC_IN1, cmd_rdy	r_stream_stat.cmd_rdy
21	DMC_IN1, full	r_stream_stat.full
20	DMC_IN1, sth	r_stream_stat.sth
19:16	Timer grp3 strobes	n.a.
15	PCRC1, correct CRC	n.a.

14	DMC_OUT1, cmd_rdy	r_stream_stat.cmd_rdy
13	DMC_OUT1, cmd_rq	r_stream_stat.cmd_rq
12	DMC_OUT1, last	r_stream_stat.last
11	DMC_OUT1, dv	r_stream_stat.dv
10	DMC_OUT1, eop	n.a.
9	DMC_OUT1, dth	r_stream_stat.dth
8	DMC_OUT1, all	r_stream_stat.all_avail
7	FIFO_IN1, rdy	n.a.
6	FIFO_OUT1, all	n.a.
5	FIFO_OUT1, rdy	n.a.
4	FIFO_OUT1, last	n.a.
3:0	Timer grp1 strobes	n.a.

Table 13.57: *Switch, SPU1 Status_in signals*

13.7.1.5.2 Timer Groups

There are four Timer groups in the ETRAX FS I/O Processor. Each group is divided into four timers and one clock generator. Timers can be started and stopped by registers in the Timer group, Triggers or other Timers inside the same group. The tmr0_en ... tmr3_en and tmr0_dis ... tmr3_dis fields in the rw_timer_grp0_cfg ... rw_timer_grp3_cfg registers are used to select which Trigger to enable or disable each Timer.

The external clock inputs to the different Timer groups are also selected through the Switch, using the ext_clk fields of registers rw_timer_grp0_cfg ... rw_timer_grp3_cfg.

The 12 MHz clock signal connected internally in the chip is synchronized using two 200 MHz flip-flops inside the Switch. This synchronized 12 MHz clock can be used as a clock in the Timer groups. SPU GIO outputs and synchronized GIO_in 1, 3, 5 and 7 can also be used as clock for the Timer groups.

Mapping of which Trigger and SPU that can be used to each Timer group is shown in the table below.

Timer group	Enable and/or Disable with	Clock source
0	Trigger group 0 Trigger group 4	SPU0
1	Trigger group 1 Trigger group 5	SPU1
2	Trigger group 2 Trigger group 6	SPU0
3	Trigger group 3 Trigger group 7	SPU1

Table 13.58: *Switch, Timer Group clock source configuration*

13.7.1.5.3 Trigger Groups

There are eight Trigger groups in the I/O Processor. Each group is divided into four Triggers making 32 Triggers all together. Trigger 0 to Trigger 3 are located in Trigger

Group 0 and Trigger 28 to Trigger 31 are located in Trigger Group 7. All Triggers are connected to their corresponding GIO_in signal. i.e., Trigger 0 is connected to GIO_in[0].

Triggers can be enabled and disabled either by a Timer or by registers in the separate Trigger group register banks. Configuring which Timer to enable each Trigger inside each Trigger group is done in the grp0_en ... grp7_en fields of the rw_trigger_grps_cfg register. Configuring the disable source is done in the grp0_dis ... grp7_dis fields in the same register.

All Triggers in each Trigger group share the same disable and enable fields. The table below shows disable selection for Trigger group 0.

Trigger Group 0		
Trigger	grp0_dis field in register rw_trigger_grps_cfg	Disable by Timer in Timer group 0
0	timer_grp0	Timer 0
1	timer_grp0	Timer 1
2	timer_grp0	Timer 2
3	timer_grp0	Timer 3
0	timer_grp0_rot	Timer 1
1	timer_grp0_rot	Timer 2
2	timer_grp0_rot	Timer 3
3	timer_grp0_rot	Timer 0

Table 13.59: *Switch, Trigger Group, Group 0 disable selection*

13.7.1.5.4 Parallel Data Path in

The input Parallel Data Path (PDP_in) consists of three parts, DMC, FIFO and Parallel CRC checker. There are two PDP_in in the ETRAX FS I/O Processor. The connection between the parts of the PDP_in are fixed, meaning input DMC0 will always be connected to input FIFO0 etc. See figure 13.13 for a picture of PDP_in 0.

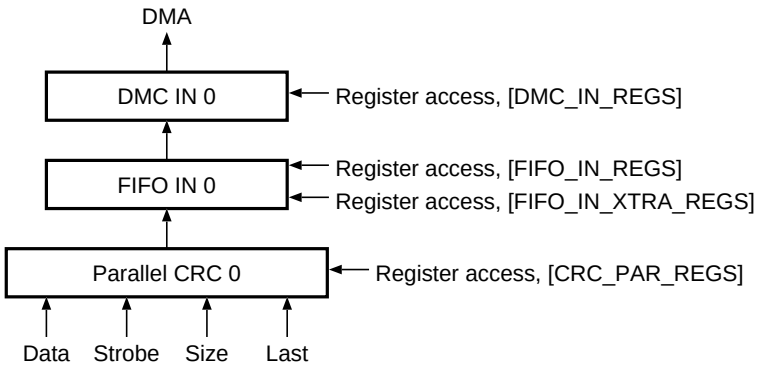


Figure 13.13: *Switch, Input Parallel Data Path 0*

As seen in figure 13.13, all parts of PDP_in have their own register banks. Data to and

from the PDP_in can be read or written using registers inside each separate part of the PDP_in.

Since the connections are fixed within the PDP_in, the configuration in the Switch is done on the input signals to the PDP_in. Below is a list of the signals that are configured:

Data Parallel data, 32, 24, 16 or 8 bits wide, strobed by strobe.

Strobe Strobe signal, controls when data is sampled.

Last Last signal, strobed by strobe, marks the data word with last. This last mark will be written to the FIFO and will also force the Parallel CRC to reinitialize when next data word is received.

Size Used to set the width of the input data bus, 32, 24, 16 or 8 bits.

The inputs to the above signals are configured in the rw_pdp0_cfg and rw_pdp1_cfg registers, using the in_src, in_strb, in_last and in_size fields.

13.7.1.5.5 Parallel Data Path out

The output Parallel Data Path (PDP_out) consists of three parts, DMC, FIFO and Parallel CRC generator. There are two PDP_out in the ETRAX FS I/O Processor. The connection between Parallel CRC generator and FIFO out are fixed, meaning Parallel CRC 0 generator will always be connected to FIFO OUT 0. Connection between DMC out and Parallel CRC generator port is configurable. See figure 13.14 below for a picture of PDP_out 0 and 1.

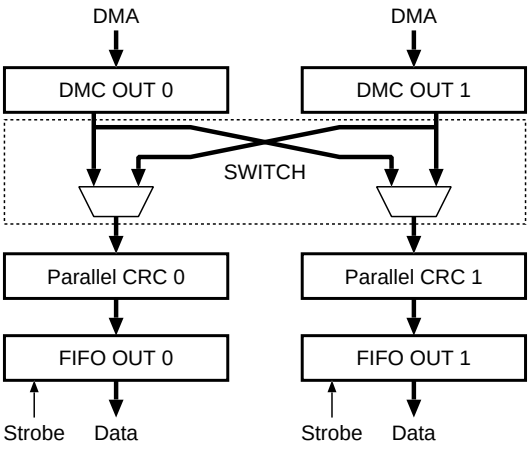


Figure 13.14: *Switch, Output Parallel Data Paths*

As seen in figure 13.14, it is possible to configure which DMC to connect to which Parallel CRC. This is useful when running a full duplex protocol with Parallel CRC since each Parallel CRC only can handle half duplex i.e., either work as generator or checker.

The configuration of DMC connection is done through the dmc0_usr/dmc1_usr and out_src fields in the rw_pdp0_cfg and rw_pdp1_cfg registers.

Selection of the strobe signal for strobing data out from the PDP is done through the out_strb field in the rw_pdp0_cfg and rw_pdp1_cfg registers.

As for the input Parallel Data Path data can be read and written through registers in each separate module register bank.

13.7.1.5.6 Serial CRC in

The ETRAX FS I/O Processor has two Serial CRC input modules. Configuration of the serial data path is done in the Switch, using the rw_sdp_cfg register. The serial data interface connected to the Serial CRC input module consists of the following three signals:

1. Data: Serial data, strobed by strobe.
2. Strobe: Strobe signal, controls when data is sampled.
3. Last: Last signal, strobed by strobe, indicates last data bit. The CRC register will be reinitialized when next bit is received.

Sources for data, strobe and last are configured using the sdp_in0_data, sdp_in1_data, sdp_in0_strb, sdp_in1_strb and sdp_in0_last, sdp_in1_last fields in the rw_sdp_cfg register. Data, strobe and last can also be set through registers in the Serial CRC in module.

13.7.1.5.7 Serial CRC out

In the I/O Processor there are two Serial CRC output modules. There is only one signal to each Serial CRC output module that can be configured - the strobe signal. Fields sdp_out0_strb and sdp_out1_strb in register rw_sdp_cfg are used to configure the strobe signal. The strobe can also be generated by writing to the rw_data register within each Serial CRC output module.

13.8 Timer group

A Timer group consists of four Timers and one Clock Generator. The clock generator has a configurable clock and can be chosen by rw_cfg.clk_src to be either the I/O Processor system clock or an external clock from the Switch. Each Timer has a configurable clock for the timer countdown. The clock used is either the I/O Processor system clock or the clock generated by the Clock Generator. The mentioned clocks can be prescaled, at most by a factor of 128, controlled by the rw_cfg.clk_gen_div and rw_cfg.clk_div registers.

Each Timer group has the following interface signals:

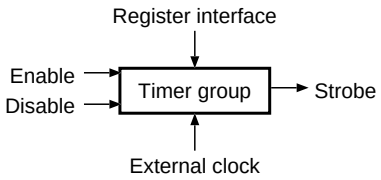


Figure 13.15: *Timer group, Overview*

Enable There are four enable signals, one for each Timer.

Disable There are four disable signals, one for each Timer.

External clock The external clock is used by the Clock Generator as a countdown strobe. It is synchronized by the I/O Processor system clock.

Register interface For a detailed description of all registers, see 25.30.

Strobe Each Timer generates a strobe. A Timer group has four output strobes.

13.8.1 Functional description

The Timer group is configured through the register interface 25.30.

13.8.1.1 Timer

When the Timer is enabled, it will count down r_tmr_cnt to zero. When the counter reaches zero, the Timer output strobe is either toggled or pulsed and an interrupt is generated.

The output strobe is used by the other I/O Processor modules for strobing data or generating timeouts.

A Timer can be reset by the rw_cmd.rst field and in some of the run modes, the Timer is reset when it is disabled. When reset the Timer will copy the value of the rw_tmr_len register to r_tmr_cnt and the countdown will not begin until the Timer is enabled.

13.8.1.1.1 Toggle or pulse mode

A Timer can toggle the output strobe when the countdown reaches zero or it can generate a strobe which has the length of one I/O Processor cycle, depending on the value of rw_tmr_cfg.out_mode. The toggle mode can be used for generating clocks and the pulse mode is useful e.g., as a timeout or to enable/disable another Timer.

13.8.1.1.2 Run modes

A Timer has four different run modes (rw_tmr_cfg.run_mode). The run mode decides how the Timer will act when it is enabled or disabled. In some run modes, the Timer

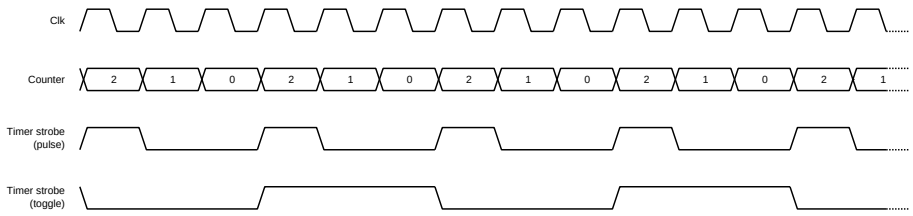


Figure 13.16: *Timer group, Difference between pulse and toggle mode*

is reset when it is disabled. Resetting the Timer will copy the value of the rw_tmr_len register to r_tmr_cnt.

Stop mode In this mode the Timer will be reset when it is disabled.

Pause mode In this mode the Timer will pause when it is disabled. In other words, the Timer countdown is not reset and will continue the countdown once the Timer is enabled.

Complete mode In this mode the Timer will continue its countdown when disabled. In other words, the countdown will not stop immediately but it will not resume the countdown after it reaches zero. If the Timer is disabled a second time during the completion of the countdown it will stop the countdown immediately and reset itself.

Run once mode In this mode the Timer will count down to zero once and then disable itself automatically. If the Timer is disabled during countdown, it will immediately stop the countdown and reset the counter.

13.8.1.1.3 Enable or disable a Timer

The Timer can be enabled or disabled by a Trigger or by another Timer. The owner can also start or stop the Timer by using the register interface.

For example, a Trigger is connected to the enable and disable of a Timer (i.e., enable and disable have the same control signal). See figure 13.17.

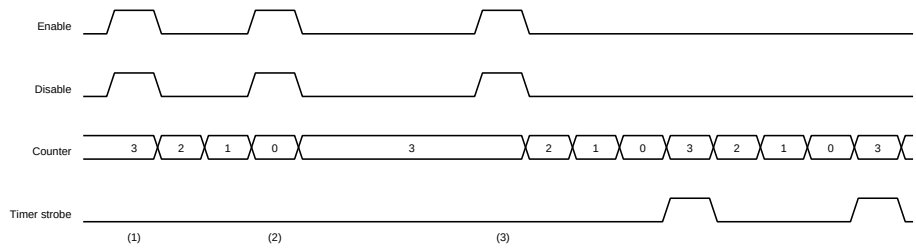


Figure 13.17: *Timer group, Enable and disable at the same time*

Time	Description
1	The Timer is started for the first time.
2	The Timer is paused.
3	The Timer resumes its countdown.

13.8.1.2 Clock Generator

The Clock Generator generates a clock which is used as clock source by a Timer, when selected in rw_tmr_cfg.clk_src.

The generated clock will have a jitter whenever the ideal clock has a half period that is not a multiple of the I/O Processor clock. In that case the clock period and the duty cycle will change during time. Possible period lengths of the generated clock are then N, N+1, and N+2. N is the largest integer which is not larger than I/O Processor system frequency divided by the desired frequency. The jitter of the generated clock will at the most be +/-2.5ns when using a 200 MHz system clock.

The counter (r_clk_gen_cnt) always starts at the value of rw_half_period_len and counts down to either zero or one.

For example, let us assume that you want to make a 18.75 MHz clock from the 200 MHz system clock. It will require 10.666667 clock cycles of the system clock to make one 18.75 MHz cycle. The ideal clock has a 50% duty cycle, so the half period is 5.333333 clock cycles of the system clock.

The Clock Generator will generate 18.75 MHz by altering the output after 5 or 6 cycles. The generated clock will be in sync with the ideal 18.75 MHz after one and a half 18.75 MHz cycle. See figure 13.18.

(*) After one and a half 18.75 MHz cycle the generated clk is in sync with the ideal clock.

The 25ns half period occurs twice as often as the 30ns half period in the above timing diagram, e.g., 2/3 of the generated clock will have a half period of 25ns and 1/3 will have a half period of 30ns.

The jitter is +/-1.667ns with the error pattern +1.667ns, -1.667ns and 0ns. The error pattern will repeat itself continuously.

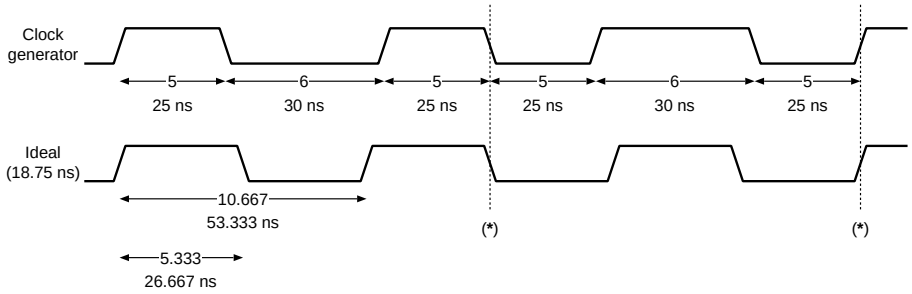


Figure 13.18: *Timer group, Clock Generator example*

13.8.1.2.1 Configuring the Clock Generator

To configure the Clock Generator, you will need to specify the smaller half period (rw_half_period_len), the ratio between the half periods (rw_half_period.quota_lo and rw_half_period.quota_hi) and specify if the smaller half period occurs more often than the larger half period (rw_half_period.quota_hi_sel). Below are the equations for retrieving the parameters put in the configuration registers.

```
    (system_freq / 2)
f = -----------------
      desired_freq

system_freq = The system frequency (200MHz)

desired_freq = The desired frequency


k = f - [f]

[f] is the largest integer which is not larger than f.
```

There are four different cases depending on the value of k:

1. $k < 0.5$

   (The shorter half period occurs more often than the longer half period.)

   rw_half_period.quota_hi_sel = short_period

   ```
   rw_half_period_quota_lo      k
   ---------------------- = -------
   rw_half_period_quota_hi   (1 - k)
   ```

   rw_half_period_len = [f]

2. k > 0.5

   (The longer half period occurs more often than the shorter half period.)

   rw_half_period.quota_hi_sel = long_period

   $$\frac{\text{rw\_half\_period\_quota\_lo}}{\text{rw\_half\_period\_quota\_hi}} = \frac{(1 - k)}{k}$$

   rw_half_period_len = [f]

3. k = 0.5

   (Every clock cycle has the same duty cycle.)

   rw_half_period.quota_hi_sel = short_period

   rw_half_period.quota_lo = 1

   rw_half_period.quota_hi = 1

   rw_half_period_len = [f]

   The duty cycle in this case will always equal [f] / (2 + (1 / [f])).

4. k = 0

   (The desired frequency can be made by dividing the system frequency directly.)

   rw_half_period.quota_hi_sel = short_period

   rw_half_period.quota_lo = 0

   rw_half_period.quota_hi = 1

   rw_half_period_len = [f]

   The generated clock will have exactly the same frequency as the desired clock.

(rw_half_period.quota_lo / rw_half_period.quota_hi) is an irreducible fraction (a fraction whose numerator and denominator cannot be cancelled down any further).

rw_half_period.quota_lo is always smaller than rw_half_period.quota_hi.

- **Example of case 1:**

  Desired frequency: 12 MHz

  System frequency: 200 MHz

  f = (200 MHz / 2) / 12MHz = irreducible fraction = 25 / 3

  [f] = 8

  k = 25 / 3 - 8 = 1 / 3

  $$\frac{\text{rw\_half\_period\_quota\_lo}}{\text{rw\_half\_period\_quota\_hi}} = \frac{(1 / 3)}{(1 - (1 / 3))} = 1 / 2$$

Results using the equations in case 1:

rw_half_period.quota_lo = 1

rw_half_period.quota_hi = 2

rw_half_period.quota_hi_sel = short_period

rw_half_period_len = 8

- **Example of case 2:**

  Desired frequency: 1024 kHz

  System frequency: 200 MHz

  f = (200 MHz / 2) / 1024kHz = irreducible fraction = 3125 / 32

  [f] = 97

  k = 3125 / 32 - 97 = 21 / 32

```
rw_half_period_quota_lo   (1 - (21 / 32))
----------------------- = --------------- = 11 / 21
rw_half_period_quota_hi       (21 / 32)
```

  Results using the equations in case 2:

  rw_half_period.quota_lo = 11

  rw_half_period.quota_hi = 21

  rw_half_period.quota_hi_sel = long_period

  rw_half_period_len = 97

- **Example of case 3:**

  Desired frequency: 64 kHz

  System frequency: 200 MHz

  f = (200 MHz / 2) / 64kHz = irreducible fraction = 3125 / 2

  [f] = 1562

  k = 3125 / 2 - 1562 = 1 / 2

  Results using the equations in case 3:

  rw_half_period.quota_lo = 1

  rw_half_period.quota_hi = 1

  rw_half_period.quota_hi_sel = short_period

  rw_half_period_len = 1562

- **Example of case 4:**

  Desired frequency: 25 MHz

  System frequency: 200 MHz

  f = (200 MHz / 2) / 25 MHz = 4

[f] = 4

k = 4 - 4 = 0

Results using the equations in case 4:

rw_half_period.quota_lo = 0

rw_half_period.quota_hi = 1

rw_half_period.quota_hi_sel = short_period

rw_half_period_len = 4

13.8.1.3 Interrupts

The Timer group has four interrupts. Each Timer generates an interrupt when it has counted down to zero.

13.9 Trigger group

The Trigger group consists of four Triggers. A Trigger is used to continuously monitor a selected input signal.

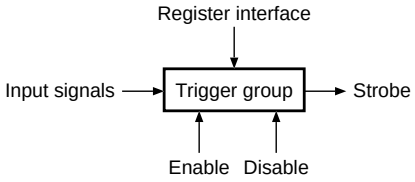


Figure 13.19: *Trigger group, Overview*

Each Trigger group has the following interface signals:

Enable There are four enable signals, one for each Trigger.

Disable There are four disable signals, one for each Trigger.

Register interface For a detailed description of all registers, see 25.31.

Input signal Each Trigger monitors one input signal.

Strobe Each Trigger generates a strobe. The Trigger group has four output strobes.

13.9.1 Functional description

A Trigger is waiting for a transition on the input signal from low to high or high to low. When the selected edge of the input signal is detected, an interrupt can be generated. An output strobe is also generated which can be used as an input to an SPU or to enable or disable a timer.

13.9.1.1 Enable and disable a Trigger

A Trigger can be enabled or disabled by another strobe from e.g. a Timer group. The Switch selects which strobes should be used as enable and disable signals. Also, one or more Triggers can be enabled or disabled at the same time using the rw_cmd register.

13.9.1.2 Trigger configuration

13.9.1.2.1 Edge detection

The Trigger can detect rising and/or falling edges of an input. It can also be configured to detect if the input signal is high or low when enabling the Trigger. The configuration is selected in rw_cfg.trig.

Figure 13.20 shows an example of how two Triggers detect an input. Both triggers detect rising edges, the second does also generate a strobe if the input is high when the Trigger is enabled.

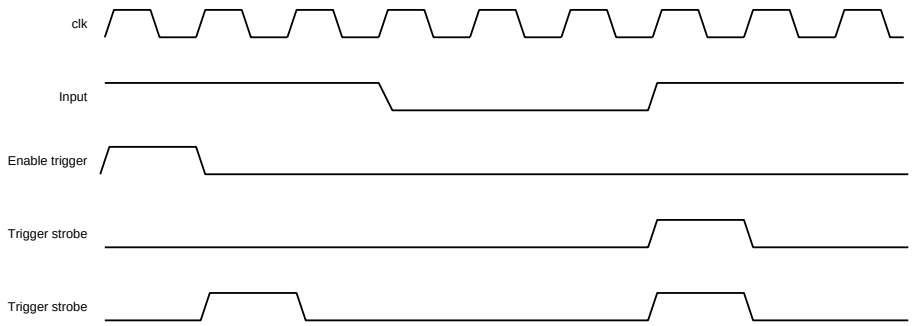


Figure 13.20: *Trigger group, Edge detection*

13.9.1.2.2 Output strobe configuration

The output strobe can be configured in four different ways in rw_cfg.action:

fall The output will stay high until an edge is detected. When edge detection has occurred the output will be low. When the trigger is disabled, the output will once again be high.

rise The output will stay low until an edge is detected. When edge detection has occurred the output will be high. When the trigger is disabled, the output will once again be low.

pulse Each time an edge is detected, the output will send a pulse. The pulse has the length of an I/O Processor system clock cycle.

toggle Each time an edge is detected, the value of the output will toggle.

A trigger can be configured to automatically disable itself when an edge is detected (rw_cfg.once).

13.9.1.3 Interrupts

Each trigger can generate an interrupt when an edge is detected. The interrupts are independent of the configuration of the output strobe.

13.10 DMA Communicator In

The DMA Communicator In (DMC_in) is the DMA in channel interface of the I/O Processor. It handles data streams from the I/O Processor to the DMA. The MPU, SPUs, or the CPU can use the DMC_in registers to send commands to the DMA and update meta data descriptors (data, context or group) for the DMA in channel. The relation between DMC_in and DMA channel numbers is described in 10.

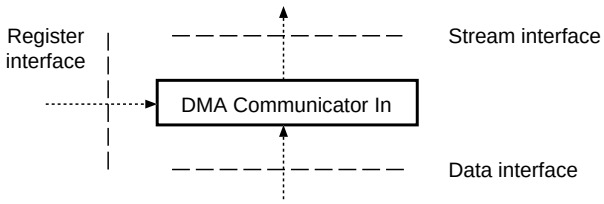


Figure 13.21: *DMA Communicator In, Overview*

13.10.1 Functional description

The DMC_in receives data either from the register interface or the data interface and transmits it to the DMA using the stream interface.

The data interface can be disabled (rw_ctrl.dif_dis), no data will then pass through it. When the data interface is enabled, the DMC_in will send data to the DMA whenever there is valid data from the data interface. The DMC_in can be configured in rw_cfg.last_dis_dif to block the data interface after the last data of the data stream is received.

The data interface should be disabled before transmitting data from the register interface. It is also possible to read the status of the stream interface, updating meta data in the descriptors (data, context and group) and setting up the interrupts, by using the register interface.

DMA commands are sent through the stream interface to the DMA via the DMC_in using rw_stream_cmd.

The DMA commands and other DMA functionality is described in 5.

13.10.1.1 Interrupts

The DMC_in can generate six interrupts. All interrupts can be masked and cleared using rw_intr_mask and rw_ack_intr. The interrupts and their meanings are:

1. Command ready interrupt (cmd_rdy)

   The interrupt is generated when the DMA is ready to receive a command.

2. Group meta data interrupt (group_md)

   The interrupt is generated when meta data of the group descriptor is valid.

3. Context meta data interrupt (ctxt_md)

   The interrupt is generated when meta data of the context descriptor is valid.

4. Data meta data interrupt (data_md)

   The interrupt is generated when meta data of the data descriptor is valid.

5. Threshold interrupt (sth)

   Generates an interrupt if the number of free bytes in the DMA is equal or larger than the specified number in rw_cfg.sth_intr.

6. DMA full interrupt (full)

   The interrupt is generated when the DMA FIFO becomes full.

13.11 DMA Communicator Out

The DMA Communicator Out (DMC_out) is the DMA out channel interface of the I/O Processor. It handles data streams from the DMA to the I/O Processor. The MPU, SPUs, or the CPU can use the DMC_out registers to send commands to the DMA and update meta data descriptors (data, context or group) for the DMA out channel. The relation between DMC_out and DMA channel numbers is described in 10.

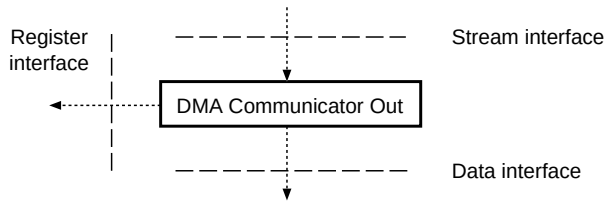


Figure 13.22: *DMA Communicator Out, Overview*

13.11.1 Functional description

The DMC_out receives data from the DMA using the stream interface and the data stream can be output using either the data interface or the register interface.

The data interface can be disabled (rw_ctrl.dif_dis), no data will then pass through it. When the data interface is enabled, it will send data whenever there is valid data from the stream interface. There is also an option to disable the data interface after DMC_out has transmitted a number of transfers, specified by rw_cfg.trf_lim.

The data interface should be disabled before receiving data to the register interface. It is also possible to read the status of the stream interface, updating meta data in the descriptors (data, context and group) and setting up the interrupts, by using the register interface.

DMA commands are sent through the stream interface to the DMA via the DMC_out using rw_stream_cmd.

The DMA commands and other DMA functionality is described in 5.

13.11.1.1 Interrupts

The DMC_out can generate nine interrupts. All interrupts can be masked and cleared using rw_intr_mask and rw_ack_intr. The interrupts and their meanings are:

1. Command request interrupt (cmd_rq)

   The interrupt is generated when the DMA is waiting for a command.

2. Transfer limit interrupt (trf_lim)

   The interrupt is generated when the DMC_out automatically disables the data interface after counting a specified number of transfers, configured in rw_cfg.trf_lim.

3. Command ready interrupt (cmd_rdy)

   The interrupt is generated when the DMA is ready to receive a command.

4. Group meta data interrupt (group_md)

   The interrupt is generated when meta data of the group descriptor is valid.

5. Context meta data interrupt (ctxt_md)

   The interrupt is generated when meta data of the context descriptor is valid.

6. Data meta data interrupt (data_md)

   The interrupt is generated when meta data of the data descriptor is valid.

7. Data threshold interrupt (dth)

   The interrupt is generated if the number of bytes buffered in the DMA is equal or larger than the specified number in rw_cfg.dth_intr. The interrupt is also generated if all data in the packet has been read from memory and it is available for immediate reading on the streaming interface.

8. Last data interrupt (last_data)

   The interrupt is generated if data from the streaming interface is the last of the data stream.

9. Data valid interrupt (dv)

   The interrupt is generated if data from the streaming interface is valid.

13.12 FIFO

The FIFO is used to provide an additional data buffer. The rate at which data is written and read is controlled by two strobes, so data is read and written at some individually predetermined rate, set either by an external unit or by an internal unit such as a timer. In addition to this, the width of the data paths towards the I/O buses of the I/O Processor can be changed, from the standard 32-bit width, to 24, 16 or eight bit wide.

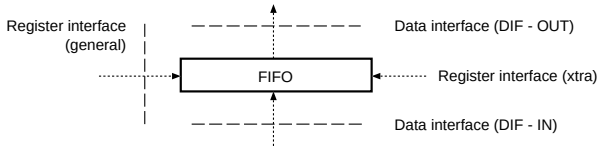


Figure 13.23: *FIFO, Overview*

13.12.1 Functional description

The FIFO is 8 bytes deep. The access mechanisms available to the FIFO varies between the input and output FIFO.

Access Mechanism	In FIFO	Out FIFO
Writing data through DIF-IN.	Supported (data arrives from I/O Processor input bus 0 or 1).	Supported (data arrives from DMC-out).
Writing data through the general register interface.	Not supported by the in FIFO.	Supported through: rw_wr1byte, rw_wr2byte, rw_wr3byte, and rw_wr4byte.
Writing data through the xtra register inter face.	Supported through the iop_fifo_in_extra.rw_wr_data register. The amount of data written depends on how the in FIFO is configured. Eg. in 8 bit mode 1 byte is written and in 16 bit mode 2 bytes are written and so on.	Not supported.
Reading data through the general register interface.	Supported through: rs_rd1byte, rs_rd2byte, rs_rd3byte, and rs_rd4byte.	Not supported.
Reading data through the xtra register interface.	Not supported.	Supported through the r_rd_data register. The number of bytes read depends on how the out FIFO is configured. Eg. in 8 bit mode 1 byte is read and in 16 bit mode 2 bytes are read and so on.
Reading data through DIF-OUT.	Data is read through DIF-OUT by the DMC. The bus-width and strobe signals are controlled entirely by hardware.	Data is read by using an external strobe source (such as a general I/O or timer), or by using the rw_strb_dif_out register.

Table 13.61: *FIFO, Access mechanisms and their relations to the in and out FIF Os*

It is not possible to read or write data from two sources at the same time, e.g., writ-

ing through any register interface at the same time as the DIF-IN is active will write corrupted data to the FIFO.

The FIFO has a mechanism for marking packet boundaries. Each byte in the FIFO can be marked with a flag, the `last` flag. The `last` flag keeps track of the last byte in a packet, and can be configured to signal an interrupt when the last byte occurs. Marking a byte differs slightly between the two types of FIFOs. The following methods are available:

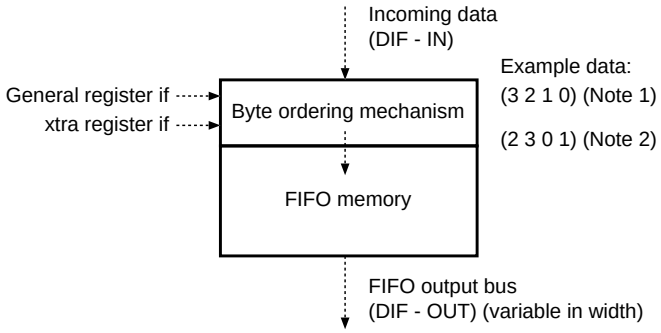
1. The `last` flag can be set through DIF-IN. This differs between the two types of FIFOs according to the following:
   - In the output FIFO: only DMC out can set the `last` flag through the DIF-IN
   - In the input FIFO: `last` can be set by an SPU, timer, or an external signal
2. The `last` flag can be set through the register interface by writing one of the following registers:
   - In the output FIFO: rw_wr1byte_last, rw_wr2byte_last, rw_wr3byte_last, rw_wr4byte_last. For further information regarding these registers, please see 25.18
   - In the input FIFO: The `last` can be set by writing to the register rw_set_last (available only in 25.19) or rw_strb_dif_in, which is available in both 25.19 *and* 25.21

13.12.1.1 The FIFO byte order

The FIFO has four different byte orders. The byte order is configured by writing to the byte_order field, in the rw_cfg register. In this text, the highest byte (bits 31 through 24) will be called ”3”, the next highest byte (bits 23 through 16) will be called ”2” etc. The FIFO can be configured to reorder the four bytes individually. There are four different possibilities:

1. 8-bit order. This is the default byte order, data is not reordered at all (i.e., 3 2 1 0 -> 3 2 1 0).
2. 16-bit order. The two bytes of each word of incoming data will be reordered. The result in the FIFO will be (3 2 1 0 -> 2 3 0 1).
3. 24-bit order. Only the lower three bytes of incoming data will be reordered, thus the result will be (3 2 1 0 -> 3 0 1 2).
4. 32-bit order. In 32-bit order all the 32 bits are reordered. The result is (3 2 1 0 -> 0 1 2 3).

The byte ordering mechanism is independent of the configured width of the FIFO output bus (see the section describing the FIFO output bus mode). When turned on, any bytes written will be reordered bytes as described above regardless of the width of the output bus. Care should be taken when using this mechanism with protocols less than 32 bits wide, as unused (and thus undefined) data can be swapped with payload data. See the figure 13.24 below.



Note 1: Data entering the byte ordering mechanism of the FIFO is in this order.

Note 2: The order of the data the FIFO is able to access.

Figure 13.24: *FIFO, Byte ordering mechanism*

13.12.1.2 FIFO Output bus mode

The width of the output bus (DIF-OUT) can be configured by accessing the mode field in the rw_cfg register. The FIFO (and thus the width of the output bus) can be configured to operate in one of the following modes:

- 32-bit mode
- 24-bit mode
- 16-bit mode
- 8-bit mode

The width of the input bus is not affected by the configured FIFO output bus mode. The table below shows the implications of the byte ordering mechanism with respect to the various FIFO output bus modes:

Mode	Description
32-bit	Any byte ordering can be used.
24-bit	It is only possible to read or write 24 bits at a time. Set the byte ordering mechanism to either 8-bit or 24-bit order.
16-bit	Use any byte order setting when writing 4 bytes at a time. If less than 4 bytes are written at any time, the byte ordering mechanism should be set to 8-bit order.
8-bit	Set the byte ordering mechanism to 8-bit order.

Table 13.62: *FIFO, Output bus modes and byte ordering mechanism*

13.12.1.3 Software interface

Interfacing the FIFO from software can be done using two different interfaces:

1. General Register interface: This interface is mapped to the primary owner through the switch. The interface is used to configure the FIFO, and is also used for writing or reading data to or from the FIFO. For further details regarding the functionality available through this interface, please refer to [FIFO_REGS].
2. The xtra register interface: The xtra-interface allows data to be read/written directly by a secondary owner. No other operations can be performed by this interface. For further details regarding the functionality, please refer to [FIFO_XTRA_REGS].

13.12.1.4 Interrupts

Interrupts are generated by five different sources. Each interrupt has a mask bit in the interrupt mask register allowing the interrupt to be turned on or off. The five interrupt sources are:

1. Underrun interrupt (urun)

   The underrun interrupt is generated if the reader reads data faster than it can be written to the FIFO. The interrupt is generated when a read operation is detected while the FIFO is empty. This interrupt is generated by both types of FIFOs (i.e., input and output).

2. Free interrupt (free)

   The free interrupt is generated when the free space in the FIFO is equal to or greater than a predefined limit set in rw_cfg.free_lim. This interrupt is an indication to a possible writer that it is possible to write data to the FIFO. This interrupt is generated by the output FIFO only.

3. Data available interrupt (dav)

   The data available interrupt is generated as soon as data becomes available in the FIFO. This interrupt is generated as soon as payload data is available in either type of FIFO (i.e., input and output).

4. Last interrupt (last_data)

   The last interrupt is generated if the `last` flag is set when writing the FIFO or when the `last` is set by an external controller. The interrupt is generated by both types of FIFOs (i.e., input and output).

5. Overrun interrupt (orun)

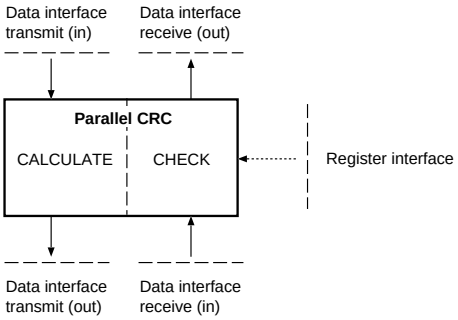
   The overrun interrupt is generated when a write attempt will cause yet unread data to be overwritten. This can never happen when the FIFO is written by the DIF-IN. This interrupt is generated by both types of FIFOs (i.e., input and output).

6. Available interrupt (avail)

   The available interrupt is generated in the input FIFO when the number of bytes in the input FIFO is greater than or equal to the value specified by the avail_lim field. This interrupt is only generated by the input FIFO.

13.13 Parallel CRC

The parallel CRC module can either check the CRC for the received data or calculate and transmit CRC when transmitting data. There are a couple of predefined generator polynomials which can be used.



Note: It is not possible to calculate and check CRC at the same time.

Figure 13.25: *Parallel CRC, Overview*

13.13.1 Functional description

The CRC module should be configured with the desired CRC polynomial. The CRC calculation is performed on the incoming data from either the transmitting interface or the receiving interface or from the register interface.

The parallel CRC module is disabled by default i.e., rw_ctrl.en is set to no. When disabled, the incoming data will be sent out unchanged and the shift register will keep its value. When the CRC module is enabled it must be configured to handle either data from the received data interface or from the transmit data interface.

13.13.1.1 Data interfaces

There are five different data interfaces in the parallel CRC module. Two are used when transmitting data (calculating CRC), the third and fourth interface are used when receiving data (checking CRC) and the last interface is the register interface which can be used either when calculating or checking data depending on the value of rw_cfg.mode. The five interfaces have all one to four bytes of data together with a strobe. Each interface has also a status bit which tells if the data bytes are the last of the packet.

The data in the receive data interface (in) can be sampled by the CRC module on the rising or falling edge of the strobe, which is configured in rw_cfg.trig. A second way of strobing data on the receive data interface is through rw_strb_rec_dif_in.

The rw_wr1byte, rw_wr2byte, rw_wr3byte or rw_wr4byte register can be used when writing data to the CRC module through the register interface. If the data should be marked as last, rw_wr1byte_last, rw_wr2byte_last, rw_wr3byte_last or rw_wr4byte_last should be used instead. The data written to the registers will be output on the receive data interface (out) or transmit data interface (out), depending on the value of rw_cfg.mode. A zero-sized word marked as the last of the packet can be sent with rw_set_last.

The CRC module calculates or checks CRC on only one data byte each clock cycle. For example, if four bytes of data enter the data transmit (in) interface while in calculate mode, no more data can enter that interface until those four bytes of data are calculated

(i.e., after four clock cycles). But data can still pass through the receive data interface. The same is also true in check mode when data enters the data receive (in) interface. If four bytes of data go into this interface of the CRC, no more data can enter until those four bytes are checked. But data can still pass through the CRC through the transmit data interface.

The r_stat.busy field should be read before writing data through the register interface to the CRC module. If the CRC module is busy calculating or checking the CRC, r_stat.busy will be set to yes, which means that data must not be written to the CRC through the register interface.

13.13.1.2 CRC Configuration

The initial shift register value (the value which is input to the shift register when reset or after the last byte of the data stream was received) is configurable in rw_init_crc. The CRC polynomial is selected in rw_cfg.poly and can be one of the following:

crc32 $p(x)=x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

crc16 $p(x)=x^{16}+x^{15}+x^2+1$

ccitt $p(x)=x^{16}+x^{12}+x^5+1$

crc5 $p(x)=x^5+x^2+1$

The current value of the CRC shift register is stored in r_sh_reg. The value of r_sh_reg can have its bits inverted (rw_cfg.inv_out) and the bit order can be reversed (rw_cfg.rev_out). The result is the calculated CRC which is stored in r_crc.

A data packet which is to be transmitted can have the value of r_crc appended at the end of the packet. This is configured in rw_cfg.crc_out.

13.13.1.3 Error detection

When receiving data from the bus, the value of the shift register, r_sh_reg, is compared with the value of a configurable register, rw_correct_crc. If the registers are not equal, a flag is raised in the status register (r_stat.err). There is also a signal which is asserted when the values are the same. The signal can be used as an input to the Slave Processing Unit (iop_spu.r_stat_in.pcrc_correct).

13.14 Serial CRC In

CRC can be computed for the incoming serial bits and if the value does not match a predefined value a status flag is raised. The CRC generator can be any polynomial with a length up to 32.

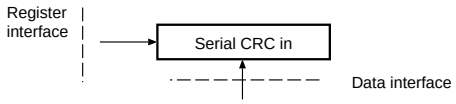


Figure 13.26: *Serial CRC In, Overview*

13.14.1 Functional description

The data bits which the CRC should be calculated on, are received from the data interface or written to the serial CRC block in rw_wr1bit. The value of the CRC shift register can be read out at any time from r_computed_crc.

13.14.1.1 Configuration

The CRC generator polynomial is selected in rw_crc and the initial value of the CRC shift register is written to rw_init_crc. The shift register is updated either when a value is written to rw_wr1bit or when receiving a strobe from e.g. a Timer. Which source is used as the strobe is selected in the Switch.

After data marked as the last of the packet is received, the value of rw_init_crc will be used for calculating CRC of the next data bit.

13.14.1.2 CRC Validation

The correct CRC is stored in rw_correct_crc. When the value of r_computed_crc is not equal the correct CRC, the status bit in r_stat.err is raised. The value of r_stat.err is updated for each new data bit.

13.15 Serial CRC Out

CRC can be computed for the outgoing serial bits and appended at the end of the data stream. The CRC generator can be any polynomial with a length up to 32.

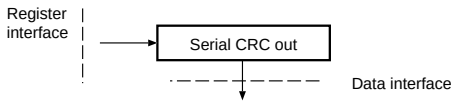


Figure 13.27: *Serial CRC Out, Overview*

13.15.1 Functional description

The data bits which the CRC should be calculated on, are written to the serial CRC block in rw_data.val. The value of the CRC shift register can be read out at any time

from r_computed_crc.

13.15.1.1 Configuration

The CRC generator polynomial is selected in rw_crc and the initial value of the CRC shift register is written to rw_init_crc. The shift register is updated either when a value is written to rw_data or when receiving a strobe from e.g. a Timer. Which source is used as the strobe is selected in the Switch.

13.15.1.2 Data interface

When rw_ctrl.out_src is set to output data, the value of rw_data will be output on the serial data interface.

When rw_ctrl.out_src is set to output CRC, a CRC bit will be output on the serial data interface. The next CRC bit is transmitted when a strobe is received on the data interface or when writing to rw_data. In this case it does not matter if it is a zero or a one written to val since the value is never used.

13.16 Synchronization and Asynchronous Paths

The Synchronization and Asynchronous Paths (SAP) module is divided into two parts, SAP_in and SAP_out. Both of these modules are controlled by registers and have their own register banks. Like other modules in the I/O Processor the SAP_in and SAP_out register banks can be owned by one of CPU/MPU/SPU0/SPU1, where CPU is the default user. See 25.23 and 25.24 for a complete listing and explanation of the registers.

The SAP_in module handles synchronization of I/O signals going into the I/O Processor, adapting them to the I/O Processor 200 MHz clock.

The SAP_out module controls output clocking of I/O signals (including output enables, OE) from the I/O Processor.

In ETRAX FS there are 72 bidirectional pins connected to the I/O Processor. The I/O Processor uses these 72 pins to form three internal buses, refer to section 13.7.1.4 for more information. The three internal buses are:

- Two 32 bit wide buses with one OE per byte.
- One 32 bit wide GIO bus with one OE per bit.

13.16.1 SAP_in Functional description

The SAP_in module handles synchronization of buses and GIO inputs to the I/O Processor. See figure 13.28 below.

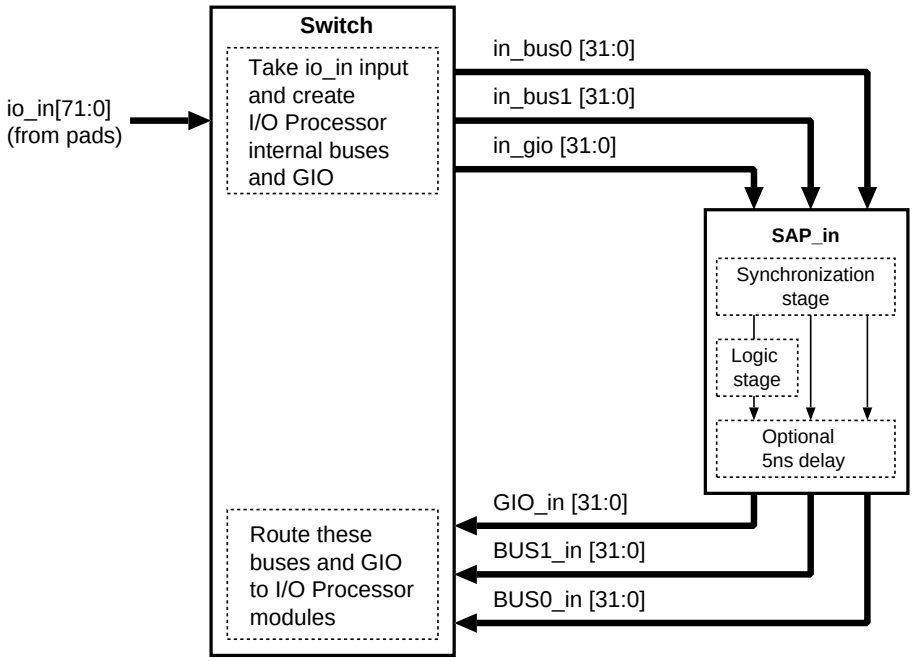


Figure 13.28: *SAP_in, Overview*

13.16.1.1 Buses (in), synchronization

Each bus is divided into four parts, one per byte. The configuration of input synchronization is done using the rw_bus0_sync and rw_bus1_sync registers. Each byte of the bus is configured using four register fields. In the following sub-sections the register fields are described.

13.16.1.1.1 byte0_sel ... byte3_sel

These register fields are used to select input synchronization type. Each byte can be synchronized with four different methods.

1. two_clk200

   Delay for this configuration is 2.5 - 7.5 ns. See figure 13.29.

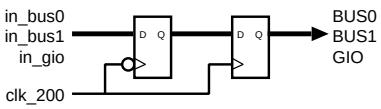


Figure 13.29: *SAP_in, Synchronization method two_clk_200*

2. tmr_clk200

Delay for this configuration is 2.5 - 7.5 ns. See figure 13.30.

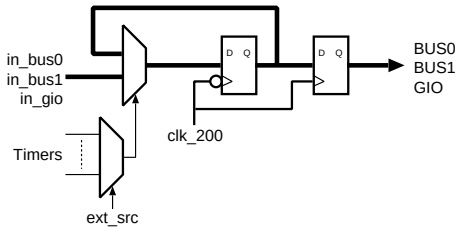


Figure 13.30: *SAP_in, Synchronization method timer_clk_200*

3. ext_clk200

   Delay for this configuration is 2.5 - 7.5 ns (relative to the strobe). See figure 13.31.

   ext_clk200 can handle input synchronization width zero hold time since data is delayed to assure strobe to arrive before data.

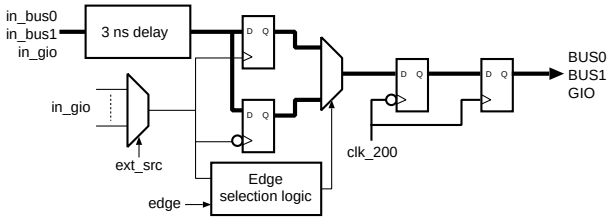


Figure 13.31: *SAP_in, Synchronization method ext_clk_200*

4. no_del_ext_clk200

   Delay for this configuration is 2.5 - 7.5 ns. See figure 13.32.

   no_del_ext_clk200 is used for protocols with short data setup time.

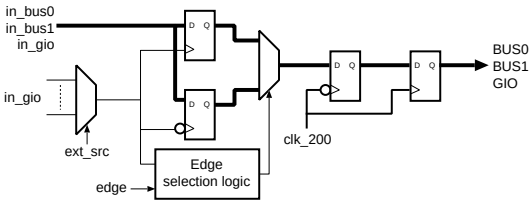


Figure 13.32: *SAP_in, Synchronization method no_del_ext_clk_200*

13.16.1.1.2 byte0_ext_src ... byte3_ext_src

Used to specify which GIO (used by ext_clk200 or no_del_ext_clk200) or which Timer (used by tmr_clk200) to use as input synchronization strobe.

The different GIO (not synchronized) which can be used as strobe are: gio1, gio6, gio7, gio18, gio19, gio23.

The different Timers which can be used are: timer_grp0_tmr3, timer_grp3_tmr3.

13.16.1.1.3 byte0_edge ... byte3_edge

These register fields are used to select which edge of the GIO (selected by byte0_ext_src ... byte3_ext_src) to use as strobe.

13.16.1.1.4 byte0_delay ... byte3_delay

These register fields are used to add a extra delay of 5 ns to the synchronized byte. See figure 13.33.

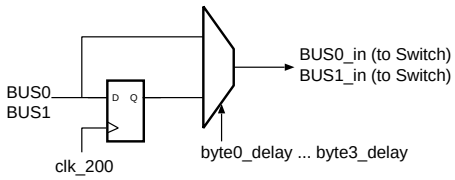


Figure 13.33: *SAP_in, BUS delay stage*

13.16.1.2 GIO:s (in)

13.16.1.2.1 Synchronization

The 32 GIO:s have the same configuration options as the two buses, see section 13.16.1.1. The configuration of each GIO is done through the vector register rw_gio.

Each GIO bit can be synchronized using the same method as for the buses, described in previous sections.

The different GIO (not synchronized) which can be used as strobe are: gio1, gio5, gio6, gio7, gio13, gio18, gio21, gio29.

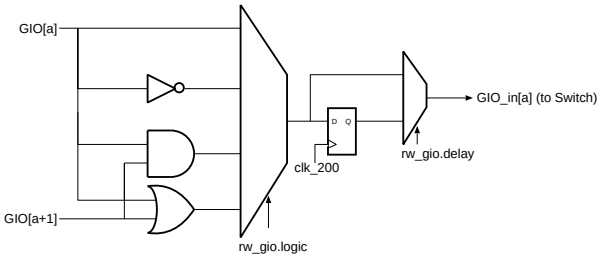
The different timers which can be used are: timer_grp0_tmr3, timer_grp1_tmr3, timer_grp2_tmr3, timer_grp3_tmr3.

The delay option, delaying the signal 5 ns (using field delay), is performed after the logic stage, see figure 13.34.

13.16.1.2.2 Logic stage

After the synchronization of the GIO signals they pass a logic stage as shown in figure 13.34 below. The logic to perform is selected by the rw_gio.logic register field.

GIO[a] and GIO[a+1] are the synchronized GIO (but before delay is added by using the delay register field).



Note: For a=31, a+1 equals 0.

Figure 13.34: *SAP_in, GIO logic stage*

13.16.2 SAP_out Functional description

The SAP_out module selects different clocks for strobing out I/O signals and OE:s. See figure 13.35.

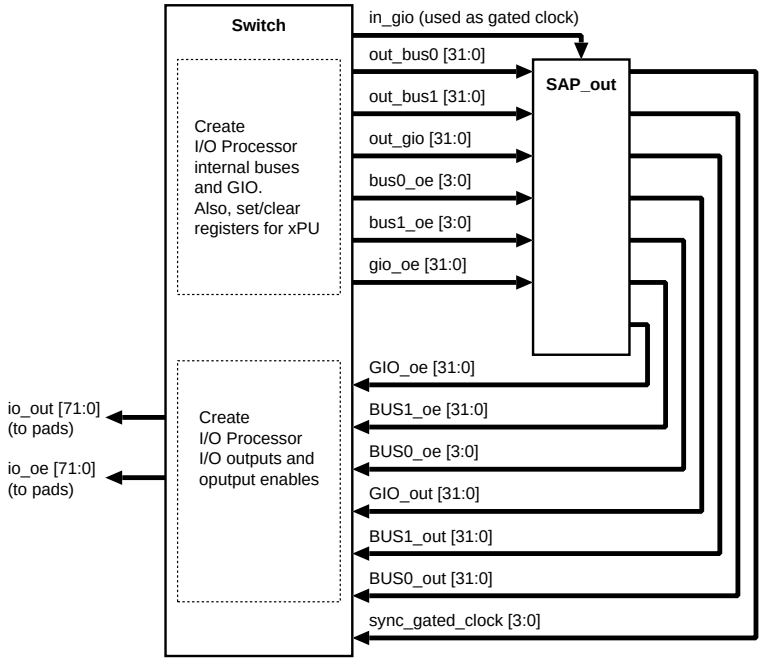


Figure 13.35: *SAP_out, Overview*

13.16.2.1 Gated clocks

Four gated clocks are specified/generated by using the rw_gen_gated register. A gated clock can be used to clock out buses, GIO and output enable signals. Figure 13.36 shows how gated clock 0 is generated, but all four gated clocks are generated in the same way.

A synchronized version of the gated clock is generated using two 200 MHz flip-flops (the first one on negedge), yielding a delay of 2.5 - 7.5 ns. These synchronized gated clocks, sync_gated_clock[3:0], can be used as strobes in the parallel data paths of the I/O Processor.

Each gated clock uses three different fields in the rw_gen_gated register, see figure 13.36.

1. clk0_src ... clk3_src

   Selects an asynchronous GIO (taken before SAP_in) to use as a gated clock.

   The different asynchronous GIO which can be used as clock are:

   gio1, gio5, gio13, gio18

2. clk0_gate_src ... clk3_gate_src

   Selects an asynchronous GIO (taken before SAP_in) to use as a gate signal to the clock selected by clk_src.

   The different asynchronous GIO which can be used as gate signal are:

   gio7, gio15, gio23, gio31

3. clk0_force_src ... clk3_force_src

   Selects an SPU gio_out signal to be used to force the gate signal to one.

   The different SPU gio_out signals which can be used to force the gate signal are:

   none, SPU0_gio6, SPU0_gio7, SPU0_gio15, SPU1_gio6, SPU1_gio7, SPU1_gio15

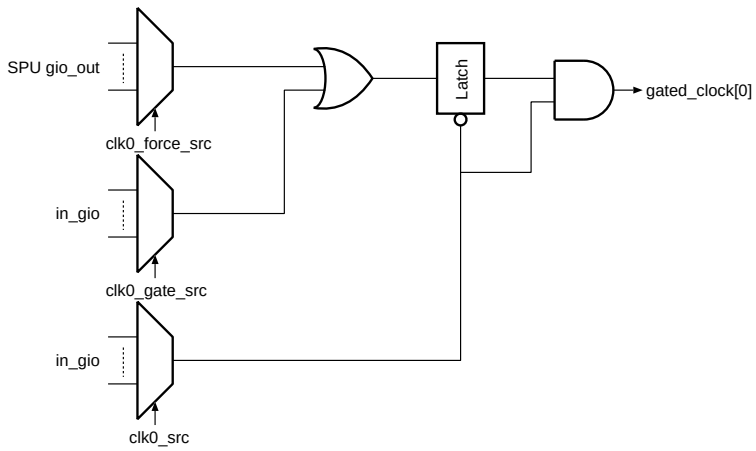


Figure 13.36: *SAP_out, Generating gated clock 0*

13.16.2.2 Buses (out)

As for the SAP_in module each bus is divided into four parts (one for each byte). For controlling output clocking behavior of buses the registers rw_bus0 and rw_bus1 are used. There are three fields for each byte of a bus. Each byte of a bus can be configured to be clocked out. See figure 13.37 (figure shows byte 0 of BUS0).

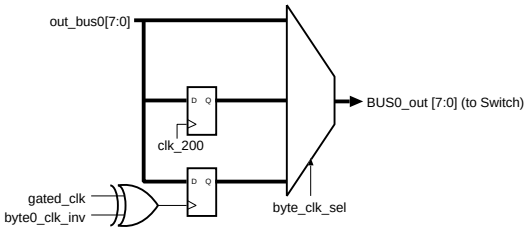


Figure 13.37: *SAP_out, Bus out synchronization, byte0 of BUS0*

In the following sub-sections detailed description is given on the fields of the bus synchronization registers, rw_bus0 and rw_bus1. As for SAP_in there are fields for each byte.

13.16.2.2.1 byte0_clk_sel ... byte3_clk_sel

These register fields are used to select output clocking type. Each byte can be clocked out in three different ways:

- none

  No output clocking. Signals do not pass any extra flip-flops.

- clk200

  One 200 MHz flip-flop is used.

- gated

  One flip-flop clocked by a generated gated clock, selected by byte0_gated_clk ... byte3_gated_clk. The selected gated clock can be inverted using byte0_clk_inv ... byte3_clk_inv.

13.16.2.2.2 byte0_gated_clk ... byte3_gated_clk

Register fields used to select one of the four generated gated clocks, see section 13.16.2.1. Only used when byte0_clk_sel ... byte3_clk_sel is set to gated.

13.16.2.2.3 byte0_clk_inv ... byte3_clk_inv

The byte0_clk_inv ... byte3_clk_inv fields give the opportunity to invert the gated clock. This enables output clocking on negative clock edge. Only used when byte0_clk_sel ...

byte3_clk_sel is set to gated.

13.16.2.3 Bus output enables

There is one output enable for each byte of a bus. The configuration of the bus OE:s are done through the following registers:

- rw_bus0_lo_oe

  Selects OE configuration of byte 0 and 1 of BUS0.

- rw_bus0_hi_oe

  Selects OE configuration of byte 2 and 3 of BUS0.

- rw_bus1_lo_oe

  Selects OE configuration of byte 0 and 1 of BUS1.

- rw_bus1_hi_oe

  Selects OE configuration of byte 2 and 3 of BUS1.

The output enable signal for each byte of a bus can be configured as shown in figure 13.38 (figure shows OE for byte 0, BUS0).

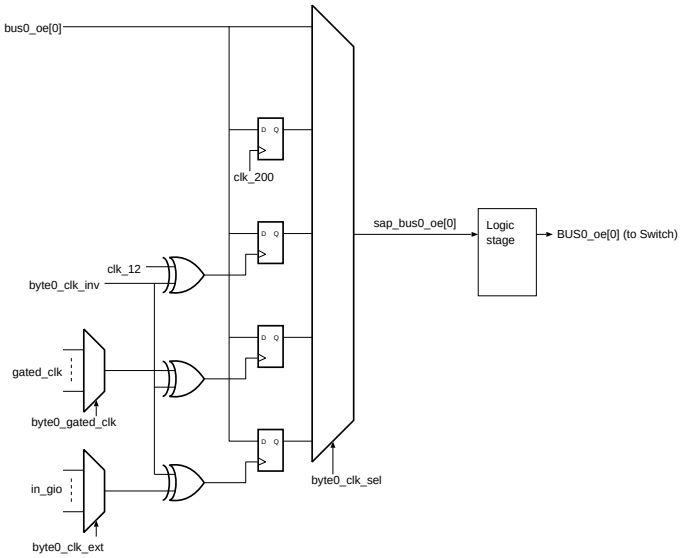


Figure 13.38: *SAP_out, Bus output enable configuration, byte 0 of BUS0*

In the following sub-sections detailed description is given on the fields of the bus synchronization registers, rw_bus0_hi_oe, rw_bus0_lo_oe, rw_bus1_hi_oe and rw_bus1_lo_oe.

13.16.2.3.1 byte0_clk_sel ... byte3_clk_sel

These register fields are used to select output clocking type. Each OE can be clocked out in five different ways:

1. none

   No output clocking. Signals do not pass any extra flip-flops.

2. clk200

   One 200 MHz flip-flop is used.

3. clk12

   One 12 MHz flip-flop is used. The 12 MHz clock can be inverted using the byte0_clk_inv ... byte3_clk_inv field.

4. gated

   One flip-flop clocked by a generated gated clock, selected by byte0_gated_clk ... byte3_gated_clk. The selected gated clock can be inverted using byte0_clk_inv ... byte3_clk_inv.

5. ext

   One flip-flop clocked by an external (asynchronous) signal selected by byte0_clk_ext ... byte3_clk_ext. The ext_clk can be inverted using byte0_clk_inv ... byte3_clk_inv.

13.16.2.3.2 byte0_clk_ext ... byte3_clk_ext

Register field to select an asynchronous GIO (taken before SAP_in) to use as external clock. Only used when byte0_clk_sel ... byte3_clk_sel is set to ext.

The different asynchronous GIO which can be used as clock are:

gio5, gio13, gio21, gio29

13.16.2.3.3 byte0_gated_clk ... byte3_gated_clk

Register field used to select one of the four generated gated clocks. Only used when byte0_clk_sel ... byte3_clk_sel is set to gated.

13.16.2.3.4 byte0_clk_inv ... byte3_clk_inv

The byte0_clk_inv ... byte3_clk_inv fields give the opportunity to invert the 12 MHz, external or gated clock. This enables output clocking on negative clock edge. Only used when byte0_clk_sel ... byte3_clk_sel is set to clk12, ext or gated.

13.16.2.3.5 byte0_logic ... byte3_logic

These registers configure the logic stage for each OE signal. See figure 13.39 (figure shows OE for byte 0 of BUS0). The out_gio which can be used to mask the output enable signal is out_gio[4] for BUS0 and out_gio[0] for BUS1.

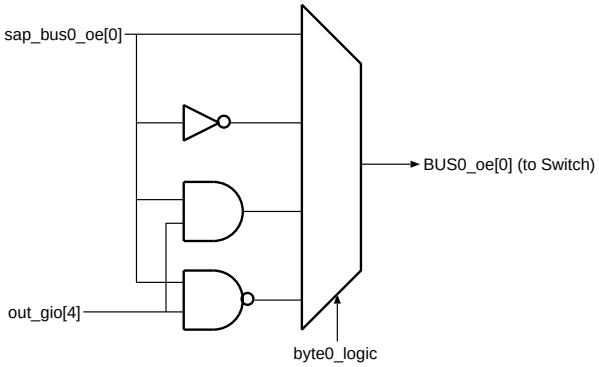


Figure 13.39: *SAP_out, Bus output enable logic stage, byte 0 of BUS0*

13.16.2.4 GIO:s (out)

Each bit of the 32 bit wide GIO_out can have its own output clocking configuration. The vector register rw_gio contains ten fields where five of them are used for GIO and the other five are used for output enable (OE) of each GIO. The GIO out signal can be configured to be clocked out according to figure See figure 13.40 (figure shows GIO[0]).

In the following sub-sections the register fields of the rw_gio register which are used to select output clocking of GIO are described.

13.16.2.4.1 out_clk_sel

This register field is used to select output clocking type. Each GIO can be clocked out in six different ways:

1. none

   No output clocking. Signals do not pass any extra flip-flops.

2. clk200

   One 200 MHz flip-flop is used.

3. tmr

   One flip-flop clocked by a 200 MHz clock. Data to the flip-flop is only changed when the timer signal, selected by the out_clk_ext field, is active.

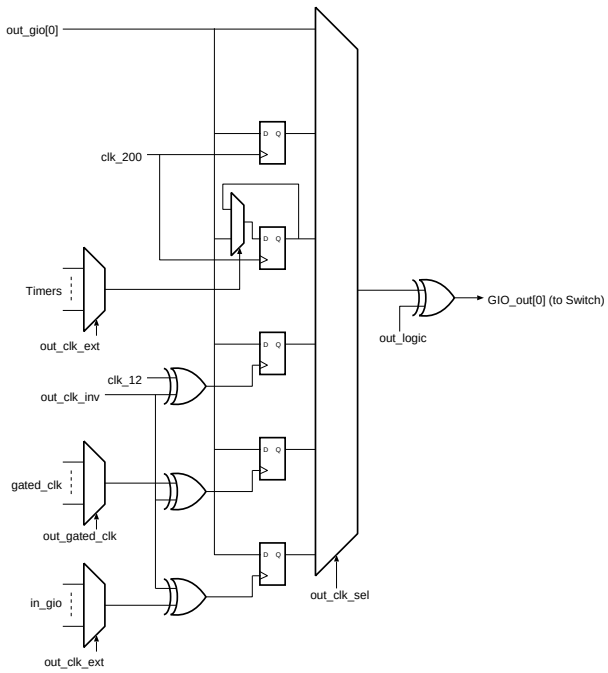


Figure 13.40: *SAP_out, GIO out synchronization, GIO_out 0*

4. clk12

   One 12 MHz flip-flop is used. The 12 MHz clock can be inverted using the out_clk_inv field.

5. gated

   One flip-flop clocked by a generated gated clock, selected by the out_gated_clk field. The selected gated clock can be inverted by the out_clk_inv field.

6. ext

   One flip-flop clocked by an external (asynchronous) signal selected by field out_clk_ext. The ext_clk can be inverted by the out_clk_inv field.

13.16.2.4.2 out_clk_ext

Register field to select an asynchronous GIO (taken before SAP_in) to use as an external clock or to select a timer. Only used when out_clk_sel is set to ext or tmr.

The different asynchronous GIO which can be used as clock are:

gio1, gio5, gio6, gio7, gio13, gio18, gio21, gio29

The different timers that can be used are:

timer_grp0_timer[2], timer_grp1_timer[2], timer_grp2_timer[2], timer_grp3_timer[2]

13.16.2.4.3 out_gated_clk

Register field used to select one of the four generated gated clocks, see 13.16.2.1. Only used when out_clk_sel is set to gated.

13.16.2.4.4 out_clk_inv

The out_clk_inv field gives the opportunity to invert the 12 MHz, external or gated clock. This enables output clocking on negative clock edge. Only used when out_clk_sel is set to clk12, ext or gated.

13.16.2.4.5 out_logic

Selects if the signal will be inverted or not.

13.16.2.5 GIO output enables

There is one output enable for each GIO out signal. As for the GIO out signals the OE:s are controlled by the rw_gio vector registers. Each GIO OE signal can be configured according to figure 13.41 (figure shows GIO_oe[0]).

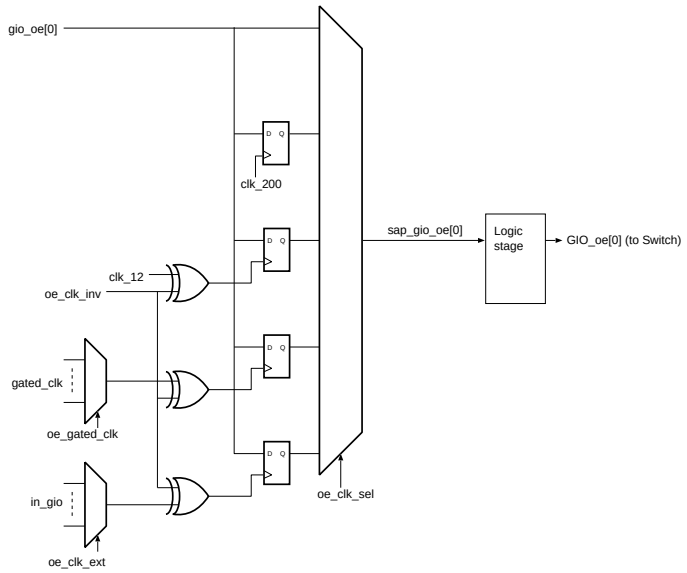


Figure 13.41: *SAP_out, GIO output enable configuration, GIO_oe 0*

In the following sub-sections the register fields of the rw_gio register which are used to select output clocking of the GIO OE:s are described.

13.16.2.5.1 oe_clk_sel

This register field is used to select output clocking type. Each OE can be clocked out in five different ways:

1. none

   No output clocking. Signals do not pass any extra flip-flops.

2. clk200

   One 200 MHz flip-flop is used.

3. clk12

   One 12 MHz flip-flop is used. The 12 MHz clock can be inverted by the oe_clk_inv field.

4. gated

   One flip-flop clocked by a generated gated clock, selected by oe_gated_clk. The selected gated clock can be inverted by the oe_clk_inv field.

5. ext

   One flip-flop clocked by an external (asynchronous) signal selected by oe_clk_ext. The ext_clk can be inverted by the oe_clk_inv field.

13.16.2.5.2 oe_clk_ext

Register field to select an asynchronous GIO (taken before SAP_in) to use as external clock. Only used when oe_clk_sel is set to ext.

The different asynchronous GIO which can be used as clock are:

gio5, gio13, gio21, gio29

13.16.2.5.3 oe_gated_clk

Register field used to select one of the four generated gated clocks, see 13.16.2.1. Only used when oe_clk_sel is set to gated.

13.16.2.5.4 oe_clk_inv

The oe_clk_inv field gives the opportunity to invert the 12 MHz, external or gated clock. This enables output clocking on negative clock edge. Only used when oe_clk_sel is set to clk12, ext or gated.

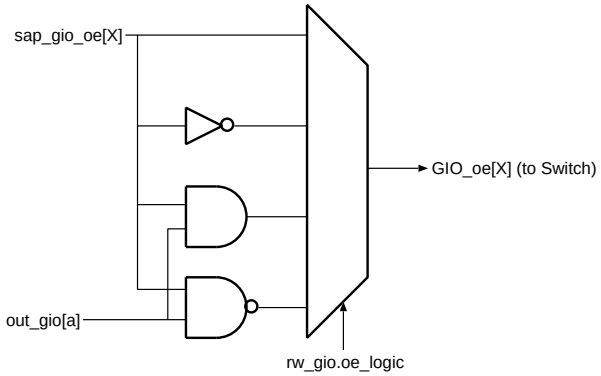


Figure 13.42: *SAP_out, GIO output enable logic stage*

13.16.2.5.5 oe_logic

Configures the logic stage for the OE. The OE can be inverted, and:ed or nand:ed. See figure 13.42.

Where X and a can be read from the following table:

X	a
31:28	4
27:24	0
23:16	4
15:8	0
7:5	4
4:1	0
0	4

Chapter 14

Memory Arbiter

14.1 References

Reference	Description
[CMB]	The Cache Memory Book, second edition, Jim Handy, Academic Press 1998, ISBN 0-12-322980-4
[REGS]	Memory arbiter registers
[MACROS]	Register macros for C programming
[REGS_BP]	Memory arbiter breakpoint registers
[MACROS_BP]	Breakpoint register macros for C programming

Table 14.1: *References*

14.2 Definitions

Term	Description
Client	An entity that requests access to memory such as a CPU or a DMA channel
Arbiter	A unit that sequences conflicting accesses to a memory according to some arbitration scheme
Memory Crossbar	The crossbar that routes accesses from clients to the requested memory

Table 14.2: *Definitions*

14.3 Overview

The memory arbiter crossbar gives multiple Clients access to multiple memories and supplies cache coherence using the MESI protocol.

The memory arbiter also implements address breakpoints. These breakpoints may be used to debug e.g. DMA behavior.

The clients are:

dma0	DMA channel 0
dma1	DMA channel 1
dma2	DMA channel 2
dma3	DMA channel 3
dma4	DMA channel 4
dma5	DMA channel 5
dma6	DMA channel 6
dma7	DMA channel 7
dma8	DMA channel 8
dma9	DMA channel 9
cpui	CPU instruction cache
cpud	CPU data cache
iop	I/O processor
slave	External memory bus slave mode

The memories are:

int	Internal 128KB RAM and 8KB ROM
ext	External memory
regs	Mode registers

14.4 Functional description

14.4.1 Memory arbitration scheme

The order in which client requests to access a specific memory are processed is decided by a memory arbitration scheme. Here a slot allocation based arbitration scheme is used. This means that the arbiter for each memory has a vector of slot owners, i.e. each slot is owned by a specific client. The arbiter state is the current position in this vector. The position is moved one step for each access made. In each state (or slot), the owner of the slot has the right to access the memory if it wants to. If a client does not use its slot, another client that wants to access the memory is chosen using a fixed priority scheme where ”client 0” has highest priority. The allocation vector is programmable to enable allocation of different bandwidths to different clients. For example,an allocation vector looking like this:

```
0 1 2 1 3 1 0 1 2 1 3 1 0 ...
```

allocates half the bandwidth to client 1, and 1/6 of the bandwidth to each of clients 0, 2 and 3. In other words, client 1 has the highest priority in every other slot while the other clients have the highest priority in every 6 slots.

The client numbers for the different clients, and the sizes of the allocation vectors can be found in [REGS].

14.4.2 Cache coherence

A mechanism making sure that all accesses to memory read or write the proper data in the presence of one or more caches is called a cache coherence protocol. The protocol used here is called MESI. More information about this can be found in e.g. [CMB].

Normally, this mechanism is invisible and exists for the convenience of the programmer. It gives the programmer the illusion of a single shared coherent memory.

14.4.3 Breakpoints

The memory crossbar contains ”global” breakpoints. They are global in the sense that they may break on accesses to memory performed by any client, CPUs, as well as DMA controllers. The breakpoints are specified as an address range, a set of access types (read, write, read exclusive etc), and a set of clients. When a breakpoint triggers, an interrupt may be generated, and the client triggering the breakpoint may be stopped. Information about the access causing a breakpoint to trigger is saved and may be read by the CPU. This entire process is controlled through mode registers.

This process can also be seen in the diagram below. The logic for one breakpoint and the common logic for interrupts and client stopping is shown:

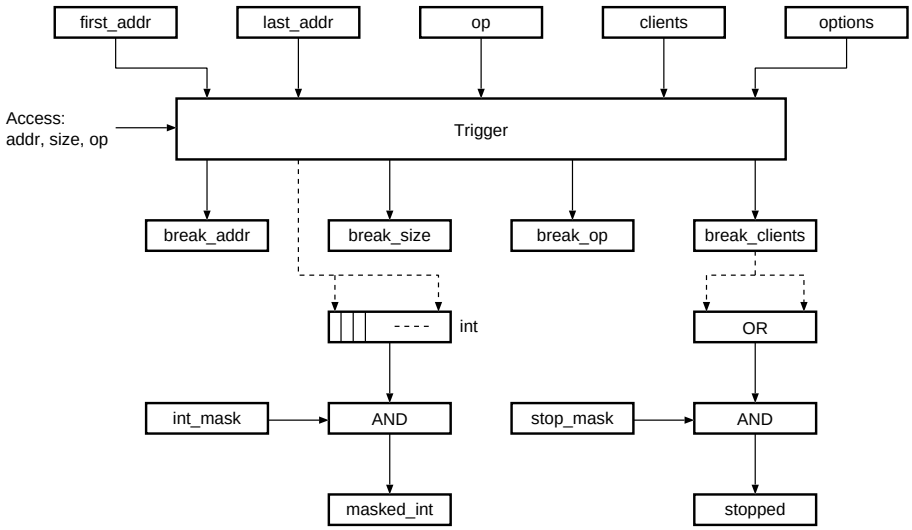


Figure 14.1: *Logic for one beakpoint and interrupts*

14.4.3.1 Setting up breakpoints

A breakpoint is set up by specifying the following:

Item	Description

Address range	Address range in which the breakpoint shall trigger. This is specified in the rw_first_addr and rw_last_addr registers of each breakpoint.
Access types	Access types on which the breakpoint shall trigger. This is specified in the rw_op register of each breakpoint. The breakpoint is turned off by specifying no access types.
Clients	Clients from which an access triggering the breakpoint may come. This is specified in the rw_clients register of each breakpoint.
Wrap	If set the breakpoint will trigger if any of the addresses covered by an access is greater or equal to rw_first_addr OR less or equal to rw_last_addr. If not set the breakpoint will trigger if any of the addresses covered by an access is greater or equal to rw_first_addr AND less or equal to rw_last_addr.

Table 14.5: *Breakpoint Setup*

The following table lists shows some setup examples:

Break on	first_addr	last_addr	wrap	op	clients
reads to 0x00000000-0x0000fff4 by any client	0x00000000	0x0000fff4	0	rd rd_excl us_rd us_rd_excl	all
any access to 0x0af4ebc3	0x0af4ebc3	0x0af4ebc3	0	all	all
any access outside 0x0af4ebc3	0x0af4ebc4	0x0af4ebc2	1	all	all
any access by client cli3	0x00000000	0xffffffff	0	all	cli3
ordinary writes outside 0x43000000-0x4400000 by clients cli4-cli7	0x44000001	0x42ffffff	1	wr	cli4- cli7

Table 14.6: *Breakpoint Setup Examples*

14.4.3.2 Breakpoint status

When a breakpoint is triggered, the following status is saved:

Status Item	Description
Address	Address of the first access triggering the breakpoint. This is saved in the r_brk_addr register of each breakpoint.
Size	Size of the first access triggering the breakpoint. This is saved in the r_brk_size register of each breakpoint.
Access type	Access type of the first access triggering the breakpoint. This is saved in the r_brk_op register of each breakpoint.
Clients	Clients which have made accesses triggering the breakpoint. This is saved in the r_brk_clients register of each breakpoint.
First Client	Client that made the first accesses triggering the breakpoint. This is saved in the r_brk_first_client register of each breakpoint.

Table 14.7: *Breakpoint Status*

14.4.3.3 Acknowledging a breakpoint

When a breakpoint has been triggered, it has to be acknowledged (or reset) to be able to be triggered on a new access. This can be done in two ways. Either write anything

to the breakpoints rw_ack register, or write to the corresponding bit in the rw_ack_intr register.

14.4.3.4 Interrupts

When a breakpoint is triggered, an interrupt may be generated. This is controlled by the interrupt registers of the memory arbiter crossbar.

Enabling and disabling of interrupts for each breakpoint is done in the rw_intr_mask register. The r_intr register tells which breakpoints have been triggered. The r_masked_intr register tells which breakpoints are generating interrupts. Interrupts may be acknowledged in rw_ack_intr (or by writing to the rw_ack register of the corresponding breakpoint).

14.4.3.5 Stopping clients

It is possible to stop further accesses from a client that has triggered a breakpoint. This feature is enabled or disabled in the rw_stop_mask register. In the r_stopped register the currently stopped clients are listed.

14.4.3.5.1 Writes

When stopping is enabled for a client, the memory access triggering the breakpoint is not stopped, only the following accesses are stopped. A write triggering the breakpoint is thus not stopped, and the address written will contain the newly written value and not the old value as one might have expected. The reason is that stopping the client before the write has taken effect would degrade memory access performance in general.

14.5 Software interface

The memory arbiter crossbar is configured through a set of registers, see [REGS] and [REGS_BP]. for details. To access the registers, fields and register constants from a C program, a set of macros is defined in [MACROS] and [MACROS_BP].

14.5.1 Allocation of arbitration slots

14.5.1.1 Registers

The arbitration slot owners are programmed by writing to registers in the memory arbiter crossbar. Each memory arbiter has its own vector register containing its arbitration slots. The name of the register is:

```
rw_<mem-name>_slots
```

where <mem-name> is the name of the memory. The default value of the slots register is 0. In other words, ”client 0” has the right to all slots after reset. For example, the memory ’ext’ having 64 slots has a vector register named ’rw_ext_slots’ of length 64 for setting the slots. See [REGS] for a detailed description of all registers.

14.5.1.2 Bandwidth versus latency

When specifying the slot allocation vector for a memory, the latency requirements of each client should be considered. For example, while the following two allocation vectors:

V1: 0 0 0 0 1 1 1 1

V2: 0 1 0 1 0 1 0 1

are equivalent regarding bandwidth, they are not equivalent in regards to latency. The maximum latency for each client is more than two times larger for V1 than for V2 (For V1 the maximum latency is almost 6 slots, for V2 it is almost 3).

14.5.1.3 Examples

14.5.1.3.1 Setting an allocation vector

- **Example:** Set the owner of one slot for memory ’ext’

```
void mem_ext_set_slot_owner( unsigned      slot_no,
                             unsigned char owner ) {
  reg_marb_rw_ext_slots r = { 0 };
  r.owner = owner;
  REG_WR_VECT( marb, regi_memarb, rw_ext_slots, slot_no, r );
}
```

- **Example:** Set all slot owners for memory ’ext’

```
void mem_ext_set_slot_vector( unsigned char *vector ) {
  int i;
  for( i = 0; i < regk_marb_rw_ext_slots_size; i++ )
    mem_ext_set_slot_owner( i, vector[i] );
}

unsigned char mem_ext_slot_vector[] = {
  regk_marb_cpu,
  regk_marb_dma0,
  regk_marb_dma1,
  regk_marb_dma2,
  regk_marb_dma0,
  regk_marb_dma3,
  regk_marb_dma4,
  regk_marb_dma0,
```

```
    regk_marb_dma5,
    regk_marb_dma6,
    ...
  };

  ...
    mem_ext_set_slot_vector( mem_ext_slot_vector );
  ...
```

14.5.2 Breakpoints

Each breakpoint has its own set of registers. See [REGS_BP] for a detailed description of these.

14.5.2.1 Setting up a breakpoint

```
typedef struct {
  unsigned                first_addr;
  unsigned                last_addr;
  reg_marb_bp_rw_op       op;
  reg_marb_bp_rw_clients  clients;
  reg_marb_bp_rw_options  options;
} bp_desc;


void bp_set( unsigned bp_ptr,     // Pointer to registers for a bp
             bp_desc  *bp ) {     // Breakpoint data

  reg_marb_bp_rw_op  r_op = { 0 };
  REG_WR( marb_bp, bp_ptr, rw_op, r_op );  // turn bp off
  REG_WR( marb_bp, bp_ptr, rw_ack, 0 );    // reset bp for new trig

  REG_WR( marb_bp, bp_ptr, rw_first_addr, bp->first_addr );
  REG_WR( marb_bp, bp_ptr, rw_last_addr,  bp->last_addr );
  REG_WR( marb_bp, bp_ptr, rw_clients,    bp->clients );
  REG_WR( marb_bp, bp_ptr, rw_options,    bp->options );

  REG_WR( marb_bp, bp_ptr, rw_op,         bp->op );  // turn bp on
}
```

14.5.2.2 To tell if a breakpoint has been triggered

- For a fixed breakpoint

```
reg_marb_r_intr i = REG_RD( marb, regi_marb, r_intr );
if( i.bp3 ) {
  // bp3 has been triggered
}
```

- Using breakpoint number

```
int bp_triggered( int bp_no ) {
  int t = REG_RD_INT( marb, regi_marb, r_intr );
  return ( t >> bp_no ) == 1;
}
```

- Using breakpoint register pointer

```
int bp_triggered( unsigned bp_ptr ) {
  int c = REG_RD_INT( marb_bp, bp_ptr, r_brk_clients );
  return c != 0;
}
```

14.5.2.3 Get information from a triggered breakpoint

```
typedef struct {
  unsigned                    addr;
  unsigned                    size;
  reg_memarb_bp_r_brk_op      op;
  reg_memarb_bp_r_brk_clients clients;
} bp_data;

void bp_info( unsigned bp_ptr,
              bp_data  *info ) {
  // read clients first, as this tells if the bp has been triggered
  // reading it after e.g. the address may give a false break address
  info->clients = REG_RD( marb_bp, bp_ptr, r_brk_clients );
  info->addr    = REG_RD( marb_bp, bp_ptr, r_brk_addr );
  info->size    = REG_RD( marb_bp, bp_ptr, r_brk_size );
  info->op      = REG_RD( marb_bp, bp_ptr, r_brk_op );
}
```

14.5.2.4 Reset a breakpoint

There are two almost equivalent ways of resetting a breakpoint:

1. Using the breakpoints ”ack” register:

   ```
   REG_WR( marb_bp, regi_marb_bp0, rw_ack, 0 );  // reset bp0
   ```

2. Using the common ”ack_intr” register:

   ```
   reg_marb_rw_ack_intr r = { .bp0 = regk_marb_yes,   // reset bp0 and bp4
                              .bp4 = regk_marb_yes };
   REG_WR( marb, regi_marb, rw_ack_intr, r );
   ```

The first way is useful for resetting a single breakpoint without having to access the common interrupt control registers. The second way is useful for resetting more than one breakpoint at a time.

14.5.2.5 Stopping clients

The possibility to stop clients after they have triggered a breakpoint should be used with care. Stopping a caching client is generally not a good idea. It may stop the entire system as a result of blocked cache coherence writes that were supposed to be carried out by the stopped client.

14.5.2.5.1 Examples

- **Example:** Enable stopping of client dma0 on breakpoint trig

```
reg_marb_rw_stop_mask r = REG_RD( marb, regi_marb, rw_stop_mask );
r.dma0 = regk_marb_yes;
REG_WR( marb, regi_marb, rw_stop_mask, r );
```

- **Example:** Tell if client dma0 has been stopped or not

```
reg_marb_r_stopped r = REG_RD( marb, regi_marb, r_stopped );
if( r.dma0 == regk_marb_yes ) {
  // dma0 has been stopped
}
```

14.5.3 Cache coherence

14.5.3.1 Cache coherence considerations

The cache coherence mechanism exists for the convenience of the programmer. It gives the programmer the illusion of a single shared coherent memory. However, this same mechanism may seriously degrade performance of the system if triggered too often. A bit of knowledge on how to avoid this is thus in place.

A bit simplified the mechanism works as follows:

Consider an access to main memory from a client (C):

1. C requests access to memory.
2. The request is granted.
3. All caches in the system check if they have cached the requested data. If a cache has cached the data and also modified it inside the cache:

   3.1. The modified data is written to main memory before the access from C may continue. (The protocol makes sure only one cache at a time may have a modified copy of the same data).

4. C's access is completed.

In the sequence above, step 3 infers no penalty on the access time as it is done in parallel with the actual memory access. Step 3.1, on the other hand, does require a lot of time if it has to be done. The total sequence does, in these cases, take about 3-4 times longer than without step 3.1.

As a result, ensure that step 3.1 above rarely occurs. In practice this means that one should take care when sharing data between a caching client (like the CPU) and other (caching or non-caching) clients (like a DMA channel). Below is a list of things to remember:

- Don't share data unless necessary. No sharing means no conflicts for the cache coherence protocol to handle.
- Make the "hand over" of data between clients infrequent.
- Avoid false sharing. If two clients are using different parts of a memory block that fit into the same cache line, unnecessary cache coherence actions may take place.
- Avoid caching rarely used shared data. For example, use hardware (memory to memory DMA or alike) for copying blocks of data instead of letting the CPU do it (via the cache).

14.5.3.2 Controlling cache coherency

Though cache coherence normally should be turned on, it it possible to turn it off in a couple of ways. One way is to force clients into only making accesses that are not snooped by the cache. This is done through the rw_no_snoop register. The other way is to tell caches to stop snooping accesses. This is done through the rw_no_snoop_rq register. See [REGS] for details.

14.5.3.3 Examples

14.5.3.3.1 Telling a client to avoid snooping

- **Example:** Make dma0 only use unsnooped accesses

```
reg_marb_rw_no_snoop r = REG_RD( marb, regi_marb, rw_no_snoop );
r.dma0 = regk_marb_yes;
REG_WR( marb, regi_marb, rw_no_snoop, r );
```

14.5.3.3.2 Telling a cache not to snoop

- **Example:** Make cpud's cache not snoop

```
reg_marb_rw_no_snoop_rq r = REG_RD( marb, regi_marb, rw_no_snoop_rq );
r.cpud = regk_marb_yes;
REG_WR( marb, regi_marb, rw_no_snoop_rq, r );
```

Chapter 15

Real Time Trace

15.1 References

Reference	Description
[MREG]	Mode registers, chapter 25.39
[MACROS]	Register macros for C programming: http://developer.axis.com
[1149]	IEEE Std 1149.1-2001
[CPU]	CPU, chapter 2
[PINS]	Pinout, chapter 16

15.2 Definitions

Term	Description
TAP	Test Access Port (The IEEE 1149.1 access port)
Guru Mode	An execution mode in the CRIS v32 CPU intended for debugging. See 2.

15.3 Overview

The real time trace block handles two software debug related interfaces. One is a real time trace port that enables tracing of the CPUs program counter in real time. The other is control of debug functionality through a register in the IEEE 1149 TAP controller.

15.4 Functional description

15.4.1 Real time tracing

The real time trace port outputs messages that can be used to reconstruct the execution sequence of the program running on the internal CPU. After the messages has been sampled by external logic, the execution sequence can be reconstructed using the messages and the object code of the traced program.

15.4.1.1 Configuration

Real time tracing is turned on/off and configured in the mode register rw_cfg. See 25.39. This register is also writable through the TAP. See section 15.4.3 for details.

15.4.1.2 PC tracing

The main type of tracing available is tracking of the CPUs program counter (PC). This tracing comes in two flavors: Lossy and lossless.

As the bandwidth of the trace port is limited, it is not always possible to track the PC at full speed at all times. When the trace logic can't keep up it is called a trace overflow. A trace overflow can either be handled (lossy tracing) or avoided (lossless tracing).

In lossy tracing, trace overflows are detected and signalled by an error message on the trace port. Then a synchronization message is output to get the tracing back on track. In this process a piece of the program flow is lost.

In lossless tracing, the CPU is stopped (stalled) when needed to avoid trace overflows. This will in most cases make the CPU run somewhat slower. Observe that if watchpoint and/or ownership tracing is enabled, see 15.4.1.3 and 15.4.1.4, there is a small risk of trace overflows even when lossless tracing is used. This is because the trace bandwidth is shared between the different trace types, and PC tracing does not consider the other trace types when avoiding overflows. Overflows in lossless mode are signaled in the same way as in lossy mode.

15.4.1.2.1 Overflow situations

Overflows occur when the trace data FIFO is full and can't hold new trace data from the CPU.

In lossy tracing mode, overflows typically occurs when there is a lot of jumps with dynamic target addresses within a short amount of time. In these cases the amount of trace data generated is larger than what can be output and stored by the FIFO.

In lossless tracing mode, overflows only occurs when there are a lot of PC tracing messages simultaneously with watchpoint and/or ownership messages. When the amount of free space in the trace data FIFO is less than a certain level the CPU is stopped. As it takes a few cycles for the CPU to stop, this level is adjusted to the worst case amount of trace data that can be generated by the CPU during the time it takes for it to stop. If

ownership or watchpoint messages are generated during the stopping of the CPU, there is a possibility that the amount of free space in the FIFO isn't enough, and an overflow occurs.

15.4.1.3 Watchpoint tracing

Breakpoints in the CPU may be configured to generate watchpoint events (as well as generating breakpoint exceptions, see 2). These watchpoint events may be configured to generate messages on the real time trace port. The watchpoint messages may be used by external logic to e.g. start and/or stop storing of the trace data.

15.4.1.4 Ownership tracing

Process ownership in the CPU is assumed to be indicated by the value of bit 0-7 in the PID register. Whenever this register changes value it is possible to get a message on the real time trace port. These messages may be used to keep track of which process is currently running.

15.4.1.5 Starting and stopping tracing

Tracing may be started and stopped in two ways. Manually by writing to the rw_cfg register, or by the use of CPU watchpoints. In the latter case one set of watchpoints may be configured to start tracing and another to stop tracing. This is all done in the rw_cfg register. It is not possible to use the same watchpoint for both starting and stopping the tracing.

15.4.2 Real time trace messages

Here follows a description of the messages that may be output on the real time trace port. The port signals are defined in section 15.5.2.

15.4.2.1 Message start and end

Messages start when the **mseo_n** signal is set to 1. When the **mseo_n** signal is 1, the **mdo** signals contain the message id number, which indicates the message type. A message ends either when a new message is started, or as described for each message type (e.g. when the maximum message length has been sent). If a message ends, and there are no new messages to send at the moment, **mdo** and **mseo_n** will be 0 until there is a new message to send.

15.4.2.2 Message types

This section describes the different messages output on the real time trace port. An overview of the messages is listed in the table below.

message	id	description
owner	2	process ownership change
sjmp	3	jumps with static target address
djmp	4	jump with dynamic target address
error	8	error message
sync	9	synchronization, output full PC
exception	11	jump to exception routine
wp	15	watchpoint trigger

Table 15.3: *Message types*

15.4.2.2.1 owner - process ownership change

id number: 2 (0x2)

Tells that the 8 least significant bits in the PID register in the CPU have changed. If the PID register is updated, and the 8 least significant bits are unchanged, no process ownership change message will be generated.

The id number is followed by the 8 least significant bits of the PID register.

The message vector ends when the new PID value has been sent. I.e. the message length is fixed to 3 nibbles.

- Example:

```
mseo_n:  1 0 0
mdo:     2 e 3
         = ===
```

This message tells that the least significant byte of PID has changed its value to 0x3e.

15.4.2.2.2 sjmp - jumps with static target address

id number: 3 (0x3)

Outputs a vector of instruction word counts between taken branches/jumps with static target addresses. Each vector element is 8 bits (2 nibbles).

The message vector ends when an instruction count is zero or when a new message starts.

- Example:

```
mseo_n:  1 0 0 0 0 0 0 0 0
mdo:     3 7 0 e 1 2 0 0 0
         = === === === ===
```

Here the vector is 7,1e,2. This means that since the last message, the following has happened:

- 7 instruction words have been executed consecutively.
- A jump or branch with static target address has been taken.
- 30 (0x1e) instruction words have been executed consecutively.
- A jump or branch with static target address has been taken.
- 2 instruction words have been executed consecutively.
- A jump or branch with static target address has been taken.

15.4.2.2.3 djmp - jump with dynamic target address

id number: 4 (0x4)

Outputs an instruction count of consecutively executed instructions followed by a jump target address. The instruction count is encoded with 8 bits (2 nibbles). The address is encoded as: Calculated PC after last message XOR-ed with the target address. This is done to reduce the number of bits needed to encode the address.

The message vector ends when 32 bits of the encoded target address have been sent. I.e. after the maximum message length (11 nibbles), or when a new message starts.

- Example:

```
mseo_n:  1 0 0 0 0 0
mdo:     4 6 0 8 a 1
         = === =====
```

This means that since the last message, 6 instruction words have been executed consecutively. Then a jump with non-static target address has been executed. If the calculated PC after the last message is 333aab02, then the target address is 333aab02 ^ 1a80 == 333ab182.

15.4.2.2.4 error

id number: 8 (0x8)

Outputs an error code. The error code is 8 bits (2 nibbles).

Error codes:

1 Program trace overflow. The internal trace buffer has been filled, and messages have been discarded. This message will be followed by a sync message.

3 Program trace overflow where at least one watchpoint message has been lost. The internal trace buffer has been filled, and messages have been discarded. This message will be followed by a sync message.

The message vector ends when the error code has been sent. I.e the message length is fixed to 3 nibbles.

- Example:

```
mseo_n:  1 0 0
mdo:     8 1 0
         = ===
```

This outputs error code 1, program trace overflow.

15.4.2.2.5 sync - synchronization

id number: 9 (0x9)

Outputs the PID and the full PC. This message is sent in the following cases:

- After the trace logic has been turned on
- After an error message
- When there is more than the maximum number of consecutively executed instruction words (255).
- When there has been more than about 250-300 number of PC trace messages since the last sync.
- When the **evti_n** input signal has been asserted.

The message vector ends when the entire PC has been output (8 nibbles), or when a new message starts. If the PC in the message consists of less than 8 nibbles, it shall be zero extended to 32 bits.

- Example:

```
mseo_n:  1 0 0 0 0 0 0 0 0
mdo:     9 1 4 2 f 3 7 a 1
         = === ===========
```

Says that PID == 0x41 and PC == 001a73f2.

15.4.2.2.6 exception - jump to exception routine

id number: 11 (0xb)

Outputs an instruction count of consecutively executed instructions followed by an absolute jump target address.

The message vector ends when 32 bits of the encoded target address have been sent. I.e. the maximum message length (11 nibbles), or when a new message starts. If the address in the message consists of less than 8 nibbles, it shall be zero extended to 32 bits.

- Example:

```
mseo_n:  1 0 0 0 0 0
mdo:     b 6 0 8 a 1
         = === =====
```

A jump to an exception routine at 0x1a8.

15.4.2.2.7 wp - watchpoint trigger

id number: 15 (0xf)

Indicates that one or more watchpoints has been triggered. Each watchpoint is represented by one bit in an 8 bit vector. If the bit is set the corresponding watchpoint was triggered.

The message vector ends when the watchpoint vector has been sent. I.e. the message length is fixed to 3 nibbles.

- Example:

```
mseo_n:  1 0 0
mdo:     f 4 0
         = ===
```

This message indicates that watchpoint 2 was triggered.

15.4.3 TAP controller interface

Debug functionality can be accessed by an external client via a data register in the on chip TAP controller. This functionality includes controlling the real time tracing, as well as doing more traditional debugging with breakpoints and single step by communicating with debug software in on chip ROM. More about this in section 15.6.

15.5 Hardware interface

15.5.1 TAP interface

Is described in [1149]. The pinout is described in 16.

15.5.2 Real time trace interface

The real time trace interface is a synchronous interface where output data is qualified by a clock. It consists of the following signals:

Signal	Direction	Description
mcko	Output	Trace data clock
mdo[3:0]	Output	Trace data
mseo_n	Output	Trace message start
evti_n	Input	Event in

The data outputs, **mdo**[3:0] and **mseo_n**, are qualified by both edges of the clock, **mcko**. The only input, **evti_n**, is asynchronous to the clock.

The real time trace signals are mapped to physical chip pins as described in 16.

15.5.2.1 Timing

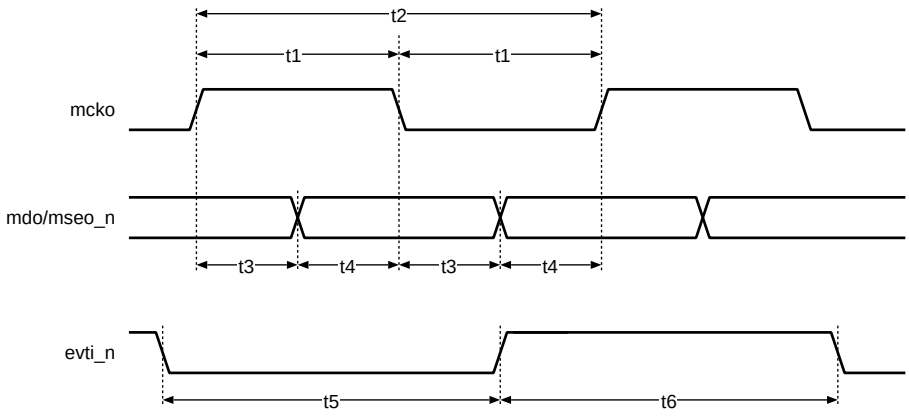


Figure 15.1: *Real time trace port timing*

Parameter	Name	Description	min	nom	max	unit
t1	tpw	Clock pulse width	4	5	6	ns
t2	tc	Clock cycle	-	10	-	ns
t3	th	Data hold time	1.0	-	-	ns
t4	tsu	Data setup time	1.5	-	-	ns
t5	tevpw	Event in pulse width	20	-	-	ns
t6	tevit	Event in inactive time	20	-	-	ns

Table 15.5: *Real time trace port timing*

15.6 Software interface

15.6.1 TAP debug data register

This is a 40 bit data register in the TAP controller. The register can be used to control the real time trace and the ROM resident on chip debugging functionality. The register is accessed through the chip TAP port.

The layout of this register is as follows:

39 - 8	7 - 4	3 - 0
data (32 bits)	sub-op (4bits)	op (4bits)

Bit 0 is the last bit shifted into the TAP controller via the TDI pin.

When the TAP debug data register is written, and the op field is not zero, the mode registers rw_tap_data and rw_tap_hdata are updated with the written value.

The following operations are defined:

op	sub-op	Description	Data
0x0 nop	-	No operation. rw_tap_data and rw_tap_hdata are not updated.	-
0x1 trcfg	-	Controls real time tracing by writing the rw_cfg register.	reg data
0x3 dbg	-	Make CPU enter Guru Mode. Must be acknowledged in the rw_tap_ctrl register.	-
0x4 dgbdi	See 25.39, rw_tap_hdata	Data to debug software in on-chip ROM.	data in
0x5 dgbdo	See 25.39, rw_tap_hdata	Data from debug software in on-chip ROM.	data out

Table 15.7: *Debug data register operation codes.*

From software running on the chip, this register is accessed through a set of memory mapped mode registers:

- rw_tap_ctrl
- r_tap_stat
- rw_tap_data
- rw_tap_hdata

These registers are described in 25.39. A set of macros for accessing these registers from a C program is defined in [MACROS].

Chapter 16

Pinout and pin multiplexing

16.1 References

[REGS]	Mode registers, chapter 25.38
[MACROS]	C macros, http://developer.axis.com

16.2 Overview

The package of the ETRAX FS is a 256 lead Plastic Ball Grid Array (PBGA). There are 80 configurable I/O pins, 125 fixed function signal pins and 51 power/ground pins.

All pins are listed in section 16.3 below. The connection alternatives for the configurable I/O pins are described in section 16.4. Configuration of the USB pins is described in 16.5.

16.3 Pinout

16.3.1 Power and ground pins

16.3.1.1 3.3 V Power pins

Name	Ball	Pad Type	Description
vdd33	A4	power	vdd for I/O ring
vdd33	A8	power	vdd for I/O ring
vdd33	A12	power	vdd for I/O ring
vdd33	A17	power	vdd for I/O ring
vdd33	D1	power	vdd for I/O ring
vdd33	D20	power	vdd for I/O ring
vdd33	H20	power	vdd for I/O ring

vdd33	J1	power	vdd for I/O ring
vdd33	M20	power	vdd for I/O ring
vdd33	N1	power	vdd for I/O ring
vdd33	U1	power	vdd for I/O ring
vdd33	U20	power	vdd for I/O ring
vdd33	Y4	power	vdd for I/O ring
vdd33	Y8	power	vdd for I/O ring
vdd33	Y12	power	vdd for I/O ring
vdd33	Y13	power	vdd for I/O ring
vdd33	Y17	power	vdd for I/O ring

Table 16.2: *3.3 V Power pins*

16.3.1.2 1.5 V Power pins

Name	Ball	Pad Type	Description
vdd15	D6	power	Core vdd
vdd15	D11	power	Core vdd
vdd15	D15	power	Core vdd
vdd15	F4	power	Core vdd
vdd15	F17	power	Core vdd
vdd15	K4	power	Core vdd
vdd15	L17	power	Core vdd
vdd15	R4	power	Core vdd
vdd15	R17	power	Core vdd
vdd15	U6	power	Core vdd
vdd15	U10	power	Core vdd
vdd15	U11	power	Core vdd
vdd15	U15	power	Core vdd
vdd15	V10	power	Core vdd

Table 16.3: *1.5 V Power pins*

16.3.1.3 Ground pins

Name	Ball	Pad Type	Description
vss	A1	power	General vss
vss	B1	power	General vss
vss	C3	power	General vss
vss	C18	power	General vss
vss	D4	power	General vss
vss	D8	power	General vss
vss	D13	power	General vss
vss	D17	power	General vss
vss	H4	power	General vss
vss	H17	power	General vss
vss	N4	power	General vss
vss	N17	power	General vss

vss	U4	power	General vss
vss	U8	power	General vss
vss	U13	power	General vss
vss	U17	power	General vss
vss	V3	power	General vss
vss	V11	power	General vss
vss	W11	power	General vss
vss	Y10	power	General vss

Table 16.4: *Ground pins*

16.3.2 Miscellaneous pins

Name	Ball	Pad Type	Description
clk	Y9	input	12 MHz clock input
rst_n	U9	input	Chip reset input
irq_n	B13	input	Interrupt request input
nmi_n	A14	input	Non maskable interrupt input
test	W9	input	Test mode select input (tie to Vss for normal operation)
plllpf	Y11	output	PLL loop filter output
phyrst_n	W3	output	Software controlled reset output

Table 16.5: *Miscellaneous pins*

16.3.3 Boot select pins

Name	Ball	Pad Type	Description
bs0	P4	bidir	Boot select 0/debug data out 0
bs1	R3	bidir	Boot select 1/debug data out 1
bs2	T2	bidir	Boot select 2/debug data out 2
bs3	T3	bidir	Boot select 3/debug data out 3
bs4	U2	bidir	Boot select 4/debug start
bs5	V1	bidir	Boot select 5/debug clk
bs6	T4	input	Boot select 6/debug data in

Table 16.6: *Boot select pins*

16.3.4 Test access port (TAP)

Name	Ball	Pad Type	Description
tck	W13	input	JTAG TAP clock input
tdi	V9	input	JTAG TAP data in
tdo	V18	output	JTAG TAP data out
tms	U12	input	JTAG TAP mode select
trst	W10	input	JTAG TAP reset input

Table 16.7: *Test access port pins*

16.3.5 Bus interface pins

16.3.5.1 Data bus pins

Name	Ball	Pad Type	Description
d0	F3	bidir	Data bit 0
d1	G4	bidir	Data bit 1
d2	F2	bidir	Data bit 2
d3	F1	bidir	Data bit 3
d4	G3	bidir	Data bit 4
d5	G2	bidir	Data bit 5
d6	G1	bidir	Data bit 6
d7	H3	bidir	Data bit 7
d8	H2	bidir	Data bit 8
d9	H1	bidir	Data bit 9
d10	J4	bidir	Data bit 10
d11	J3	bidir	Data bit 11
d12	J2	bidir	Data bit 12
d13	K2	bidir	Data bit 13
d14	K3	bidir	Data bit 14
d15	K1	bidir	Data bit 15
d16	L1	bidir	Data bit 16
d17	L2	bidir	Data bit 17
d18	L3	bidir	Data bit 18
d19	L4	bidir	Data bit 19
d20	M1	bidir	Data bit 20
d21	M2	bidir	Data bit 21
d22	M3	bidir	Data bit 22
d23	M4	bidir	Data bit 23
d24	N2	bidir	Data bit 24
d25	N3	bidir	Data bit 25
d26	P1	bidir	Data bit 26
d27	P2	bidir	Data bit 27
d28	R1	bidir	Data bit 28
d29	P3	bidir	Data bit 29
d30	R2	bidir	Data bit 30
d31	T1	bidir	Data bit 31

Table 16.8: *Data bus pins*

16.3.5.2 Address bus pins

Name	Ball	Pad Type	Description
a1	B10	bidir	Address bit 1/SDRAM dqm7
a2	C10	bidir	Address bit 2
a3	D10	bidir	Address bit 3
a4	A9	bidir	Address bit 4
a5	B9	bidir	Address bit 5
a6	C9	bidir	Address bit 6

a7	D9	bidir	Address bit 7
a8	B8	bidir	Address bit 8
a9	C8	bidir	Address bit 9
a10	A7	bidir	Address bit 10
a11	B7	bidir	Address bit 11
a12	A6	bidir	Address bit 12
a13	C7	bidir	Address bit 13
a14	B6	bidir	Address bit 14
a15	A5	bidir	Address bit 15
a16	D7	bidir	Address bit 16
a17	C6	bidir	Address bit 17/SDRAM bank address 0
a18	B5	bidir	Address bit 18/SDRAM bank address 1
a19	C5	bidir	Address bit 19/SDRAM dqm0
a20	B4	bidir	Address bit 20/SDRAM dqm1
a21	A3	bidir	Address bit 21/SDRAM dqm2
a22	D5	bidir	Address bit 22/SDRAM dqm3
a23	C4	bidir	Address bit 23/SDRAM write enable
a24	B3	bidir	Address bit 24/SDRAM cas
a25	B2	bidir	Address bit 25/SDRAM ras

Table 16.9: *Address bus pins*

16.3.5.3 Chip select pins

Chip select signals that have dedicated pins are listed in the table below. The **csp2_n**, **csp3_n**, **csp5_n** and **csp6_n** chip selects are multiplexed on configurable I/O pins, see 16.4.3.1.

Name	Ball	Pad Type	Description
csd0_n	E1	bidir	SDRAM chip select 0
csd1_n	E2	bidir	SDRAM chip select 1
cse0_n	A10	output	Flash/EPROM chip select 0
cse1_n	A11	output	Flash/EPROM chip select 1
csp0_n	B12	bidir	Peripheral chip select 0
csp1_n	C12	bidir	Peripheral chip select 1
csp4_n	D12	bidir	Peripheral chip select 4
csr0_n	C11	bidir	SRAM chip select 0
csr1_n	B11	bidir	SRAM chip select 1
css_n	A13	bidir	Slave chip select

Table 16.10: *Chip select pins*

16.3.5.4 Bus interface control pins

Name	Ball	Pad Type	Description
rd_n	E4	bidir	Read signal
wr0_n	A2	bidir	Write signal for byte 0/SDRAM dqm4
wr1_n	C2	bidir	Write signal for byte 1/SDRAM dqm5

wr2_n	D2	bidir	Write signal for byte 2/SDRAM dqm6
wr3_n	D3	bidir	Write signal for byte 3
wait_n	C13	input	Wait input
sdcke	E3	bidir	SDRAM clock enable
sdclk	C1	bidir	SDRAM clock output

Table 16.11: *Bus interface control pins*

16.3.5.5 External DMA/slave mode handshake pins

External DMA/slave mode hand shake signals that have dedicated pins are listed in the table below. The **hsh4 - hsh7** handshake signals are multiplexed on configurable I/O pins, see 16.4.3.1.

Name	Ball	Pad Type	Description
hsh0	A15	bidir	DMA/slave mode handshake signal 0
hsh1	B15	bidir	DMA/slave mode handshake signal 1
hsh2	C15	bidir	DMA/slave mode handshake signal 2
hsh3	A16	bidir	DMA/slave mode handshake signal 3

Table 16.12: *External DMA/slave mode handshake pins*

16.3.5.6 Bus arbitration pins

Name	Ball	Pad Type	Description
bg	C14	bidir	Bus grant input/output
brin	D14	input	Bus request input
brout	B14	bidir	Bus request output

Table 16.13: *Bus arbitration pins*

16.3.6 Ethernet interface 0

Name	Ball	Pad Type	Description
e0col	V6	input	Ethernet collision detect
e0crs	V8	input	Ethernet carrier sense
e0mdc	Y1	output	Ethernet management clock
e0mdio	Y2	bidir	Ethernet management data
e0phyclk	Y3	output	Ethernet transceiver clock/txer
e0rxclk	Y7	input	Ethernet receive clock
e0rxd0	W6	input	Ethernet receive data bit 0
e0rxd1	Y6	input	Ethernet receive data bit 1
e0rxd2	V7	input	Ethernet receive data bit 2
e0rxd3	W7	input	Ethernet receive data bit 3
e0rxdv	U7	input	Ethernet receive data valid
e0rxer	W8	input	Ethernet receive error
e0txclk	Y5	input	Ethernet transmit clock

e0txd0	V4	output	Ethernet transmit data bit 0
e0txd1	U5	output	Ethernet transmit data bit 1
e0txd2	V5	output	Ethernet transmit data bit 2
e0txd3	W5	output	Ethernet transmit data bit 3
e0txen	W4	output	Ethernet transmit enable

Table 16.14: *Ethernet interface 0 pins*

16.3.7 Asynchronous serial port 0

Name	Ball	Pad Type	Description
s0cts_n	U3	input	Asynchronous serial port 0 clear to send
s0rts_n	W2	output	Asynchronous serial port 0 request to send
s0rxd	V2	input	Asynchronous serial port 0 data in
s0txd	W1	output	Asynchronous serial port 0 data out

Table 16.15: *Asynchronous serial port 0 pins*

The external clock input signal **ext_clk** is shared between all asynchronous and synchronous serial ports and the timer module, see 16.4.3.4.1.

16.3.8 USB pins

Name	Ball	Pad Type	Description
u0vm	V12	bidir	USB minus in/out
u0vp	W12	bidir	USB plus in/out

Table 16.16: *USB pins*

16.3.9 Configurable I/O pins

16.3.9.1 Port pa

Name	Ball	Pad Type	Description
pa0	B16	bidir	Configurable I/O port pa bit 0
pa1	C16	bidir	Configurable I/O port pa bit 1
pa2	A18	bidir	Configurable I/O port pa bit 2
pa3	D16	bidir	Configurable I/O port pa bit 3
pa4	C17	bidir	Configurable I/O port pa bit 4
pa5	B17	bidir	Configurable I/O port pa bit 5
pa6	B18	bidir	Configurable I/O port pa bit 6
pa7	A19	bidir	Configurable I/O port pa bit 7

Table 16.17: *Configurable I/O port pa pins*

16.3.9.2 Port pb

Name	Ball	Pad Type	Description
pb0	A20	bidir	Configurable I/O port pb bit 0
pb1	B19	bidir	Configurable I/O port pb bit 1
pb2	B20	bidir	Configurable I/O port pb bit 2
pb3	C19	bidir	Configurable I/O port pb bit 3
pb4	D18	bidir	Configurable I/O port pb bit 4
pb5	E17	bidir	Configurable I/O port pb bit 5
pb6	C20	bidir	Configurable I/O port pb bit 6
pb7	D19	bidir	Configurable I/O port pb bit 7
pb8	E18	bidir	Configurable I/O port pb bit 8
pb9	E19	bidir	Configurable I/O port pb bit 9
pb10	F18	bidir	Configurable I/O port pb bit 10
pb11	G17	bidir	Configurable I/O port pb bit 11
pb12	E20	bidir	Configurable I/O port pb bit 12
pb13	F19	bidir	Configurable I/O port pb bit 13
pb14	G18	bidir	Configurable I/O port pb bit 14
pb15	F20	bidir	Configurable I/O port pb bit 15
pb16	G19	bidir	Configurable I/O port pb bit 16
pb17	G20	bidir	Configurable I/O port pb bit 17

Table 16.18: *Configurable I/O port pb pins*

16.3.9.3 Port pc

Name	Ball	Pad Type	Description
pc0	H18	bidir	Configurable I/O port pc bit 0
pc1	H19	bidir	Configurable I/O port pc bit 1
pc2	J17	bidir	Configurable I/O port pc bit 2
pc3	J18	bidir	Configurable I/O port pc bit 3
pc4	J19	bidir	Configurable I/O port pc bit 4
pc5	J20	bidir	Configurable I/O port pc bit 5
pc6	K17	bidir	Configurable I/O port pc bit 6
pc7	K18	bidir	Configurable I/O port pc bit 7
pc8	K19	bidir	Configurable I/O port pc bit 8
pc9	K20	bidir	Configurable I/O port pc bit 9
pc10	L20	bidir	Configurable I/O port pc bit 10
pc11	L18	bidir	Configurable I/O port pc bit 11
pc12	L19	bidir	Configurable I/O port pc bit 12
pc13	M19	bidir	Configurable I/O port pc bit 13
pc14	M18	bidir	Configurable I/O port pc bit 14
pc15	M17	bidir	Configurable I/O port pc bit 15
pc16	N20	bidir	Configurable I/O port pc bit 16
pc17	N19	bidir	Configurable I/O port pc bit 17

Table 16.19: *Configurable I/O port pc pins*

16.3.9.4 Port pd

Name	Ball	Pad Type	Description
pd0	N18	bidir	Configurable I/O port pd bit 0
pd1	P20	bidir	Configurable I/O port pd bit 1
pd2	P19	bidir	Configurable I/O port pd bit 2
pd3	P18	bidir	Configurable I/O port pd bit 3
pd4	R20	bidir	Configurable I/O port pd bit 4
pd5	R19	bidir	Configurable I/O port pd bit 5
pd6	P17	bidir	Configurable I/O port pd bit 6
pd7	R18	bidir	Configurable I/O port pd bit 7
pd8	T20	bidir	Configurable I/O port pd bit 8
pd9	T19	bidir	Configurable I/O port pd bit 9
pd10	T18	bidir	Configurable I/O port pd bit 10
pd11	V20	bidir	Configurable I/O port pd bit 11
pd12	T17	bidir	Configurable I/O port pe bit 12
pd13	U18	bidir	Configurable I/O port pd bit 13
pd14	U19	bidir	Configurable I/O port pd bit 14
pd15	V19	bidir	Configurable I/O port pd bit 15
pd16	W20	bidir	Configurable I/O port pd bit 16
pd17	Y20	bidir	Configurable I/O port pd bit 17

Table 16.20: *Configurable I/O port pd pins*

16.3.9.5 Port pe

Name	Ball	Pad Type	Description
pe0	W19	bidir	Configurable I/O port pe bit 0
pe1	Y19	bidir	Configurable I/O port pe bit 1
pe2	W18	bidir	Configurable I/O port pe bit 2
pe3	V17	bidir	Configurable I/O port pe bit 3
pe4	U16	bidir	Configurable I/O port pe bit 4
pe5	Y18	bidir	Configurable I/O port pe bit 5
pe6	W17	bidir	Configurable I/O port pe bit 6
pe7	V16	bidir	Configurable I/O port pe bit 7
pe8	W16	bidir	Configurable I/O port pe bit 8
pe9	V15	bidir	Configurable I/O port pe bit 9
pe10	U14	bidir	Configurable I/O port pe bit 10
pe11	Y16	bidir	Configurable I/O port pe bit 11
pe12	W15	bidir	Configurable I/O port pe bit 12
pe13	V14	bidir	Configurable I/O port pe bit 13
pe14	Y15	bidir	Configurable I/O port pe bit 14
pe15	W14	bidir	Configurable I/O port pe bit 15
pe16	Y14	bidir	Configurable I/O port pe bit 16
pe17	V13	bidir	Configurable I/O port pe bit 17

Table 16.21: *Configurable I/O port pe pins*

16.4 Multiplexing of configurable I/O pins

The configurable I/O pins are grouped in five groups, **pa** to **pe**. Port **pa** contains 8 pins, while **pb** to **pe** contain 18 pins each. Multiplexing between the different I/O alternatives is done by the pinmux block. The pinmux also controls the connection of the USB pins **u0vp** and **u0vm** to the I/O processor, see section 16.5.

Port **pa** is shared between General I/O and bus interface functions.

Ports **pb** to **pe** are shared between General I/O, fixed protocol I/O, and the I/O processor.

The pinmux is controlled through a set of registers, see 25.38 for details. To access the registers, fields and register constants from a C program, a set of macros is defined in [MACROS].

16.4.1 Overview

An overview of the pin mapping alternatives for the **pa** to **pe** ports is given in the figure below:

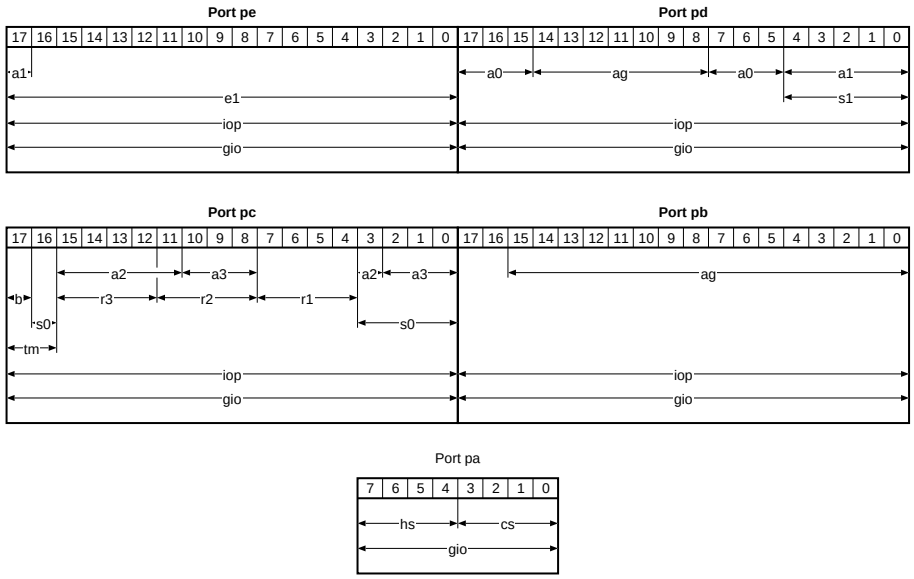


Figure 16.1: *Port pa to pe pin mapping alternatives*

The list below defines the abbreviations used in figure 16.1.

gio General I/O

iop I/O processor

hs External DMA/slave mode handshake pins **hsh4-hsh7**

cs Chip select pins **csp2_n**, **csp3_n**, **csp5_n** and **csp6_n**

r1 Asynchronous serial port 1

r2 Asynchronous serial port 2

r3 Asynchronous serial port 3

b External baud rate clock input

s0 Synchronous serial port 0

s1 Synchronous serial port 1

ag ATA general pins

a0 ATA bus 0 pins

a1 ATA bus 1 pins

a2 ATA bus 2 pins

a3 ATA bus 3 pins

e1 Ethernet interface 1

tm Timer

16.4.2 Principles for signal multiplexing

16.4.2.1 Input signals

Input signals from the configurable I/O pins are connected to the different I/O units in parallel, without any gating. This means that the I/O interface will always have access to its inputs, regardless of whether the I/O interface is selected in the pinmux configuration registers or not.

The only exception to this is that the input signals from the USB pins **u0vp** and **u0vm** to the I/O processor are multiplexed with a few signals from pins in port **pc** and **pe**, see section 16.5.

16.4.2.2 Output signals

The output signals are multiplexed using an AND-OR structure. The pinmux configuration registers have one or several configuration bits for each I/O interface. Each bit selects one or several signals and/or output enables from the I/O interface to control the corresponding pins.

The AND-OR structure will look like this:

If the pinmux registers are configured in a way that allows more than one I/O interface to control the same pin, the signals and output enables from the different I/O interfaces will be OR-ed together.

Pins that are not configured to be controlled by any I/O interface will have their outputs turned off.

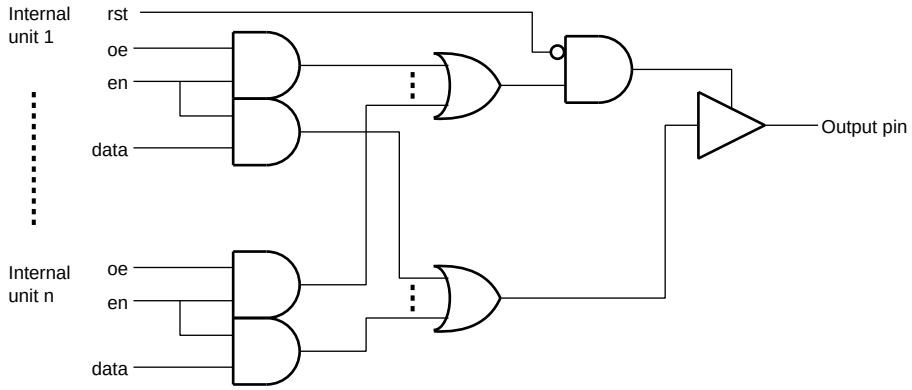


Figure 16.2: *AND-OR structure for output signals*

16.4.2.3 Reset behavior

All pinmux configuration registers have the default value 0. This results in that no I/O interfaces control any pins after a reset, and all pins will therefore have their outputs turned off. The pinmux will turn off all outputs immediately (asynchronously) at reset, and keep them turned off until the software enables them.

16.4.3 Pin mapping

16.4.3.1 Bus interface signals on port pa

Port **pa** is shared between General I/O and bus interface functions. Port **pa** is configured in rw_pa. Each pin is separately configurable. The bus interface functions mapped to port **pa** are shown in the tables below:

- Chip selects:

Chip select	Pin	Direction used
csp2_n	**pa0**	bidir
csp3_n	**pa1**	bidir
csp5_n	**pa2**	output
csp6_n	**pa3**	output

Table 16.22: *Chip selects shared with General I/O*

- External DMA/slave mode handshake signals:

Handshake signal	Pin	Direction used
hsh4	**pa4**	bidir
hsh5	**pa5**	bidir
hsh6	**pa6**	bidir
hsh7	**pa7**	bidir

Table 16.23: *Handshake signals shared with General I/O*

The multiplexing between the dreq, dack, tc_in, tc_out and sreq functions is done internally in the bus interface.

16.4.3.2 General I/O

The General I/O block has 80 I/O signals that can be connected to configurable I/O ports **pa** to **pe**. For the general I/O there are individual configuration bits for each I/O signal.

The General I/O signals are configured in rw_pa, rw_pb_gio, rw_pc_gio, rw_pd_gio and rw_pe_gio.

General I/O signals	Pins	Direction used
pa0 - pa7	**pa0 - pa7**	bidir
pb0 - pb17	**pb0 - pb17**	bidir
pc0 - pc17	**pc0 - pc17**	bidir
pd0 - pd17	**pd0 - pd17**	bidir
pe0 - pe17	**pe0 - pe17**	bidir

Table 16.24: *General I/O signals*

16.4.3.3 I/O processor

The I/O processor has 72 I/O signals that can be connected to configurable I/O ports **pb** to **pe**. For the I/O processor there are individual configuration bits for each I/O signal.

The I/O processor signals are configured in rw_pb_iop, rw_pc_iop, rw_pd_iop and rw_pe_iop.

I/O processor signals	Pins	Direction used
pb0 - pb17	**pb0 - pb17**	bidir
pc0 - pc17	**pc0 - pc17**	bidir
pd0 - pd17	**pd0 - pd17**	bidir
pe0 - pe17	**pe0 - pe17**	bidir

Table 16.25: *I/O processor signals*

16.4.3.4 Fixed protocol I/O blocks

The fixed protocol I/O block connections are configured in rw_hwprot.

16.4.3.4.1 Asynchronous serial ports

The asynchronous serial ports 1-3 are mapped to the configurable I/O port **pc** according to the tables below. Asynchronous serial port 0 shares the **ext_clk** input with the other ports, but has otherwise dedicated pins that are not controlled by the pinmux, see 16.3.7.

- Asynchronous serial port 1:

Serial port 1 signal	Pin	Used direction
txd	**pc4**	output
rts_n	**pc5**	output
rxd	**pc6**	input
cts_n	**pc7**	input

Table 16.26: *Asynchronous serial port 1 signals*

- Asynchronous serial port 2:

Serial port 2 signal	Pin	Used direction
txd	**pc8**	output
rts_n	**pc9**	output
rxd	**pc10**	input
cts_n	**pc11**	input

Table 16.27: *Asynchronous serial port 2 signals*

- Asynchronous serial port 3:

Serial port 3 signal	Pin	Used direction
txd	**pc12**	output
rts_n	**pc13**	output
rxd	**pc14**	input
cts_n	**pc15**	input

Table 16.28: *Asynchronous serial port 3 signals*

- General:

  The asynchronous serial ports share an external clock input with the synchronous serial ports and the timer block.

External clock signal	Pin	Used direction
ext_clk	**pc17**	input

Table 16.29: *External serial port clock signal*

16.4.3.4.2 Synchronous serial ports

The synchronous serial ports 0-1 are mapped to configurable I/O ports pc and pd according to the tables below.

- Synchronous serial port 0:

Serial port 0 signal	Pin	Used direction
data	**pc0**	bidir

clk	**pc1**	bidir
status	**pc2**	bidir
frame	**pc3**	bidir
din	**pc16**	input

Table 16.30: *Synchronous serial port 0 signals*

· Synchronous serial port 1:

Serial port 1 signal	Pin	Used direction
data	**pd0**	bidir
clk	**pd1**	bidir
status	**pd2**	bidir
frame	**pd3**	bidir
din	**pd4**	input

Table 16.31: *Synchronous serial port 1 signals*

· General:

The synchronous serial ports share an external clock input with the asynchronous serial ports and the timer block.

External clock signal	Pin	Used direction
ext_clk	**pc17**	input

Table 16.32: *External serial port clock signal*

16.4.3.4.3 ATA

The ATA interface is mapped to configurable I/O ports pb to pe according to the tables below. There is one configuration bit for each of the four ATA buses, and one bit for the signals that are common between the buses.

· General ATA signals:

ATA signal	Pin	Used direction
data0	**pb0**	bidir
data1	**pb1**	bidir
data2	**pb2**	bidir
data3	**pb3**	bidir
data4	**pb4**	bidir
data5	**pb5**	bidir
data6	**pb6**	bidir
data7	**pb7**	bidir
data8	**pb8**	bidir
data9	**pb9**	bidir
data10	**pb10**	bidir
data11	**pb11**	bidir

data12	**pb12**	bidir
data13	**pb13**	bidir
data14	**pb14**	bidir
data15	**pb15**	bidir
a0	**pd8**	output
a1	**pd9**	output
a2	**pd10**	output
cs0_n	**pd11**	output
cs1_n	**pd12**	output
ext_oe	**pd13**	output
reset_n	**pd14**	output

Table 16.33: *General ATA signals*

- Ata bus 0 signals:

ATA bus 0 signal	Pin	Used direction
dior0_n	**pd5**	output
diow0_n	**pd6**	output
dmack0_n	**pd7**	output
iordy0	**pd15**	input
dmarq0	**pd16**	input
intrq0	**pd17**	input

Table 16.34: *ATA bus 0 signals*

- Ata bus 1 signals:

ATA bus 1 signal	Pin	Used direction
dior1_n	**pd0**	output
diow1_n	**pd1**	output
dmack1_n	**pd2**	output
iordy1	**pe17**	input
dmarq1	**pd3**	input
intrq1	**pd4**	input

Table 16.35: *ATA bus 1 signals*

- Ata bus 2 signals:

ATA bus 2 signal	Pin	Used direction
dior2_n	**pc11**	output
diow2_n	**pc12**	output
dmack2_n	**pc13**	output
iordy2	**pc3**	input
dmarq2	**pc14**	input
intrq2	**pc15**	input

Table 16.36: *ATA bus 2 signals*

· Ata bus 3 signals:

ATA bus 3 signal	Pin	Used direction
dior3_n	**pc0**	output
diow3_n	**pc1**	output
dmack3_n	**pc8**	output
iordy3	**pc2**	input
dmarq3	**pc9**	input
intrq3	**pc10**	input

Table 16.37: *ATA bus 3 signals*

16.4.3.4.4 Ethernet interface 1

Ethernet interface 1 is mapped to configurable I/O port pe according to the tables below. There is one configuration bit for the transceiver management signals and one bit for the rest of the signals. Ethernet interface 0 has dedicated pins that are not controlled by the pinmux.

· Ethernet interface 1:

Ethernet 1 signal	Pin	Used direction
rxd0	**pe0**	input
rxd1	**pe1**	input
rxd2	**pe2**	input
rxd3	**pe3**	input
rxdv	**pe4**	input
rxer	**pe5**	input
col	**pe6**	input
crs	**pe7**	input
rxclk	**pe8**	input
txclk	**pe9**	input
phyclk	**pe10**	output
txen	**pe11**	output
txd0	**pe12**	output
txd1	**pe13**	output
txd2	**pe14**	output
txd3	**pe15**	output

Table 16.38: *Ethernet interface 1 signals*

· Ethernet interface 1 transceiver management:

Ethernet 1 signal	Pin	Used direction
mdc	**pe16**	output
mdio	**pe17**	bidir

Table 16.39: *Ethernet interface 1 transceiver management signals*

16.4.3.4.5 timer

The timer has one external clock input and one terminal count output, that are mapped to configurable I/O port pc according to the table below. The external clock input **ext** is shared with the asynchronous and synchronous serial ports.

Timer signal	Pin	Used direction
tc	**pc16**	output
ext	**pc17**	input

Table 16.40: *Timer signals*

16.5 USB pin mapping

The internal USB transceiver has two external pins, **u0vp** and **u0vm**, and seven internal signals.

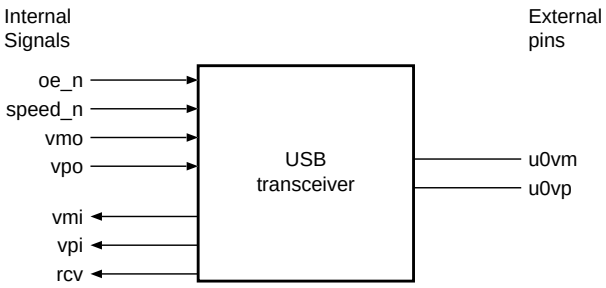


Figure 16.3: *USB pin mapping*

The pinmux can be configured to connect the internal signals of the USB transceiver to the I/O processor in two different ways, controlled by the en_usb0 and en_usb1 fields of the rw_usb_phy register.

- Mapping enabled by en_usb0:

Transceiver signal	I/O processor signal
oe_n	**pc12** output
speed_n	**pc13** output
vmo	**pc8** output
vpo	**pc9** output
vmi	**pc10** input
vpi	**pc11** input
rcv	**pc14** input

Table 16.41: *Mapping enabled by en_usb0.*

- Mapping enabled by en_usb1:

Transceiver signal	I/O processor signal
oe_n	**pe12** output
speed_n	**pe13** output
vmo	**pe8** output
vpo	**pe9** output
vmi	**pe10** input
vpi	**pe11** input
rcv	**pe14** input

Table 16.42: *Mapping enabled by en_usb1.*

The normal connections for the I/O processor outputs are not affected. I/O processor output signals that are configured to control the internal USB transceiver can be configured to go out on their normal pins in the pc or pe port in parallel.

If both en_usb0 and en_usb1 are enabled, the two I/O processor output signals for each transceiver signal will be OR-ed together, and the input signals from the transceiver will go to both I/O processor inputs. If none is enabled, the USB pins will be high-z.

Inputs from the internal USB transceiver to the I/O processor are multiplexed with the normal input signals from the pc and pe ports.

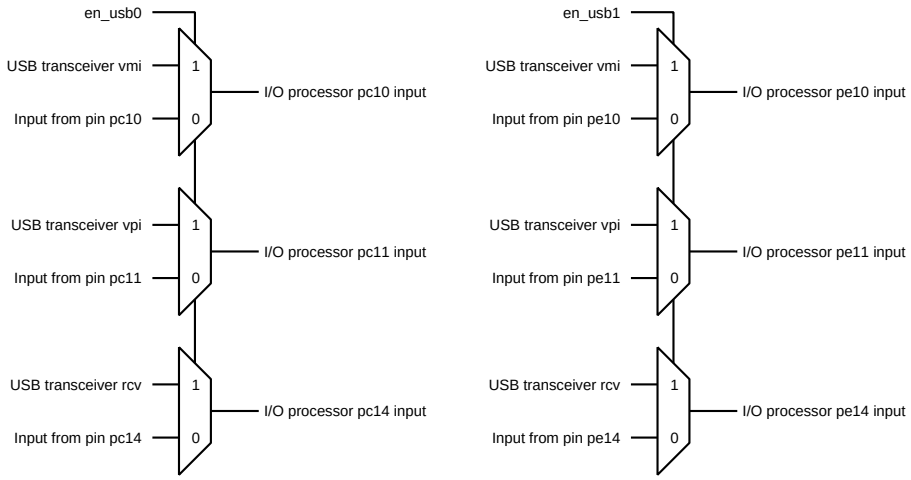


Figure 16.4: *Multiplexing of internal USB transceiver inputs*

Chapter 17

Stubless Debugging

17.1 Introduction

Stubless debugging is a method which makes it possible to debug software without using special debug software in RAM. An external debugger interfaces with the internal debug functionality in ETRAX FS using the TAP controller described in 15. It is possible to read and write all CPU registers, mode registers, internal and external memory. Using these commands it is also possible to set break points etc. All functionality to implement GDB from the external debugger is available.

The internal debug functionality is partly implemented by the CPU and partly by a program in an internal ROM. The functionality is available after the chip enters guru mode, described in 2.1.10.6. When this mode is entered the ETRAX FS does some initialization, described in the aforementioned chapter, and control is given to the program in ROM which awaits further commands in the mode registers, written to by the external debugger using the TAP controller. The mode registers are rw_tap_data and rw_tap_hdata, further described in 15.

17.2 Entering guru mode

By setting rw_tap_hdata.op.dbg guru mode is entered. Guru mode can also be entered by internal events as described in 2.1.10.6.

op	sub_op	rw_tap_data
dbg	unused	unused

Regardless of how guru mode is entered in ETRAX FS it is signalled to the external debugger by rw_tap_hdata.op.dbgdo and rw_tap_hdata.sub_op.gmode in the mode registers. After that the internal debugger is waiting for further commands.

op	sub_op	rw_tap_data
dbgdo	gmode	unused

17.3 Debug functions

All further communication between internal and external debugger is with rw_tap_hdata.op codes dbgdi and dbgdo. The former should always be set by the external debugger when writing to the mode registers and the latter is always set when the internal debugger replies.

The function to be performed is communicated to the internal debugger in rw_tap_hdata.sub_op. The functions are described in table 17.3.

sub_op	Function
gmode	Guru mode entered
rdreg	Read general register
rdpreg	Read special register
rdsreg	Read support register
wrreg	Write general register
wrpreg	Write special register
wrsreg	Write support register
rdmem	Read dword from memory
wrmem	Write dword to memory
rdmemb	Read byte from memory
wrmemb	Write byte to memory
ret	Return from guru mode

Table 17.3: *Debug functions and sub-op codes.*

17.3.1 Read/write register

General registers, special registers and support registers can all be written to individually. The bank for support registers is chosen by setting the SRS special register before calling read/write on a specific support register.

Among the things done when guru mode is entered is that R0-R3 and CCS are saved in support registers. After this they are used by the internal debugger and may not be changed while in guru mode. Instead the copies, which all will be restored when leaving guru mode, should be used. This, and the other register actions done by the CPU when guru mode was entered, has to be taken into consideration by the external debugger. The details are specified in 2.1.10.6.

17.3.1.1 Read register

The register set to operate on is selected with rw_tap_hdata.sub_op and the register number should be in rw_tap_data. Reading of reserved general registers R0-R3 is undefined.

op	sub_op	rw_tap_data
dbgdi	cmd	regnr

The contents of the register is returned in rw_tap_data and sub_op command repeated.

op	sub_op	rw_tap_data
dbgdo	cmd	value

17.3.1.2 Write register

The register set is selected in rw_tap_hdata.sub_op.

op	sub_op	rw_tap_data
dbgdi	cmd	unused

This is acknowledged with rw_tap_hdata.op.dbgdo set and the command is repeated in rw_tap_hdata.sub_op.

op	sub_op	rw_tap_data
dbgdo	cmd	unused

Next the data for the write should be provided with the register number in rw_tap_hdata.sub_op and the new register value in rw_tap_data.

op	sub_op	rw_tap_data
dbgdi	regnr	value

This is acknowledged with the register number in rw_tap_hdata.sub_op and the updated value in rw_tap_data.

op	sub_op	rw_tap_data
dbgdo	regnr	value

17.3.2 Read/write memory

Memory accesses can be done with 8 or 32 bit data.

17.3.2.1 Read

A memory read is performed with rw_tap_hdata.sub_op command rdmem or rdmemb and address in rw_tap_data.

op	sub_op	rw_tap_data
dbgdi	cmd	address

Data from this location is returned in rw_tap_data with rw_tap_hdata.sub_op unchanged.

op	sub_op	rw_tap_data
dbgdo	cmd	value

17.3.2.2 Write

A memory write is performed by setting rw_tap_hdata.sub_op command wrmem or wrmemb and address in rw_tap_data.

op	sub_op	rw_tap_data
dbgdi	cmd	address

This is acknowledged with rw_tap_hdata.op.dbgdo set and rw_tap_hdata.sub_op repeated.

op	sub_op	rw_tap_data
dbgdo	cmd	unused

Next the data for the write should be provided with the command repeated in rw_tap_hdata.sub_op and the new value in rw_tap_data.

op	sub_op	rw_tap_data
dbgdi	cmd	value

This is acknowledged with the command repeated in rw_tap_hdata.sub_op and the updated value in rw_tap_data.

op	sub_op	rw_tap_data
dbgdo	cmd	value

17.3.3 Return from guru mode

Guru mode will be exited using the RETN instruction. If the external debugger wishes to jump to another address than that which was interrupted by the guru mode, this can be done by setting the NRP special register before continuing. Note that it is the NRP register that should be modified directly and not the copy made by the CPU in the support function registers when guru mode was entered. See 2.1.10.6.

op	sub_op	rw_tap_data
dbgdi	ret	unused

Chapter 18

Timers

18.1 References

Reference	Description
[REGS]	Mode registers, chapter 25.44
[MACROS]	http://developer.axis.com
[PINMAP]	Pinout, chapter 16

18.2 Overview

The timer block contains the following functions:

- Two programmable 32-bit timers that can be clocked by an internal or external clock. The timers can be set up to generate interrupts at selected intervals.
- One 8-bit counter that can count either external events or terminal count events from the programmable timers.
- One 32-bit read-only timer, running at 100 MHz.
- One trig point, which can be set up to generate an interrupt when one of the programmable timers or the read-only timer reaches a selected value.
- One watchdog timer which can be set up to generate a non-maskable interrupt and a system reset if not stopped or re-activated within a selected amount of time.

18.3 Functional Description

18.3.1 Programmable Timers

The timer block contains two programmable 32-bit timers, tmr0 and tmr1.

18.3.1.1 Timer Operation

Each timer counts downwards from a programmable start value set in the rw_tmr0_div or rw_tmr1_div register. When the timer counts down from 1, it reloads the start value and generates an interrupt. The programmed start value is thus equal to the loop time of the timer (in number of timer clock cycles, see 18.3.1.2). A start value of 0 will give a loop time of 2^{32} clock cycles.

Each timer has three operation states, which are set in the rw_tmr0_ctrl or rw_tmr1_ctrl register:

Mode	Description
ld	The start value is continuously loaded into the timer.
hold	The timer is stopped but keeps its current value.
run	The timer is running.

Table 18.2: *Timer States*

The current values of the timers are readable from the r_tmr0_data and r_tmr1_data registers.

18.3.1.2 Timer Clock Frequency

The input frequency can be selected individually for each timer. The following alternatives are available, and can be configured in the rw_tmr0_ctrl or rw_tmr1_ctrl register:

- Off
- External serial/timer clock input, see 16
- 29.493088 MHz
- 32.000 MHz
- 32.768362 MHz
- 100.000 MHz

(The frequencies listed assume an exact 12 MHz input clock to the chip.)

18.3.1.3 Timer Output

One pin can be configured as a timer output, see 16. This pin can be configured in rw_out to use either of the two timers. The timer output will toggle each time the selected timer wraps.

18.3.1.4 Reset Behavior

The timers are in the load (ld) state after reset, with the input clock turned off. The start value is undefined after reset. The timer output is off (i.e. constant 0) after reset.

18.3.2 Counter

18.3.2.1 Counter Operation

There is one 8-bit counter that can be configured to count either external events (pulses on the serial/timer clock input), or number of loops in either of the two programmable timers.

The counter starts from 0, and increases by 1 for each event. The counter saturates at 255. The counter is readable from register rs_cnt_data with the side effect of resetting the counter to 0, or from register r_cnt_data without the side effect.

The counter keeps the cnt interrupt set whenever the counter value is non-zero. To clear the interrupt, the counter must first be cleared by reading the rs_cnt_data register.

18.3.2.2 Reading the Counter

Reading the counter value yields a 32-bit value where the counter value is the upper 8 bits. When a timer is selected as the source for the counter events, the lower 24 bits of that timer are read from the lower 24 bits of rs_cnt_data / r_cnt_data. This makes it possible to read a timer, read the counter, and reset the counter all in one indivisible operation. (This assumes that only the 24 lower bits of the timer are used.)

When counting external events, the lower 24 bits of rs_cnt_data / r_cnt_data come from the 24 lower bits of the continuous read-only timer.

18.3.2.3 Reset Behavior

The counter value is 0 after reset, and the counter is turned off.

18.3.3 Continuous Read-Only Timer

There is a continuous 32-bit read-only timer that starts from 0 at reset, and counts upwards. The input frequency of the read-only timer is 100 MHz (given a 12 MHz clock input to the chip). The value of the read-only timer can be read from the r_time register.

18.3.3.1 Test Mode

The read-only timer has a test mode to allow it to be tested quickly. While test mode is enabled, the timer is set to 0xffffff00. When test mode is released, the timer continues from this value. Since the watchdog uses bit 16 of the read-only timer as its input clock, going in and out of test mode can be used to clock the watchdog for test purposes.

The test mode becomes permanently disabled when the read-only timer reaches 65536 (0x00010000). It can also be permanently disabled via the dis field in the rw_test register. Once permanently disabled, the test mode can not be re-enabled until after a system reset.

18.3.4 Timer Trig Point

The timer block contains one timer trig point. The trig point can be used when an interrupt is wanted at a certain point in time, relative to the read-only timer or one of the programmable timers.

18.3.4.1 Trig Point Operation

The timer trig point can be set in the rw_trig register. The value of the rw_trig register is compared with one of the programmable timers or with the read-only timer, as selected in rw_trig_cfg.

The trig interrupt is generated when the trig point matches the value of the selected timer and the timer is clocked to leave that value. The interrupt is only generated if the selected timer is running.

18.3.4.2 Reset Behavior

The trig point is disabled after a system reset, and the trig point value is undefined.

18.3.5 Watchdog Timer

18.3.5.1 Watchdog Operation

The watchdog timer is an 8-bit timer with a configurable start value. Once started the watchdog counts downwards with a frequency of 763 Hz (100/131072 MHz). When the watchdog counts down to 1, it generates an NMI (Non Maskable Interrupt), and when it counts down to 0, it resets the chip.

There are two possible commands to the watchdog:

Command	Description
stop	Stops the watchdog.
start	Starts the watchdog with the selected start value. If the watchdog is already running, it is re-started from the start value.

Table 18.3: *Watchdog Commands*

When starting the watchdog, a 7 bit key value is supplied. Further commands while the watchdog is running will only take effect if the new key value matches the bitwise inversion of the key value of the previous valid command. This mechanism will protect the watchdog from being re-started or stopped by accident.

18.3.5.2 Configuring and reading the Watchdog

The watchdog is configured in the rw_wd_ctrl register. This register will always give 0 when read. The effective watchdog command and the current value of the watchdog timer can be read from the r_wd_stat register. The key value can not be read.

18.3.5.3 Reset Behavior

The watchdog is turned off after reset.

18.3.6 Interrupts

The timer block can generate the following interrupts:

tmr0 The tmr0 interrupt is generated when the tmr0 timer counts down from 1.

tmr1 The tmr1 interrupt is generated when tmr1 timer counts down from 1.

cnt The cnt interrupt is generated whenever the counter value is non-zero.

trig The trig interrupt is generated when the trigpoint matches the selected timer and the timer counts down from that value.

The interrupts are cleared by writing to the rw_ack_intr register. For the cnt interrupt, the counter must first be cleared or the clear of the interrupt will not take effect.

All interrupts can be masked through the rw_intr_mask register. The masked and non-masked interrupts can be read from the r_masked_intr and r_intr registers.

18.4 Hardware Interface

18.4.1 Timer Input Clock

There is one external input clock signal to the timer block. This clock input is shared with the serial ports, see 16.

The input clock is sampled and synchronized with the internal 100 MHz system clock. This results in a maximum input clock frequency of slightly below 100 MHz. Operation is guaranteed up to 83 MHz. The synchronization detects the positive edge of the input clock.

The timing for the timer clock input is given below.

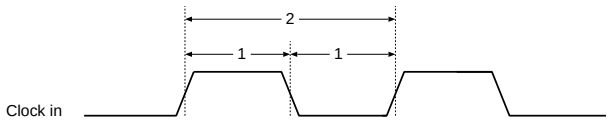


Figure 18.1: *Timer input*

Parameter Number	Explanation	Min	Nom	Max	Unit
t1	External clock pulse width	4	-	-	ns
t2	External clock cycle	12	-	-	ns

Table 18.4: *Timer Input Clock*

18.4.2 Timer Output

There is one pin that can be configured as timer output, see 16. The timing for the timer output is given below.

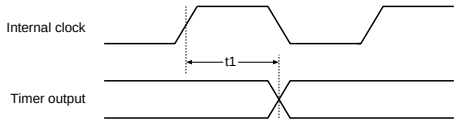


Figure 18.2: *Timer output*

Parameter Number	Explanation	Min	Nom	Max	Unit
t1	Timer output delay from internal clock	2	-	8	ns

Table 18.5: *Timer Ouput*

18.5 Software Interface

The different functions within the timer block are controlled by a set of registers, see 25.44 for a detailed description. To access the registers, fields and register constants from a C program, a set of macros is defined in [MACROS].

A few things that may need special programming considerations are described below:

18.5.1 Timer and counter

A typical use for programmable timers is to generate a periodic timer interrupt. The interrupt is generated each time the timer wraps. With short timer interrupt period in a system with long interrupt latency, the timer may wrap a second or more times before the interrupt is serviced.

To still be able to count the number of timer loops, the counter can be connected to the timer output. The rs_cnt_data register allows reading the counter, resetting the counter and reading the lower 24 bits of the selected timer, all in one indivisible operation. This avoids partly updated values when reading the timer and counter while the timer wraps.

When the counter is used in this way, either the timer interrupt (tmr0 or tmr1) or the counter interrupt (cnt) may be used as the periodic interrupt. The unused interrupt should be masked off. The interrupt should be acknowledged (through rw_ack_intr) after the counter has been read and cleared. Using the counter interrupt may give shorter response times than using the timer interrupt. If the timer wraps after the clear of the counter but before the acknowledge of the interrupt, the counter interrupt will occur again immediately, while the timer interrupt will not occur until the timer wraps a second time.

18.5.2 Trig point

The trig point can be used when an interrupt is wanted at a certain point in time, relative to the read-only timer or one of the programmable timers. If the time until the desired trig point interrupt is short, you may not be able to guarantee that the software will set the trig point in time to actually get the interrupt in the expected cycle of the selected timer. To handle this, the following procedure could be used:

1. Mask off the trig interrupt in rw_intr_mask.
2. Set the trig point value in rw_trig.
3. Enable the trig point and select timer in rw_trig_cfg.
4. Read the timer value to check if we have already passed the trig point.
5. If the trig point was not yet reached, you enable the trig interrupt in rw_intr_mask. The trig interrupt will now occur when expected.
6. If the trig point was already reached, you disable the trigpoint in reg:rw_trig_cfg, acknowledge the trig interrupt in rw_ack_intr and call the trig point interrupt routine directly.

Chapter 19

Asynchronous serial port

19.1 References

Reference	Description
[REGS]	Mode registers, chapter 25.40
[MACROS]	Register macros, http://developer.axis.com
[STRMUX]	DMA connections, chapter 10
[PINMAP]	Pinout, chapter 16
[DMA]	DMA, chapter 5

Table 19.1: *References*

19.2 Overview

The asynchronous serial module contains one asynchronous serial port with full buffering and parity control. The serial port has one handshake signal in each direction. The port can be polled, interrupt driven or DMA controlled.

The receiver and transmitter can be operated independently. They support baud rates from 56.25 baud to 12.5 Mbaud.

The ETRAX FS contains four instances of the asynchronous serial port. One port is always available, and three ports are multiplexed with other functions, see 16.

19.3 Functional Description

19.3.1 Asynchronous serial port registers

The mode registers for the asynchronous serial port are described in 25.40. There are 15 registers for each serial port:

Registers	Description
rw_tr_ctrl rw_rec_ctrl	Control registers for transmitter and receiver.
rw_tr_baud_div rw_rec_baud_div	Baud rate divider values.
rw_dout	Transmit data.
rs_stat_din r_stat_din	Status and receive data registers. Reading the rs_stat_din register will clear the receiver status fields, while r_stat_din is read without any side effect.
rw_tr_dma_en	Transmitter DMA enable.
rw_xoff rw_xoff_clr	Controls automatic xoff detection.
rw_rec_eop	Generates end of packet for the receive DMA.
rw_intr_mask rw_ack_intr r_intr r_masked_intr	Interrupt control.

Table 19.2: *Asynchronous serial port registers*

19.3.2 Baud rate selection

The baud rate is set by a combination of a base frequency and a divide factor. The baud rate will be:

```
base frequency / ( 8 * divide factor)
```

The following base frequencies are available:

- 100 MHz
- 32.768 MHz
- 32 MHz
- 29.493 MHz
- External clock (maximum 83 MHz)

The divide factor can be set in the range 1-65536. This gives a baud rate range from 56.25 baud to 12.5 Mbaud.

The baud rate can be set individually for the receiver and transmitter. For the transmitter, the base frequency is set in the base_freq field of rw_tr_ctrl, and the divide factor is set in rw_tr_baud_div. For the receiver, the base frequency is set in the base_freq field of rw_rec_ctrl, and the divide factor is set in rw_rec_baud_div. For a divide factor of 65536, the corresponding register should be set to 0.

Standard baud rates in the range 75 baud to 3.6864 Mbaud can be achieved with the 29.493 MHz base frequency:

Baud rate	Divide factor
75	49152
150	24576
300	12288
600	6144
1200	3072
2400	1536
4800	768
9600	384
19200	192
38400	96
57600	64
76800	48
115200	32
230400	16
460800	8
921600	4
1843200	2
3686400	1

Table 19.3: *Baud rates and divide factors*

The baud rates derived from this table will be 64 ppm faster than the nominal baud rate value.

19.3.3 Serial protocol operation modes

The operation modes for the transmitter are configured in rw_tr_ctrl, and for the receiver they are configured in rw_rec_ctrl.

19.3.3.1 Character format

The receiver and transmitter can be individually configured for 7 or 8 data bits and odd, even, mark, space or no parity.

The transmitter can be configured for one or two stop bits.

The receiver can be configured to take one sample in the middle of each bit or to take the majority of three samples.

19.3.3.2 Handshake signals

The transmitter can be configured to handle **cts_n** automatically, or to ignore **cts_n**. When automatic **cts_n** handling is selected, a high on the **cts_n** input will halt the transmitter after the ongoing byte, and keep it halted until **cts_n** becomes low again.

The **rts_n** output can be controlled via the rw_rec_ctrl register. The **rts_n** output can also be used as a direction control to the external transceiver for RS 485 operation, see section 19.3.3.4 below.

19.3.3.3 Automatic xoff handling

The serial port can be configured to detect when an xoff character is received, and to automatically stop transmission after the ongoing byte. The enabling of the automatic xoff mode and the character code for the xoff character are configured in the rw_xoff register.

A detected xoff character will set the xoff_detect field in the rs_stat_din and r_stat_din registers. After the ongoing byte is completely transmitted, the tr_idle and tr_empty status fields and interrupts will also be set. The xoff detected condition is cleared by writing to the rw_xoff_clr register. The xoff detected condition will also be cleared when the automatic xoff handling is turned off, or when the serial receiver is disabled.

19.3.3.4 RS 485 operation

For RS 485 operation, the **rts_n** signal is used as a direction control for an external transceiver. The value of the rts_n field in the rw_rec_ctrl register should be set to the value corresponding to the ”transmitter off” state of the external transceiver.

RS 485 direction control is enabled by the auto_rts field of rw_tr_ctrl. When enabled, the **rts_n** output will toggle in the beginning and end of a transmission. The set up time for **rts_n** before the transmission and the hold time after the transmission can be configured in the rts_setup and rts_delay fields of rw_tr_ctrl. If transmission of a new byte is initiated before the time set by rts_delay has expired, no **rts_n** toggling will occur between the two bytes.

To avoid reception of the transmitted data, the receiver can be configured for half duplex operation. In this mode, the receiver is disabled while there is an ongoing transmission. The half duplex mode is set in the half_duplex field of rw_rec_ctrl.

19.3.3.5 Stop transmitter

The transmitter can be stopped in a controlled way by using the stop field of the rw_tr_ctrl register.

When the stop field is set, the transmission of the ongoing byte (if any) will be completed including the specified number of stop bits. Thereafter the tr_empty and tr_idle fields of the rs_stat_din and r_stat_din registers will be set, and the **txd** output will be set to the value of the txd field of rw_tr_ctrl.

19.3.3.6 Internal loop back

The serial port has an internal loop back mode which is controlled by the loopback field of rw_rec_ctrl. When set, the **txd** signal is internally looped back to the **rxd** signal, and the external **rxd** input is ignored. The transmitted data is still output on the external **txd** signal. The **rts_n** and **cts_n** signals are not affected by the loop back mode.

19.3.4 CPU controlled operation

19.3.4.1 Transmitter

Data to be transmitted is written to the rw_dout register. This starts the transmission and clears the tr_rdy field in rs_stat_din and r_stat_din. If interrupts are used, the tr_rdy interrupt should be cleared by writing to the rw_ack_intr register after the transmission is started.

When the transmitter is ready to accept new data from the CPU, the tr_rdy field is set in rs_stat_din and r_stat_din. This also sets the tr_rdy interrupt.

CPU controlled transmission can take place independent of whether DMA is enabled or not. If data is available from both sources, the CPU transmission will have priority.

The two status fields tr_empty and tr_idle in rs_stat_din and r_stat_din can be used to check when a transmission is completed.

The condition for tr_idle is:

```
(no ongoing transmission) and
((DMA disabled) or
((automatic xoff) and (xoff detected)) or
(transmitter stopped) or (transmitter disabled))
```

The condition for tr_empty is:

```
(no ongoing transmission) and
((DMA disabled) or (DMA FIFO empty) or
((automatic xoff) and (xoff detected)) or
(transmitter stopped) or (transmitter disabled))
```

19.3.4.2 Receiver

When data is available from the receiver, the dav in rs_stat_din and r_stat_din is set. This also sets the dav interrupt.

Received data is available in the data field of the rs_stat_din and r_stat_din registers. Reading the rs_stat_din register will clear the dav field, while reading the r_stat_din will leave the status unchanged.

When interrupts are used, the dav interrupt should be cleared by writing to rw_ack_intr after the dav status has been cleared.

If the serial receiver encounters a parity error, framing error or overrun error, the errors are indicated in the rs_stat_din and r_stat_din registers. The error status is valid whenever the dav field is set, and is cleared when the rs_stat_din register is read.

Data can be read by the CPU even if the receiver is in DMA mode. This will not affect the DMA data stream.

19.3.5 DMA controlled operation

19.3.5.1 DMA channel connections

The serial port can be connected to internal DMA. The transmitter and receiver have separate DMA channels and can be configured independently. The mapping between serial ports and DMA channels is described in 10.

19.3.5.2 Transmitter

DMA transmission is enabled in the rw_tr_dma_en register. Before a DMA transmission can take place, the internal DMA channel must be connected to the serial port and started. How to set up the internal DMA is described in 5.

The transmission continues as long as there is data available, or until an out_eop is present in a DMA descriptor, see 5. When out_eop is set, the en field of rw_tr_dma_en is cleared after the data buffer associated with the DMA descriptor has been fully processed. An out_eop will thus stop further DMA transfers. To continue, the en field of the rw_tr_dma_en must be set again.

The completion of a DMA transmission can be detected by the tr_empty or tr_idle fields in rs_stat_din and r_stat_din. The criteria for the two status bits are defined in section 19.3.4.1 above. Note that the tr_empty status can be used as an indication of a completed DMA transfer only if the associated DMA channel has reached end of list. The use of tr_idle for indication of DMA completion requires that the out_eop is set in the last descriptor of the DMA transfer.

19.3.5.3 Receiver

Receiver DMA is enabled in the dma_mode field of rw_rec_ctrl. Before reception through DMA can take place, the internal DMA channel must be connected and started. How to set up the internal DMA is described in 5.

In a DMA controlled operation, received data is sent to the connected internal DMA channel. The DMA may buffer up to 32 bytes before any data is written to memory. Therefore, after the last byte is received there may be up to 32 bytes left in the DMA. To write out the remaining data to memory, an in_eop must be issued to the DMA channel. This can be done in two ways:

1. Software generated in_eop
2. Time out based in_eop

The software can generate an in_eop by writing to the rw_rec_eop register.

Time out based in_eop generation is enabled by setting the auto_eop field in rw_rec_ctrl. An in_eop will be generated if no new data is received after the timeout configured in the timeout field of rw_rec_ctrl. The timeout can be configured to 0-7 character times. A value of 0 will result in an in_eop as soon as characters are not received back to back.

At least one character must be received in DMA mode before a time out based in_eop is generated.

If auto_eop is set during DMA mode reception, and the DMA mode is disabled while keeping the auto_eop set, an in_eop is generated immediately provided that at least one byte was received since the DMA was started or the previous in_eop was given. If auto_eop is disabled at the same time as the DMA mode, no in_eop is generated.

To handle receive errors in a DMA controlled operation, there are two different modes, controlled by the dma_err field in rw_rec_ctrl:

1. Stop on error
2. Ignore errors

Stop on error In this mode, a receive error generates an in_eop to the DMA, and halts the receiver. The byte causing the error, and any bytes arriving thereafter, will be discarded until the error condition is cleared.

The error status and the data in rs_stat_din and r_stat_din will be updated according to the last discarded byte. For example, if a parity error or framing error occurs, and a correct byte arrives thereafter, both bytes will be discarded, the parity error and framing error status will be cleared and the overrun error status and rec_err status will be set. The last discarded byte will be present in the data field of the rs_stat_din and r_stat_din registers.

The error condition is cleared by reading the rs_stat_din register. The occurrence of an error can be detected through the in_eop interrupt from the DMA channel.

Ignore errors In this mode, bytes containing parity errors or framing errors will be forwarded to the DMA. Overrun errors will cause lost bytes, but no errors will halt the DMA. The error status will be updated according to the last arriving byte. The rec_err field in rs_stat_din and r_stat_din can be used to check whether any errors occurred during the data transfer.

The dav field will not be affected by whether DMA is enabled for the receiver or not. Thus, dav will be set as soon as a byte is received, and will remain set until the rs_stat_din register is read by software. When disabling the DMA mode, the dav field must be cleared in order to avoid overrun errors if polled or interrupt driven operation will follow.

19.3.6 Interrupts

The serial port has the following interrupts:

tr_rdy The tr_rdy interrupt is generated whenever the serial transmitter is ready to receive a new data byte for transmission.

tr_empty The tr_empty interrupt is generated whenever the tr_empty status is set. The criteria for generating tr_empty are defined in section 19.3.4.1 above.

tr_idle The tr_idle interrupt is generated whenever the tr_idle status is set. The criteria for generating tr_idle are defined in section 19.3.4.1 above.

dav The dav interrupt is set whenever new data is available from the serial receiver.

Note that all interrupts are generated whenever the corresponding status bits are set, but the interrupts remain active until they are cleared by writing to the rw_ack_intr register. If the corresponding status bit is still set when the interrupt is cleared, the interrupt will be generated again immediately.

All interrupts can be masked through the rw_intr_mask register. The masked and non-masked interrupts can be read from the r_masked_intr and r_intr registers.

19.4 Hardware Interface

19.4.1 Input and output signals

Each serial port has five I/O signals:

Signal	Direction	Description
cts_n	in	Clear to send
ext_clk	in	Clock input for external baud rate clock.
rxd	in	Receive data
rts_n	out	Request to send / transceiver direction
txd	out	Transmit data

Table 19.4: *Serial port signals*

The ETRAX FS contains four asynchronous serial ports. Port 0 is always available, and the three other ports are shared with other functions. The mapping of serial port signals to I/O pins is described in 16. The **ext_clk** input signal is shared between all four serial ports and also shared with other functions.

I/O pins have LVTTL signal levels, and external buffers are needed to adapt to the RS232 or RS485 standard. The **s0txd** and **s0rts_n** outputs of serial port 0 will go high immediately at reset. If the application requires immediately defined output values on the other serial ports, it can be achieved with pull up resistors.

19.4.2 Signal timing

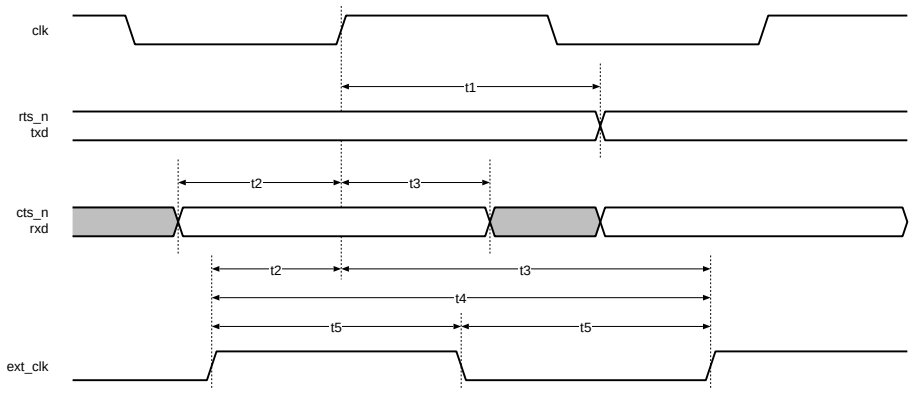


Figure 19.1: *Signal timing*

Nbr	Name	Description	min	max	unit	note
t1	tod	Output delay from internal clock	2	9	ns	-
t2	tsu	Input setup to internal clock	4	-	ns	1
t3	th	Input hold time from internal clock	0	-	ns	1
t4	tecp	Clock period of ext_clk	12	-	ns	-
t5	tecw	Pulse width of ext_clk	4	-	ns	-

Table 19.5: *Asynchronous serial timing*

Note 1: The **cts_n**, **rxd** and **ext_clk** are internally synchronized. Setup and hold times can be ignored unless recognition in a specific clock cycle is required.

19.5 Software Interface

19.5.1 General

The serial port is controlled through a set of registers, see 25.40 for details. To access the registers, fields and register constants from a C program, a set of macros is defined in [MACROS].

The I/O signals of serial ports 1 to 3 are multiplexed with other functions. Before the output signals of serial port 1 to 3 can be used, the pins must be configured, see 16. Serial port 0 is always available.

19.5.2 DMA operation

The serial port can be polled, interrupt driven or DMA controlled. If DMA is used, the corresponding DMA channels must first be connected to the serial port, see 10, and the

DMA must be set up and started, see 5. The serial port DMA operation uses simple data descriptor lists. Data descriptors for serial port operation should not have the wait field set.

For output descriptors, the out_eop field of the descriptor can be used to stop the DMA operation of the transmitter after all data associated with the descriptor has been transmitted. The tr_idle status and/or interrupt can be used to indicate that the transmitter has stopped. The DMA transmissions can be resumed by writing to rw_tr_dma_en.

If the output DMA reaches end of list the transmitter will halt when all data has been transmitted. This condition can be detected by the tr_empty status and/or interrupt in conjunction with the end of list status from the DMA. Note that it is not sufficient to look only at tr_empty since this status may occur also in case of a temporary DMA underrun situation.

The input DMA channel may buffer up to 32 bytes before any data is written to the memory. To write out the remaining data, an in_eop has to be issued to the DMA. How to do this is described in 19.3.5.3.

Chapter 20

ATA Interface

20.1 References

reference	Description
[DMA_MDS]	Documentation for the internal DMA channels, chapter 5.
[ATAREGS]	ATA interface mode registers, chapter 25.3.
[ATASTD]	Information technology - AT Attachment with Packet Interface - 6 (ATA/ATAPI-6)
[PINMAP]	The ETRAX FS external pin database, chapter 16.
[STRMUX]	Documentation for the DMA channel connections, chapter 10.
[MACROS]	Register macros for the ATA interface, http://developer.axis.com

Table 20.1: *References*

20.2 Definitions

ATA	Advanced Technology Attachment interface, described in this document.
PIO	The Programmed Input/Output ATA transfer mode
DMA	Direct Memory Access, referring to the ATA transfer mode unless it is explicitly stated that DMA refers to an internal DMA channel.
UDMA	The Ultra Direct Memory Access ATA transfer mode

Table 20.2: *Definitions*

20.3 Overview

The Advanced Technology Attachment (ATA) interface implements PIO mode, DMA mode and UDMA mode transfers between the ETRAX FS and an externally connected ATA compliant device, for example a hard disk, CD-ROM or CD writer.

In ATA, most of the state logic and intelligence resides in the host's driver software and in the connected device's firmware, so the interface is not very complex.

Basically, the PIO mode transfers allow the host to read and write internal device registers. This is useful when configuring future data transfers which are done over the bus using one of the three transfer modes mentioned above.

It is up to the driver software to decide what commands to use to tell the device to send or receive the desired data, and to know what to do with received data or when to send data. The hardware interface just does the transfer.

The interface communicates its state and initiates transfers based on software accesses to its registers. Preferably, data should be transferred to or from memory via internal DMA channels to achieve maximum performance.

Data can also be transferred via registers in the PIO and DMA ATA transfer modes. For example, data can be transferred using these modes to simplify driver software where performance is not an issue, or if the internal DMA channels are unavailable.

Configuration registers for selecting the ATA bus handshaking timings to use exist as well, so different submodes of the supported transfer modes can be selected.

The attached devices can signal an interrupt to the host. This interrupt is transparent to the ATA interface itself - it is only used for communication between the device and the host software.

While there is only a single ATA controller in the ETRAX FS, support for up to four external ATA buses exists, by duplicating the necessary control wires and multiplexing them internally. As a result, parallel transfers among the buses are not possible.

20.4 Functional description

The ATA interface is controlled through registers, see the ATA interface register description 25.3 for details on registers mentioned below.

20.4.1 Transfer parameters

Each transfer has a number of parameters:

- Bus selection (0-3)
- Bus address A0/A1/A2/CS0/CS1
- Transfer length
- Transfer handshaking mode (PIO/DMA/UDMA)
- Transfer speed/timing (referred to as submode timing below, and sometimes referred to as PIO/DMA mode 0, 1, 2 etc)
- Memory transfer mode (register based or internal DMA based)

Except for the transfer submode timings and transfer length, the other parameters are set at the same time as the transfer is started by a single write to the rw_ctrl2 register. The submode timings as well as some other configuration bits need to be written to the rw_ctrl0 and rw_ctrl1 registers before any transfer is started. If the transfer is a multiword transfer (most are, except for device register accesses) then the rw_trf_cnt register also needs to be loaded with the number of words to transfer before any transfer is initiated.

For all transfers, the software must wait for the busy bit in rs_stat_data to clear before initiating the transfer.

20.4.2 Host transfer method

If memory transfer is done through an internal DMA channel, the associated channel should be initialized and started before any transfer is initiated. The ATA interface used in ETRAX FS uses internal DMA channels 2 and 3. In order for the interface to get access to them, the strmux must be configured. For details about how to configure the strmux, please refer to 10.

In a typical software driver, device commands are issued through register-based PIO transfers and bulk data is transferred using internal DMA in combination with ATA DMA or UltraDMA mode bus transfers at the highest speed the device supports.

There are three transfer lengths involved in an ATA data transfer:

1. the transfer counter in the interface rw_trf_cnt
2. the internal DMA descriptor lengths (if internal DMA is used)
3. the transfer length written by an ATA command to the device.

Normally, the transfer length written to the rw_trf_cnt register should match the total length of the enabled DMA descriptors and the amount of data requested to transfer to/from the ATA device. During device reads, the ATA interface will issue an EOP to the internal DMA channel when the transfer counter reaches zero. The transfer counter also controls the length of the transfer during writes. When the interface is configured for UltraDMA, the transfer counter value shall include the word containing CRC meaning that the value must be increased by one. The EOP bit in the out channel descriptors should not be set.

The software has to set up the DMA channels to have the same transfer width as the ATA transfer to be started. In most circumstances this should be 16 bits wide.

20.4.3 Address counter

The ATA address bus is 3 bits wide. Address 0 is used when doing bulk data transfer, but the other 7 addresses usually correspond to device registers that are to be loaded with data to initiate an ATA command. Usually the software writes these registers individually using PIO mode and a register controlled data transfer.

However, internal DMA can also be used to load the different registers in succession because when any of the registers except the data transfer register (0) is accessed, the address is increased by one. So in order to load ATA registers 1 to 7 with data, a DMA descriptor of length 7 can be set up and the transfer initialized to ATA address 1. This is usually only a very minor optimization compared to a register controlled load, since the device specifications often mandates that the actual ATA bus transfer mode used for the register access is a relatively slow PIO mode anyway.

20.4.4 Interrupts

Each ATA bus has an interrupt signal that can be raised by an attached device. The interrupts can be masked using the rw_intr_mask register.

The interrupts are level triggered, i.e., the internal interrupt will be set whenever the ATA bus interrupt signal is active, and will remain set until acknowledged in the rw_ack_intr register inside the ATA interface. If the ATA bus interrupt signal is still active when the corresponding internal interrupt is acknowledged, the internal interrupt will immediately be set again.

20.4.5 Handling timeouts and errors

High-level error conditions are reported to the host software by the ATA devices using the normal ATA register polling and interrupt methods and does not require any separate support in the ATA interface.

However sometimes the low-level handshaking can be disrupted, either because of faulty device implementations, faulty cabling/electronics or configuration errors. The interface's internal state might get stuck waiting for a protocol event that never happens or the internal DMA channels might be configured to read more data than the device is commanded to read.

The interface hardware does not detect such conditions since for most protocol operations, it is legal for the device to pause indefinitely. It is up to the driver software to set timeouts and reset the interface if it decides something is wrong.

20.5 Hardware interface

In this document we specify the ATA interface external signals. These signals will in turn be mapped onto the ETRAX FS's external pins. For details about the mapping of the signal names mentioned below onto the ETRAX FS's pins, please refer to 16.

The direction of the signals in the ATA interface is given below. Signal names ending with an underscore and n (_n) are active low. Explanations:

I/O Bidirectional signal that changes direction during operation.

In Input signal, signal from device to host.

Out Output signal, signal from host to device.

The ATA interface has the following external signals:

General signals	Direction	Description
reset_n	Out	Reset
ext_oe	Out	Data output enable
cs_n[0]	Out	Chip select bit 0
cs_n[1]	Out	Chip select bit 1

Table 20.3: *General signals*

Address/data signals	Direction	Description
a[0]	Out	Device address0
a[1]	Out	Device address1
a[2]	Out	Device address2
data[0]	I/O	Data bus bit0
data[1]	I/O	Data bus bit1
data[2]	I/O	Data bus bit2
data[3]	I/O	Data bus bit3
data[4]	I/O	Data bus bit4
data[5]	I/O	Data bus bit5
data[6]	I/O	Data bus bit6
data[7]	I/O	Data bus bit7
data[8]	I/O	Data bus bit8
data[9]	I/O	Data bus bit9
data[10]	I/O	Data bus bit10
data[11]	I/O	Data bus bit11
data[12]	I/O	Data bus bit12
data[13]	I/O	Data bus bit13
data[14]	I/O	Data bus bit14
data[15]	I/O	Data bus bit15

Table 20.4: *Address/data signals*

Control signals	Direction	Description
dior_n[0]	Out	I/O read device 0
dior_n[1]	Out	I/O read device 1
dior_n[2]	Out	I/O read device 2
dior_n[3]	Out	I/O read device 3
diow_n[0]	Out	I/O write device 0
diow_n[1]	Out	I/O write device 1
diow_n[2]	Out	I/O write device 2
diow_n[3]	Out	I/O write device 3
iordy[0]	Out	I/O ready device 0
iordy[1]	Out	I/O ready device 1
iordy[2]	Out	I/O ready device 2
iordy[3]	Out	I/O ready device 3
dmack_n[0]	Out	DMA acknowledge device 0

dmack_n[1]	Out	DMA acknowledge device 1
dmack_n[2]	Out	DMA acknowledge device 2
dmack_n[3]	Out	DMA acknowledge device 3
dmarq[0]	In	DMA request device 0
dmarq[1]	In	DMA request device 1
dmarq[2]	In	DMA request device 2
dmarq[3]	In	DMA request device 3
intrq[0]	In	Interrupt request, device 0
intrq[1]	In	Interrupt request, device 1
intrq[2]	In	Interrupt request, device 2
intrq[3]	In	Interrupt request, device 3

Table 20.5: *Control signals*

Most signals in the ATA interface are independent of operating mode. However there are three control signals that are multiplexed for different usage depending on operating mode and they are explained below. I/O read (dior_n) is used as DMA ready during Ultra DMA data in bursts and as data strobe during UltraDMA data out bursts. I/O write (diow_n) is used as stop during UltraDMA data bursts. Finally the iordy signal is used as DMA ready during UltraDMA data out bursts and as data strobe during UltraDMA data in bursts.

To be able to handle up to 4 devices in parallel, the ATA interface has 4 sets of control signals. The data and address lines are multiplexed so only one device may use them at any given time.

20.6 Software interface

In the following sections a short description of the ATA interface's software interface is given.

20.6.1 Configuration registers

There are a number of registers which may be configured by software. All registers are described in 25.3. To access the registers, fields and register constants from a C program, a set of macros is defined in [MACROS].

20.6.2 DMA descriptors

The ATA interface requires at least one data descriptor for each DMA channel. When creating a DMA descriptor list, the EOP bit should not be set in any descriptor while the EOL bit should be set in the last descriptor of the list. The width of the data channel used by the DMA must be configured by software before any transfers are initiated. In most cases the width will be 16 bits. The above is true for both the in and out channel. For further details on how to configure the DMA, please refer to 5.

20.6.3 Transfer modes

There are three available transfer modes, PIO, DMA and UDMA. Please refer to 25.3 for details on how to configure the interface for each mode.

20.6.4 Software reset

The interface state can be reset using the en field in the rw_ctrl0 register, thus allowing the interface to be reconfigured. The rst field in the same register can be used to reset a device connected to the ATA interface.

Chapter 21

Ethernet Interface

21.1 References

Reference	Description
[IEEE8023]	IEEE Std 802.3, 1998 Edition
[ETHREGS]	Mode registers, chapter 25.12
[ETH_H]	C macros, http://developer.axis.com
[DMA_MDS]	DMA, chapter 5
[PINMAP]	Pinout, chapter 16
[STRMUX]	DMA connection, chapter 10
[BOOT_MDS]	Boot methods, chapter 6

Table 21.1: *References*

21.2 Definitions

Term	Description
DMA out	DMA out means DMA out from memory into the Ethernet interface.
DMA in	DMA in is DMA from the Ethernet interface to the memory.
EOP	The DMA data descriptor end of packet control bit.
EOL	The DMA data descriptor end of list control bit.
PHY	Physical Layer in the OSI 7 layer model described in section 3.1
(Ethernet) Station	The portion of the Ethernet hardware that resides in any networked computer equipment, such as a personal computer (PC) file server, mainframe computer or printer
MAC	Media Access Control sublayer in the OSI 7 layer model. See section 3.1.
GAT	Group Address Table, used for deciding if if a group addressed frame should be received or not, see section 4.2.2.
MDIO	Management Data Input Output interface, a serial interface used for communicating with the PHY.
SOF	Start Of Frame delimiter byte, which is part of an Ethernet frame.

Table 21.2: *Definitions*

21.3 Overview

The ETRAX FS includes two on-chip instances of the Ethernet interface. The interface supports transfer rates of 10 Mbit/s and 100 Mbit/s, using the the Media Independent Interface (MII) as external interface toward the Physical layer. The Ethernet interface is connected to two DMA channels, one in each direction. When receiving and transmitting packets over a Local Area Network (LAN) the Ethernet MAC sublayer keeps track of the network status and checks that packets has been transmitted and received correctly. To understand where the Ethernet interface fits in the network hierarchy the OSI 7 layer model can be of help. In the table below the Ethernet interface implements part of the Data Link Layer, more precisely the MAC sublayer.

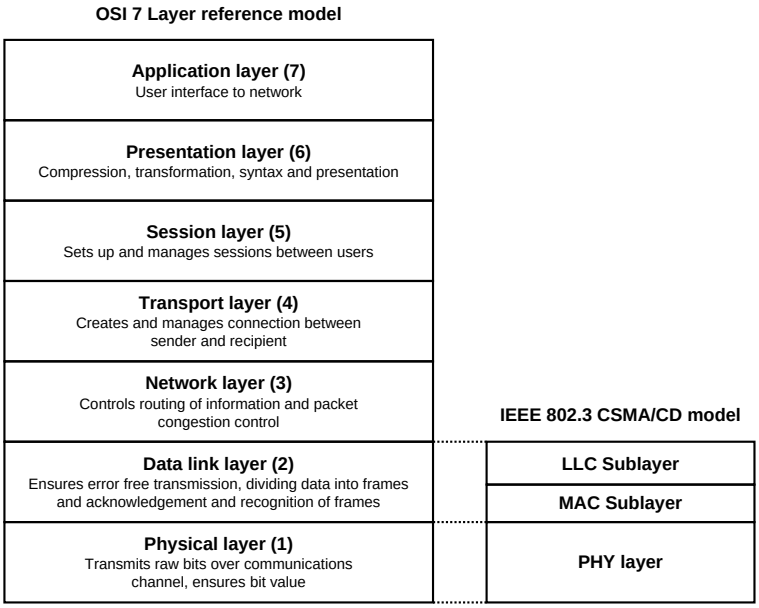


Figure 21.1: *OCI and IEEE 802.3 model comparison*

When sending data over an Ethernet network, data is separated into frames (packets). The frame format is shown in the figure below, all lengths are in bytes.

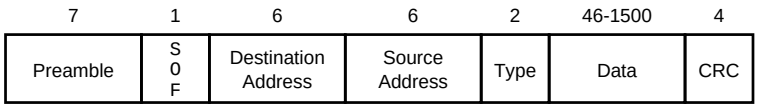


Figure 21.2: *Frame format*

Before using the interface there are two things that needs to be done. First you need to configure the interface's registers such as MAC address and half or full duplex etc. A full description of the Ethernet interface registers is given in 25.12. After configuring the interface the user needs to setup and activate one or multiple DMA descriptors

linked together as a list for each of the DMA channels. When finished the Ethernet interface is ready to receive and transmit data over a network.

A system overview diagram:

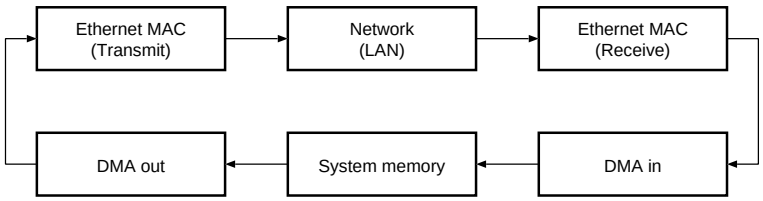


Figure 21.3: *System overview*

21.4 Functional description

21.4.1 Transmitter

When transmitting, the Ethernet interface starts by sending the preamble and the SOF. After that the interface continues by reading data from the memory via the DMA, starting with the destination and source addresses followed by the type field, continuing with the data payload. The interface continues to read data from the DMA out channel until EOP is reached. When EOP is reached, the transmitter will automatically append the frame's CRC. The crc field in the rw_tr_ctrl register can be used to disable the transmitter from appending the CRC. The table below shows which parts of the frame that are provided by the Ethernet interface and which parts that are read from memory.

Frame field	Source
Preamble	Generated by the Ethernet interface
SOF delimiter	Generated by the Ethernet interface
Destination address	Read from memory
Source address	Read from memory
Type	Read from memory
Data	Read from memory
CRC	Generated by the Ethernet interface

Table 21.3: *Transmit frame source*

Packets shorter than minimum length (64 bytes) can be automatically padded by the interface. The interface will in this case increase the length by appending zeros to the data payload. Automatic padding is enabled by configuring the pad field in the rw_tr_ctrl register.

21.4.1.1 Error handling

If the medium is free the transmitter may start a transmission. There are two events that will cause an error when transmitting, these are:

- Collisions
- Buffer underrun

When a collision is detected, a jam sequence will be sent. The jam sequence ensures that all stations on the network detect the collision. The sequence consists of 32 bits, all 1's. After that, the Ethernet interface will wait a random backoff time before trying to transmit the frame again. After a collision has been handled the appropriate error counter in the r_tr_cnt register will be incremented. If 15 retries are made without success the DMA list will be restored and the excessive collision interrupt will be generated. The transmitter will then wait until the clr field in the rw_clr_err register has been set. When the clr field is set, the transmitter will resume operation again, starting with the packet that generated the excessive collision.

Buffer underrun is handled in much the same way as excessive collisions. Buffer underrun occurs when the DMA out channel can not deliver data at the rate required by the Ethernet interface. In this case the interface will detect the underrun condition and abort the transmission by sending a 32 bit jam sequence. After that, the interface will stop and generate an underrun interrupt. By setting the clr bit in the rw_clr_err register, the transmitter will resume operation again starting with the packet that generated the underrun.

21.4.2 Receiver

When receiving, the Ethernet interface starts by sensing the preamble and SOF which are used by the receiver to lock to an incoming bit stream. After that it receives the destination and source addresses. The rest of the frame will be received if one of following three criteria is fulfilled:

1. The destination address matches an enabled station address.
2. The destination address is a broadcast address and the reception of broadcast frames has been enabled. Please refer to section 4.2.1.
3. The destination address is a group (multicast) address that matches the GAT or it is a unicast address that matches the GAT at same time as the individual field in the rw_rec_ctrl register is set.

If none of the above is fulfilled, the frame will be discarded.

Assuming that one of the criteria above is fulfilled, the interface then continues by receiving the type field, followed by the frame payload. In parallel with receiving the frame, the interface calculates the CRC for the frame. This value is compared with the CRC included in the frame. If these two matches, the frame is received without errors

and the CRC is written to memory. If not, the frame is discarded unless the bad_crc field in the rw_rec_ctrl register is configured to allow erroneous CRC.

Table 4.2 illustrates which frame segments that are written to memory and which that are removed by the receiver.

Frame field	Action
Preamble	removed by the receiver
SOF delimiter	removed by the receiver
Destination address	written to memory
Source address	written to memory
Type	written to memory
Data	written to memory
CRC	written to memory

Table 21.4: *Receiver frame format*

21.4.2.1 Address recognition

There are three types of addresses defined in the IEEE 802.3 standard [IEEE8023]. Below is a description of each type and how they are used:

- Individual address

  The individual address is a 48 bit unique address assigned to one station on the network. The Ethernet interface supports two different station addresses, MA0 and MA1. The addresses are defined by writing to the rw_ma0_lo/rw_ma0_hi and rw_ma1_lo/rw_ma1_hi registers. To enable recognition of each address the ma0 and ma1 fields in the rw_rec_ctrl register are used.

- Group address

  The group (multicast) address is used to index the 64-bit Group Address Table (GAT). The 6-bit hash address used to index the GAT is derived from the 48 bit destination address (DA) using the following formula:

  GAT index[5:0] = DA[5:0] ^ DA[11:6] ^ DA[17:12] ^ DA[23:18] ^ DA[29:24] ^ DA[35:30] ^ DA[41:36] ^ DA[47:42]

  Note: ^ means XOR

  There are two registers rw_ga_hi and rw_ga_lo which contains the GAT.

  The indexed entry in the GAT indicates whether the frame should be copied to memory or discarded.

- Broadcast address

  Reception of frames with broadcast address, FF:FF:FF:FF:FF:FF, can be enabled using the broadcast field in the rw_rec_ctrl register.

21.4.2.2 Received frame length check

The lengths of the incoming frames can be checked against the specific limits of the IEEE 802.3 [IEEE8023] frame format:

- Minimum: 64 bytes - Maximum: 1518 or 1522 bytes

The maximum length of a standard Ethernet packet is 1518 bytes. The length can be extended to 1522 bytes when VLAN tagging, according to IEEE 802.1q, is used. The maximum length is configured with the max_size field in the rw_rec_ctrl register.

If the frame length check is enabled and the frame length is outside of these boundaries, the frame is discarded. The frame length check is enabled using the oversize and undersize fields in the rw_rec_ctrl register.

21.4.2.3 Error handling

There is a number of error events that the receiver can detect:

- Buffer overrun
- CRC and alignment errors
- Oversized/undersized frames

These errors are all handled in the same way. The interface detects an error, discards the current frame, increments the appropriate error counter and then waits for a new frame.

21.4.3 Duplex and flow control

The choice of half or full duplex is made manually by configuring the duplex field in the rw_rec_ctrl register. However, when the ETRAX FS is configured for network boot mode, the choice of half or full duplex is made automatically by reading an external pin, please refer to 6 for further details. If half duplex is used the interface will not transmit and receive packets at the same time. If full duplex is used the interface may transmit and receive frames at the same time. Both the CRS and COL signals are ignored in full duplex mode. When configured for full duplex mode, the interface can be configured to react on 802.3x flow control frames through the flow_ctrl field in the rw_gen_ctrl register.

Note: If a flow control frame is received with an incorrect OP code, this frame will be received and written to memory. The software handling the memory buffers should discard these packets.

21.4.4 MDIO interface

The MDIO interface is used when reading and writing the MII management registers inside the PHY. Externally the interface consists of two signals, mdio (management data input output) and mdc (management data clock). To control these signals there are three fields available in the rw_mgm_ctrl register, mdio, mdc and mdoe (management data output enable). The mdoe signal selects the direction of the mdio pin. The mdio field selects the value on the mdio pin. The mdc field controls the mdc pin which is used as clock for the mdio pin regardless of direction. To read the value of the mdio pin the

mdio field in the r_stat register should be read. All three fields should be controlled by the software driver. The exact details of how to configure the MDIO interface depends on which PHY that is used.

21.4.5 Error and statistics counters

There are 10 different error and statistics counters. The purpose of these counters are to track certain statistics like number of frames with CRC errors and number of collisions. This is useful when analyzing a network. Below is a list of all counters available with a short explanation:

- crc_err

  This counter tracks the number of frames with CRC error. The counter is used to update the aFrameCheckSequenceErrors counter described in [IEEE8023].

- align_err

  This counter tracks the number of frames with alignment error. The counter is used to update the aAlignmentErrors counter described in [IEEE8023].

- oversize

  This counter tracks the number of oversized frames. The counter is used to update the aFrameToLongErrors counter described in [IEEE8023].

- congestion

  This counter tracks the number of otherwise correct frames that were not received due to a receiver overrun condition. The counter is used to update the aFramesLostDueToIntMACRcvError counter described in [IEEE8023].

- single_col

  This counter tracks the number of frames involved in exactly one collision. The counter is used to update the aSingleCollisionFrames counter described in [IEEE8023].

- mult_col

  This counter tracks the number of frames involved in multiple collisions. The counter is used to update the aMultipleCollisionFrames counter described in [IEEE8023].

- late_col

  This counter tracks the number of frames involved in late collisions. The counter is used to update the aLateCollisions counter in [IEEE8023].

- deferred

  This counter tracks the number of deferred transmit frames. The counter is used to update the aFramesWithDeferredXmissions counter in [IEEE8023].

- carrier_loss

  This counter tracks the number of transmit frames for which the carrier sense signal was not constantly present during the transmission. The counter is used to update the aCarrierSenseErrors counter described in [IEEE8023]. In full duplex mode the contents of this counter is undefined.

- sqe_err

  This counter tracks the number of transmit frames for which the sqe test signal was not recognized, 10 Mbit mode only. The counter is used to update the aSQETestErrors counter described in [IEEE8023].

Each counter is 8 bits wide and all 10 counters occupy a total of three status registers. Each counter will generate an interrupt when it reaches 128 and will saturate when it reaches 255. The counter values can be read with or without side effects. When reading a register with side effects, all counters in that register will be reset to zero.

21.4.6 Interrupts

There are 14 different interrupts that the Ethernet interface can generate, below is a list and a short explanation for each one:

Interrupt	Explanation
crc	Generated when the CRC error counter reaches 128.
align	Generated when the alignment error counter reaches 128.
oversize	Generated when the oversize counter reaches 128.
congestion	Generated when the congestion counter reaches 128.
single_col	Generated when the single collision counter reaches 128.
mult_col	Generated when the multiple collision counter reaches 128.
late_col	Generated when the late collision counter reaches 128.
deferred	Generated when the frame deferred counter reaches 128.
carrier_loss	Generated when the carrier loss counter reaches 128. This interrupt should be masked off while in full duplex mode.
sqe_test_err	Generated when the SQE test error counter reaches 128.
orun	Generated when receiver overrun has occurred.
urun	Generated when transmitter underrun has occurred.
exc_col	Generated when an excessive collision has occurred.
mdio	Generated when the MDIO pin is low. This interrupt should be masked off during normal transfers over the MDIO interface.

Table 21.5: *Ethernet interrupts*

21.4.7 Loop back mode

The Ethernet interface supports internal loop back mode. This means that packets can be passed directly from the transmitter to the receiver. The loop back mode can be enabled using the loopback field in the rw_gen_ctrl register.

21.4.8 Handshake protocol

The Ethernet interface can be configured to operate in handshake mode using the protocol field in the rw_gen_ctrl register. The mode implements a mechanism, allowing the Ethernet protocol to be used in a simplified manner, where the col and crs pins are used for requesting and acknowledging data transfers. This mode is intended for a direct connection between two units as most of the error handling and control mechanisms has been removed. Please contact Axis Communications AB for further details.

21.4.9 Phyclk pin

The **phyclk** pin can be configured to have one of three different functions, by writing to the phy field in the rw_gen_ctrl register:

21.4.9.1 25MHz clock output

A 25 MHz clock that can be used by the PHY is output on the **phyclk** pin. Set the phy field to mii_clk for this mode.

21.4.9.2 Transmit error

A transmit error signal can be output on the **phyclk** pin. Set the phy field to mii for this mode. It is possible to deliberately corrupt a frame by setting the tx_er bit in a DMA data descriptor meta data field (bit 1 of the user configurable meta data field).

21.4.9.3 Address recognized output

In this mode, the **phyclk** pin will signal if an incoming packet matches the station address. Set the phy field to mii_arec for this mode.

21.5 Hardware Interface

The Ethernet interface uses the Media Independent Interface (MII) as external hardware interface. The signals are described below. The ETRAX FS has two instances of the Ethernet interface. Ethernet port 0 is always available while port 1 is shared with other functions. For details on how the MII signals are mapped to the external pins please refer to 16.

21.5.1 External pin description

The direction of the signals in the Ethernet MAC interface is given below. Explanations:

- **In** Input signal, signal from PHY to MAC.

- **Out** Output signal, signal from MAC to PHY.
- **I/O** Bidirectional signal.

The Ethernet MAC interface has the following external pins:

21.5.1.1 Transmitter signals

Signal	Direction	MII usage
txclk	In	Transmit clock
txd[0]	Out	Data out, bit 0
txd[1]	Out	Data out, bit 1
txd[2]	Out	Data out, bit 2
txd[3]	Out	Data out, bit 3
txen	Out	Transmit enable
phyclk	Out	25MHz clock / Transmit error / Address recognized

Table 21.6: *Transmitter signals*

21.5.1.2 Receiver signals

Signal	Direction	MII usage
rxclk	In	Receive clock
rxd[0]	In	Data in, bit 0
rxd[1]	In	Data in, bit 1
rxd[2]	In	Data in, bit 2
rxd[3]	In	Data in, bit 3
rxdv	In	Receive data valid
rxer	In	Receive error

Table 21.7: *Receiver signals*

21.5.1.3 Network status signals

Signal	Direction	MII usage
crs	In	Carrier sense
col	In	Collision

Table 21.8: *Network status signals*

21.5.1.4 Transceiver management signals

Signal	Direction	MII Usage
mdc	Out	Management clock
mdio	I/O	Management data

Table 21.9: *Transceiver management signals*

21.5.2 Reset behavior

At reset, all pins of Ethernet port 0 will be defined within 18 **clk** cycles after the rise of **rst_n**. All outputs will then be low and the **e0mdio** pin will be turned off until the Ethernet interface is started. Since Ethernet port 1 is shared with other functions, all of its pins will be turned off immediately at reset.

21.5.3 Signal timing

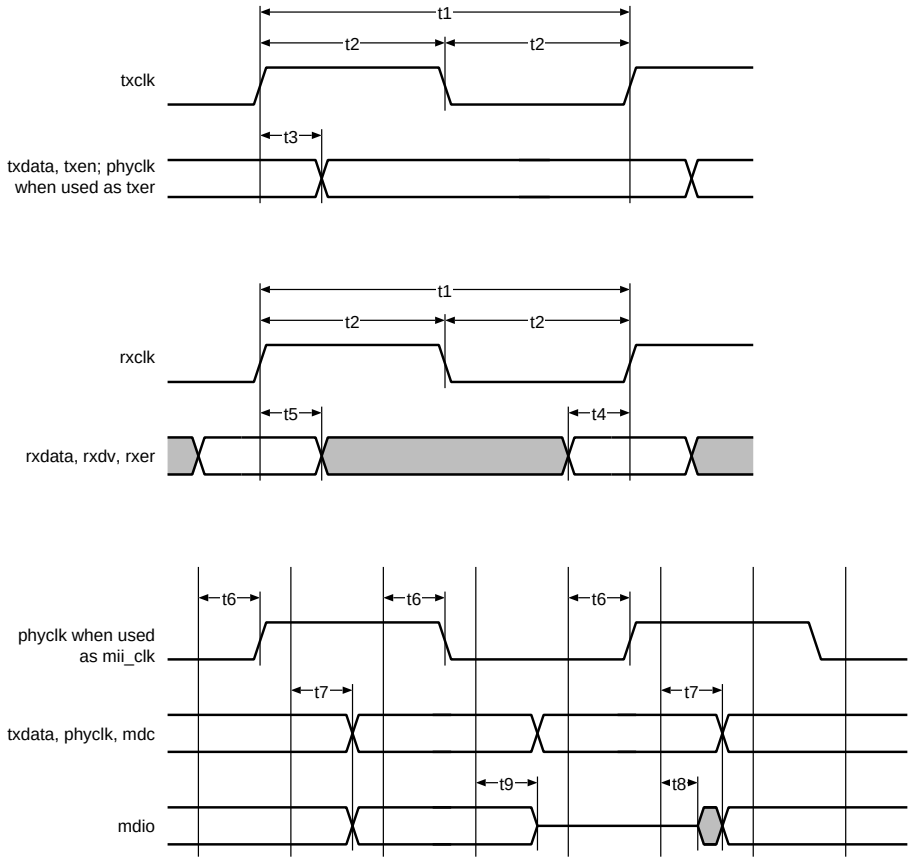


Figure 21.4: *Network interface timing*

Nbr	Name	Description	min	typ	max	unit
t1	tcp	txclk and rxclk clock period	35	-	-	ns
t2	tcw	txclk and rxclk pulse width	10	-	-	ns

t3	tdd	txdata and txen delay from txclk, phyclk delay from txclk when used as txer	3	-	10	ns
t4	tds	rxdata, rxdv and rxer setup time to rxclk	1	-	-	ns
t5	tdh	rxdata, rxdv and rxer hold time from rxclk	2	-	-	ns
t6	tcd	phyclk delay from internal clock, when used as 25MHz clock output	2	-	8	ns
t7	tmd	txdata, phyclk, mdc and mdio delay from internal clock when controlled by the rw_mgm_ctrl register	2	-	8	ns
t8	tme	mdio output enable from internal clock	3	-	-	ns
t9	tmz	mdio turn off time from internal clock	1	-	7	ns

Table 21.10: *Ethernet interface timing*

21.6 Software Interface

In the following sections a detailed description of how to configure and use the Ethernet interface is given.

21.6.1 Configuration registers

There are a number of registers which may be configured by software. In addition to these registers there are also a number of status registers which are software read-only. All registers are described in 25.12. C programing macros that can be used when accessing the registers are available in [ETH_H].

21.6.2 DMA and pin configuration

Before the Ethernet interface can be started, it must be connected to the corresponding internal DMA channels, see 10, and the DMA channels must be configured and started, see 5. Ethernet interface 0 has dedicated pins. For Ethernet interface 1, the signals are multiplexed with other functions, and the pins must be configured before they are used, see 16.

21.6.3 DMA descriptors

The Ethernet interface requires one context descriptor for each DMA channel and at least one data descriptor for each channel. For details on how to configure the DMA and set up the descriptors, please refer to 5.

21.6.3.1 Transmitter

When configuring the out channel descriptors the last descriptor of a packet must have the EOP bit and wait bit set. This is signalling to the interface when the last byte of the data payload is being delivered. The interface continues by sending the four bytes

containing the calculated CRC. The interface will then command the DMA to move on to the next packet or, for example after a collision, to restore the DMA list position.

21.6.3.2 Receiver

When the Ethernet interface is receiving an incoming bit stream, it will write it to memory via the DMA. When the bit stream reaches its end, the Ethernet interface will detect it and stop writing data to the DMA in channel. It will then start comparing the received CRC with its own calculated CRC. If the bit stream is received successfully the Ethernet interface will store the current DMA context descriptor, signal EOP and move on to the next descriptor. If the reception of the bit stream is unsuccessful, the DMA will restore the context descriptor and wait for a new packet.

21.6.4 Software reset

The interface state can be reset using the en field in the rw_gen_ctrl register, thus allowing the interface to be reconfigured. In order to not reset the interface while it is transmitting a frame, the cancel field in the rw_tr_ctrl register can be used. This bit will stop the transmitter after the current transmission attempt (if any). The transmitter will then enter the excessive retry state. In order to get the transmitter to resume operation the clr field in the rw_clr_err register must be set.

21.6.5 Configuration example

Setup of the Ethernet interface can be divided into two parts:

- Setup of DMA data descriptors
- Configuration of Ethernet interface registers

The focus of this section is on how to configure the Ethernet interface, examples of how to setup a basic DMA descriptor list can be found in 5. The following sections show a typical configuration of the Ethernet interface's registers, after which the interface is ready for both reception and transmission of frames.

21.6.5.1 rw_gen_ctrl

Field	Value	Explanation
loopback	no	Normal mode
protocol	ether	Ethernet (CSMA/CD)
phy	mii_clk	MII with 25MHz clock output on the phyclk pin
en	yes	Enable interface

Table 21.11: *rw_gen_ctrl configuration example*

21.6.5.2 rw_rec_ctrl

Field	Value	Explanation
max_size	size1518	Max packet size = 1518 bytes
duplex	full	Full duplex
bad_crc	discard	Discard frames with CRC error
oversize	discard	Discard oversized frames
undersize	discard	Discard undersized frames
broadcast	rec	Receive broadcast frames
individual	no	The GAT only matches group addresses
ma1	no	Enable MA1 address
ma0	yes	Enable MA0 address

Table 21.12: *rw_rec_ctrl configuration example*

21.6.5.3 rw_tr_ctrl

Field	Value	Explanation
ignore_crs	no	Measure interframe gap from last neg. edge of crs.
hsh_delay	no	No delay (Don't care in Ethernet CSMA/CD mode)
cancel	no	Don't cancel pending transmission attempts
ignore_col	no	Use collision detection (Don't care in full duplex)
retry	yes	Enable transmission retries
pad	yes	Add padding of short frames in hardware
crc	yes	Add CRC in hardware

Table 21.13: *rw_tr_ctrl configuration example*

21.6.5.4 rw_ma0_lo

Lower 32 bits of the Ethernet MAC address, station 0.

21.6.5.5 rw_ma0_hi

Upper 16 bits of the Ethernet MAC address, station 0.

21.6.5.6 rw_ma1_lo

Lower 32 bits of the Ethernet MAC address, station 1.

21.6.5.7 rw_ma1_hi

Upper 16 bits of the Ethernet MAC address, station 1.

21.6.5.8 rw_ga_lo

Lower 32 bits of the group address table.

21.6.5.9 rw_ga_hi

Upper 32 bits of the group address table.

21.6.5.10 rw_test_ctrl

Field	Value	Explanation
backoff	no	Used during testing, backoff test mode disabled.
snmp	no	Used during testing, SNMP test mode disabled.
snmp_inc	dont	Used during testing, SNMP test mode increment clock.

Table 21.14: *rw_test_ctrl configuration example*

Chapter 22

General I/O

22.1 References

Reference	Description
[REGS]	Mode registers, chapter 25.13
[MACROS]	http://developer.axis.com
[PINMAP]	Pinout, chapter 16

Table 22.1: *References*

22.2 Overview

The General I/O block controls outputs and reads data on the 80 configurable I/O pins of the ETRAX FS chip, ports **pa**, **pb**, **pc**, **pd** and **pe**. The General I/O is multiplexed with other I/O functions according to [PINMUX].

The interrupt inputs on the 8 pins of port **pa** are also handled by the General I/O block.

22.3 Functional description

22.3.1 General I/O ports

The general I/O is partitioned into five ports: **pa**, **pb**, **pc**, **pd** and **pe**. Port **pa** is 8 bits wide, and ports **pb** to **pe** are each 18 bits wide. This partitioning matches the partitioning of the configurable I/O pins in the ETRAX FS chip, see 16.

Each port has one read/write register for output data, and a separate read-only register for input data. Reading the output data register just returns the data written to the register, while the input data register returns the actual data on the pins. There is also one output enable register per port, with an individual direction control field for each signal in the port.

The output data registers drive the output buffers directly without any further logic except the pin multiplexing described in 16. The input data registers read the values on the pins directly without any intermediate logic.

22.3.2 Interrupts on port pa

The 8 pins on port **pa** can serve as interrupt inputs. Each of the 8 interrupts can be individually disabled or set to one of 6 different interrupt modes:

- Level triggered active high
- Level triggered active low
- Interrupt always set
- Positive edge triggered
- Negative edge triggered
- Triggered on both edges

In the level triggered modes, the interrupt will be set as soon as the input goes to the active level. The interrupt will be active until it is acknowledged in the rw_ack_intr register, even if the input goes to the inactive level. If the input is still at the active level when acknowledged, the interrupt will immediately be set again.

The set mode always keeps the interrupt set regardless of the level on the input pin. This mode is intended mainly for test purposes, but it can also be used as a software initiated interrupt if the pin is not used as an input for an external interrupt. The interrupt will be active until it is acknowledged in the rw_ack_intr register. If the mode is still set when the interrupt is acknowledged, the interrupt will immediately be set again.

The edge triggered modes will set an interrupt when the input makes a transition in the specified direction. The interrupt will be active until it is acknowledged in the rw_ack_intr register.

22.3.3 Reset behavior

After a system reset, all General I/O ports are set to inputs, and all General I/O interrupts are turned off. The content of the output data registers is undefined.

22.4 Hardware interface

22.4.1 General I/O signals

The General I/O is connected to the configurable I/O ports of the ETRAX FS chip:

ETRAX FS Pin	General I/O Signals

pa[7:0]	General I/O port pa data + interrupts
pb[17:0]	General I/O port pb data
pc[17:0]	General I/O port pc data
pd[17:0]	General I/O port pd data
pe[17:0]	General I/O port pe data

Table 22.2: *Configurable I/O Ports*

22.4.2 Data output timing

Outputs and output enables are controlled directly from the General I/O mode registers, with only the combinatorial delay through 16 and the output buffers.

The timing is given below. The timing figures assume a load of 50 pF on the pins.

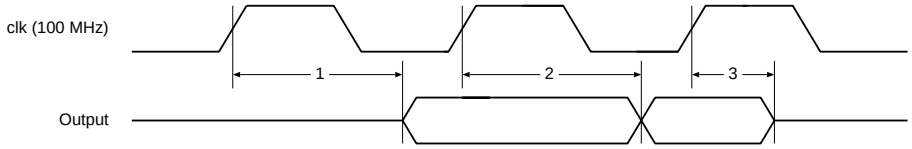


Figure 22.1: *Data Outputs*

Parameter	Explanation	Min	Max	Unit
1	Turn on time from clock	2	9	ns
2	Data delay time from clock	2	8	ns
3	Turn off time from clock	1	8	ns

Table 22.3: *Data Outputs*

22.4.3 Data input timing

Inputs are read directly from the pins, with only the combinatorial delay through the input pads.

The timing is given below.

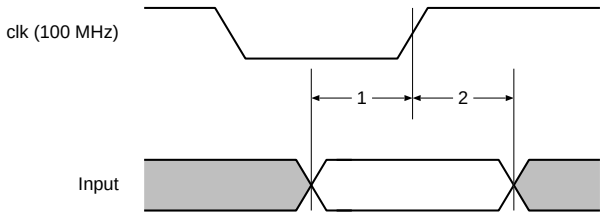


Figure 22.2: *Data Inputs*

Parameter	Explanation	Min	Max	Unit
1	Set up time to clock	4	-	ns
2	Hold time from clock	0	-	ns

Table 22.4: *Data Inputs*

The inputs are synchronized internally, and the setup and hold times only need to be taken into consideration if detection in a specific clock cycle is required.

22.4.4 Interrupt input timing

The interrupts are synchronized with double flip-flops before they are forwarded to the r_intr register.The interrupt timing is given below.

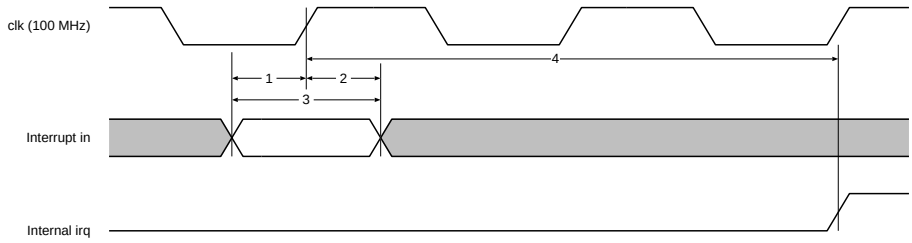


Figure 22.3: *Interrupt Inputs*

Parameter	Explanation:	Min	Nom	Max	Unit
1	Set up time to clock	4	-	-	ns
2	Hold time from clock	0	-	-	ns
3	Pulse width	12	-	-	ns
4	Detection latency	-	20	-	ns

Table 22.5: *Interrupt Inputs*

The setup and hold times only need to be taken into consideration if detection in a specific clock cycle is required.

22.5 Software interface

The General I/O is controlled through a set of registers, see 25.13 for details. To access the registers, fields and register constants from a C program, a set of macros is defined in [MACROS].

Before General I/O signals can be used as outputs, they must be configured as General I/O in the pin multiplexer for the configurable I/O pins, see 16. Input values are always available for reading through the General I/O, regardless of the pin multiplexer configuration.

22.5.1 Programming considerations

22.5.1.1 Port read after write

Reading the value on a pin immediately after changing the output on the same pin may give an undefined result.

To read the new value on the pin, the signal must propagate from the output data register to the input data register within one clock cycle. This may or may not be the case, depending on the external load on the pin and other factors like e.g. temperature. With an external load of 50 pF or less, an inserted delay of one clock cycle between the write and the read is sufficient to always read the updated value on the pin.

22.5.1.2 Acknowledge of level triggered interrupts

In the level triggered interrupt modes, the interrupt will be set whenever the input is at the active level. In these modes, you will typically need to acknowledge the interrupts in the external module driving the pin first, and then, after an appropriate delay, acknowledge the General I/O interrupt through the rw_ack_intr register.

When the required delay is calculated, the two cycle interrupt detection latency, see 22.4.4, must be taken into consideration, as well as any delay in the external unit.

Chapter 23

Synchronous Serial Interface

23.1 References

Reference	Description
[SSER_REG_DOC]	Registers, chapter 25.41
[MACROS]	http://developer.axis.com
[PINSPEC]	Pinout, chapter 16
[I2S]	I^2S bus specification
[IEC]	The IEC60958 standard, http://www.iec.ch
[SPI]	Motorola MC68HC08AS32A datasheet
[OKI]	OKI Semiconductor MSM7731-02 datasheet
[MAX1202]	MAXIM MAX1202 datasheet
[I2C]	I^2C bus specification
[ATMEL]	Atmel AT45CS1282 datasheet

23.2 Definitions

SSI Synchronous Serial Interface (this block)

GIO General Input/Output

GI General Input

I2C Inter-IC Control bus

I2S Inter-IC Sound bus

LSB Least significant bit

MSB Most significant bit

SPI Serial Peripheral Interface

synchronous The term synchronous refers to the existence of a dedicated clock signal in the SSI interface. All supported protocols (except IEC60958) have a clock

signal that is used to sample or synchronize the other signals, and is connected between the transmitting and the receiving unit.

Serial Serial means that data is sent one bit at a time over one wire. Some protocols have been extended to two or three data wires in the same direction.

Clock The serial clock (also called bit clock) signal toggles two times during the transmission of one data bit (see figure 23.2). In most protocols, the transmitter uses one of the edges to output its data, and the receiver uses the opposite clock edge to lock the incoming data in a flip-flop. The clock signal can be produced by the transmitting or the receiving unit, or can come from an external source.

Master The unit deciding when data transfers shall occur. Need not be the unit that produces the clock. The master always produces the frame signal (if a frame signal exists).

Slave A unit communicating at the demand of another unit.

Clock gating A gated clock is a clock where some pulses have been masked away. In the SSI context, this means that the clock only toggles when there are data bits to transfer. A gated clock is always created by the master.

Frame A physical signal or an SSI-internal event marking the beginning and, optionally, the duration of a serial word.

Word A word is any collection of binary digits that form one unit. In this chapter, different kinds of words are discussed. When there can be ambiguities, a word that is transferred over the serial interface is denoted a *serial word* or a *sample*, and a word stored in a memory is called a *memory word*. The number of bits in a word is called its word length or its sample size. The word length of a serial word is the number of data bits associated with one frame event.

Bulk mode In bulk mode, data is sent whenever data is available. Typical examples are: Register read or write commands to an external device, and a print job to a printer.

Isochronous mode In isochronous mode, data is sent at periodical intervals, controlled by a timer. If no data is available in the transmitter when the timer expires, an error occurs (underrun). A typical example is any stream of real-time samples, e.g., audio or video. The rate at which serial words start is called the word rate.

Flow control Flow control is a means for the receiver to tell the transmitter to stop and continue transmission. Without flow control, overrun errors might occur if the receiver is unable to process all the received data quickly enough.

23.3 Overview

The ETRAX FS includes two on-chip instances of the SSI block. The SSI is designed to interface to many simple synchronous serial protocols, with an emphasis on protocols used to transport digital audio between ICs on a circuit board. A connection example can be seen in figure 23.1.

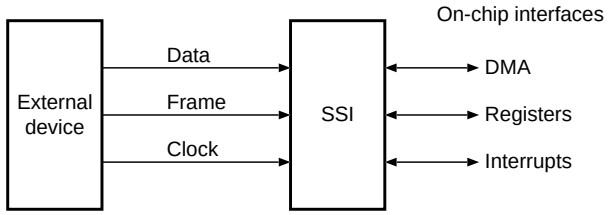


Figure 23.1: *SSI connection*

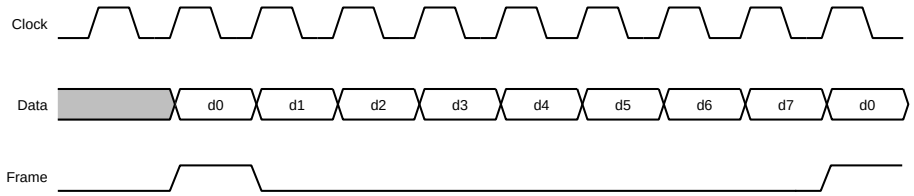


Figure 23.2: *Typical unidirectional synchronous serial protocol*

Most of the protocols that the SSI handle are very simple (See figure 23.2), but many variations of these protocols are supported.

The SSI contains one transmitter and one receiver block. There is also some common logic for generating and decoding frame signals, and a clock generator.

Up to six external pins can be used by the SSI, and the maximum supported clock frequency is 110 MHz.

23.4 Functional description

23.4.1 Operating modes

Below is a brief introduction to the different main modes. For further information see the respective subsection for each mode below.

Lowspeed mode Lowspeed mode is designed to support the following:

- I2S, master, slave, internal and external clock.
- SPI, similar to the MC68HC08 microprocessor SPI interface (all modes), and specifically compatible with the MAX1202 A/D converter.
- Compatibility with both audio and control interfaces of the Oki MSM7731 acoustic echo canceller.
- A subset of I2C.
- Many other simple synchronous serial protocols.

Lowspeed mode operates up to 16.67 MHz. There can be up to two simultaneous I2S output streams, and up to three simultaneous I2S input streams. The word length can be set with bit precision between 4 and 64 bits, and clock gating is possible.

Highspeed mode Highspeed mode operates up to 100 MHz with the internal clock, and 110 MHz with an external clock. The word length can be controlled with nibble precision, up to 256 bits. This mode also supports fast SPI mode as defined by [ATMEL] and SPI as used by SDCARD/MMC.

Wiresave mode Wiresave mode operates up to 100 MHz with the internal clock, and 110 MHz with an external clock. There is one data output, with up to approximately 100 Mbit/s transmission. There can also be up to three data inputs, enabling up to approximately 300 Mbit/s reception.

Wiresave mode has common or separate transmit/receive clocks and does not support clock gating. There is specialized framing and flow control and fixed word length at 16 bits per word, LSB first transmission. Wiresave mode requires only two pins for output (clock, data), and requires only #inputs + 1 pins for input (clock, data0, data1, data2).

IEC60958 mode IEC60958 mode supports consumer and professional modes with 32, 44.1, 48 or 96 k samples per second, in stereo. Bit-level data formatting has to be performed by software. 32 bits per word are always used, of which up to 24 are used for audio data. IEC60958 mode uses isochronous communication. Only one data pin is required for the receiver and one for the transmitter, but the transmitter also requires an external clock to operate.

In all main modes except IEC60958, the transmitter can run in either bulk or isochronous mode, selected with the rw_tr_cfg.rate_ctrl mode register field. For further information on this topic see section 23.4.2.

The idle value of the **data** pin is not defined for protocols other than wiresave mode. If a defined value is required, use the rw_tr_cfg.data_pin_use.gio0 or rw_tr_cfg.data_pin_use.gio1 settings.

23.4.1.1 Lowspeed mode

This is the most flexible main mode of the SSI. Lowspeed mode is selected by setting rw_tr_cfg.mode or rw_rec_cfg.mode to lospeed and by clearing the IEC60958 fields. Also clear the field rw_tr_cfg.use_md in the transmitter. Most options for lowspeed mode are explained in the sections on clocking, reset, frame signalling and flow control. A few more are covered here.

By setting the rw_tr_cfg.dual_i2s field, two I2S transmitters are enabled[1]. They will share the same clock and frame signal and have one data output each. In other words, data will be output in exactly the same clock cycles on both outputs. Similarly, two or three I2S inputs can be enabled by setting the fields rw_rec_cfg.slave2_en and/or rw_rec_cfg.slave3_en (slave 3 without slave 2 is not a valid mode).

[1] When using both I2S transmitters in mode register driven mode, one word of dummy data might be sent as the first word after the SSI is enabled. If this can not be tolerated, use DMA mode instead.

The available pins limit the total number of I2S inputs and outputs. Two I/O pins and one pure input pin are available for data transfer. Note that each I2S input/output normally carries one stereo audio channel or two mono channels. For information on how to set up data in memory when using multiple receivers or transmitters, see section 23.6.1.

The fields rw_tr_cfg.sh_dir and rw_rec_cfg.sh_dir select whether the MSB or the LSB is sent first over the serial line. When serial word size is less than or equal to 16 bits, this does not affect the way data is organized in memory, but for larger word sizes it does. See section 23.6.1 for more details.

23.4.1.1.1 SPI

The [SPI] datasheet has been used as the basis for this implementation. According to the [SPI], the maximum clock frequency is 8.4 MHz, but this SSI implementation can use somewhat higher frequencies. As the clock frequency approaches 10 MHz, however, not all timing figures in [SPI] will be fulfilled (e.g. clk high/low time will fall below 50 ns, setup time of miso will fall below 45 ns, access time of slaves must be shorter than 40 ns). Also note the high speed SPI mode in section 23.4.1.2.2. A configuration example for SPI is available in section 23.7.2.

23.4.1.1.2 OKI MSM7731 microprocessor interface

According to [OKI], the minimum period of the microprocessor interface (not the audio interface) is 100 ns (10 MHz). With this SSI however, it is not possible to use a shorter period than 140 ns (7.14 MHz) while still fulfilling the other timing requirements of the interface. This should have no practical implications.

23.4.1.1.3 MAX1202 A/D converter interface

The data sheet [MAX1202] specifies a maximum serial interface clock frequency of 2.0 MHz. However, due to the slow access time of the MAX1202, the maximum clock frequency that can be guaranteed to work with the SSI without detailed timing analysis is 1.92 MHz.

An example of MAX1202 configuration is provided in section 23.7.3.

23.4.1.1.4 I2C

The SSI can act as an I2C master in single master environments. Note however that the SSI is not capable of ”clock stretching” (see [I2C]). If the device uses clock stretching, a lower communication frequency might avoid clock stretching to occur. An attempt by a device to use clock stretching will not cause electrical problems, only logical failure.

Further, the I2C mode of the SSI is not fully automatic. The CPU has to generate the start and stop conditions, but the SSI can take care of the rest of the communication.

Since there is no maximum limits in the involved timing, this is generally not much of a problem.

Both standard-mode (100 kbit/s) and fast-mode (400 kbit/s) communication is possible, but in fast mode the clock frequency has to be lowered to 384 kbit/s not to violate the tLOW time of the I2C specification (fast-mode I2C requires a non-square clock waveform to run at 400 kbit/s).

An example of how to connect and program the SSI for I2C communication can be found in section 23.7.4.

23.4.1.2 Highspeed mode

This mode is, like lowspeed mode, quite generic, but it lacks some of the low speed mode features:

- Word length with nibble precision instead of bit precision, on the other hand word length can be up to 256 bits.
- The normal clock gating features (in and out) are not available. There is another feature available to gate an internal clock output, though.
- Only positive edge clocking is possible when using internal clock at 100 MHz (the fields rw_tr_cfg.clk_pol, rw_rec_cfg.clk_pol and rw_frm_cfg.clk_pol have no effect at 100 MHz internal clock).

23.4.1.2.1 Special output clock gating feature

Since the normal clock gating features mentioned elsewhere in this document are not available in highspeed mode, other semi-automatic clock gating features have been added. They are controlled by the register fields rw_extra.clkon_en, rw_extra.clkoff_en and rw_extra.clkoff_cycles and enables the clock to be turned on once, and turned off once.

Note that these features only turn on/off the output clock. This means that the transmitter and receiver will start and continue unaffected. To make the first output data bit appear to be clocked out by the first edge of the output clock, set rw_frm_cfg.tr_delay to 2 (assuming internal frame source, external frame source will not work with this clock gating feature).

Further, set up the transmitter DMA with just the needed amount of data, set EOP at the end and set the rw_tr_cfg.eop_stop field, so that a tidle interrupt is generated when finished and so that no extra data is output after the clock is turned off.

The receiver will also continue receiving as usual even though there is no external clock, and bogus data will be fed to the DMA after the normal data.

Another thing to keep in mind is that the clock will not be turned off if the DMA can not supply or remove data fast enough. Therefore the communication will be corrupted in that case (i.e. bulk mode will not work as intended). As a consequence isochronous communication is recommended, then underrun/overrun interrupts will be issued in case of data corruption.

The clock will not be turned off between words either, so words should usually be sent back-to-back in this mode.

23.4.1.2.2 Fast SPI master mode

Communication with a fast SPI slave, such as an Atmel flash (see [ATMEL]), using SPI or ”RapidS” is possible. This communication utilizes the special output clock gating feature described in 23.4.1.2.1. Also note that this communication mode is semi-automatic, CPU intervention is required for each *burst*. To make communication efficient it is therefore desirable to use as long bursts as possible. See further the example in section 23.7.5 on Atmel flash communication.

23.4.1.3 Wiresave mode

The wiresave mode has been implemented to enable high-speed communication in a simple way, consuming as few I/O pins as possible, without using any special I/O cells or signalling standards. It is a proprietary mode and will only be useful for connecting to other devices specifically built for it.

Both receiver and transmitter can be configured to use wiresave mode. The transmitter can only use one data pin, while the receiver can use up to three data pins, enabling 300 Mbit/s reception speed.

To enable wiresave mode, set the fields rw_tr_cfg.mode or rw_rec_cfg.mode to wiresave. Further set the fields rw_tr_cfg.sample_size or rw_rec_cfg.sample_size to 3 (ie 4 nibbles == 16 bits), and fields rw_tr_cfg.sh_dir or rw_rec_cfg.sh_dir to lsbfirst.

To enable more than one receiver data stream, set the fields rw_rec_cfg.slave2_en (for two streams) and also rw_rec_cfg.slave3_en (for three streams). All three receiver data streams and the transmitter can be used simultaneously.

If only one clock is used with bidirectional communication, the external unit needs to have some type of timing adjustment logic for the data signal(s) travelling in the opposite direction compared to the clock. No such timing adjustment features are available in this SSI.

If the external unit is equipped with programmable timing adjustment logic, it might be possible to communicate without transmitting any clock at all. The requirements are that the SSI:s of both communicating chips are synchronous to each other (by using the same reference clock), and that there are enough timing margins.

23.4.1.3.1 Mode of operation

Wiresave mode uses start bits, similar to asynchronous serial ports. The idle value of the data line is ’1’. One cycle of value ’0’ indicates that a word is starting. The value in the next cycle determines if the data word to come is an ordinary data word (0) or a metadata word (1). After the metadata bit follows 16 data bits, and when these have been transferred the data line either goes back to its idle value, or a new start bit is transmitted (stop bits can be used if desired). I.e., without stop bits, 18 bits are consumed to transfer a payload of 16 bits.

If more than one data line is used, only data to receiver 0 has to have a defined idle value and only this line transfers start and metadata bits. The other data inputs need only be defined when transferring actual data, and data is transferred in the same clock cycles as on data line 0 . When metadata is transferred, only receiver 0 is used and the other data lines are ignored. The data from the receivers ends up in memory in the order described in section 23.6.1.

23.4.1.3.2 Metadata use

The metadata is handled differently from ordinary data inside the SSI, and there are the following uses for it[2]:

flow control There is logic in the SSI to prevent buffer overflow, by sending and receiving xon/xoff metadata codes. See section 23.4.3.2 for details.

end-of-packet signalling At DMA eop, the transmitter will inspect the data descriptor metadata. As long as the metadata is not 'tx_null', it will be transmitted as a metadata word after the last ordinary data and an md_sent interrupt will be issued. Take care not to send metadata with undesired side effects, see section 23.6.5.

A received metadata word marks the end of a packet, and the receiver DMA handles the end-of-packet condition as described in 5.4.1.2. Additionally, the md_rec interrupt will be issued, if enabled.

As can be seen, the net effect of transmitting and receiving non-special metadata is that an end-of-packet condition will be forwarded over the communication link.

receiver manual eop and stop When software uses the fields rw_rec_cfg.stop or rw_rec_cfg.force_eop to (stop transfer and) force an eop to the DMA, the metadata that ends up in the DMA descriptor is listed in table 23.10 in section 23.6.5. This metadata is never transferred over the serial line, it is just written to the receiver DMA descriptor. This metadata will be written to the DMA descriptor in all modes at stop or at a forced eop to the DMA, not just wiresave mode.

23.4.1.4 IEC60958 mode

The SSI contains one IEC60958 receiver and one transmitter, i.e. one IEC60958 data stream can be transferred in each direction. The transmitter and receiver can be used independently.

To enable IEC60958 mode, set rw_tr_cfg.use60958 and/or rw_rec_cfg.use60958. Select lowspeed mode, 32 bits per word, LSB first. If transmitter is used, set rate control to isochronous, select external clocking, and apply an external clock to either **clk** or **ext_clk**, as selected by rw_cfg.clk_in_sel. Configure the field rw_tr_cfg.iec60958_ckdiv to match the frequency of the external clock, according to the following table:

Fsamp	ckdiv=0	ckdiv=1	ckdiv=2	ckdiv=3

[2] If the field rw_tr_cfg.use_md is unset, no metadata whatsoever will be transmitted over the serial line.

32 ksa/s	4.096	8.192	12.288	16.384
44.1 ksa/s	5.6448	11.2896	16.9344	22.5792
48 ksa/s	6.144	12.288	18.432	24.576
96 ksa/s	12.288	24.576	n/a	n/a

Table 23.2: *IEC60958 ext clock frequencies, MHz*

The maximum external clock frequency in IEC60958 mode is 25 MHz.

For the receiver, configure the field rw_rec_cfg.iec60958_ui_len as described in section 23.4.1.4.2 below. Then enable the SSI as usual. The transmitter will start transmitting within a few clock cycles and the receiver will start writing data to its mode register or to memory after receiving a correct preamble.

23.4.1.4.1 Data format

In IEC60958 mode the word length is always 32 bits, and the LSB is sent first. Figure 23.3 presents the use of the bit fields in each word.

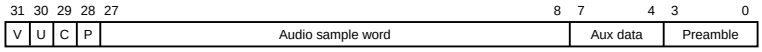


Figure 23.3: *Data format*

Explanation:

V: Validity flag

U: User data

C: Channel status

P: Parity bit

The data has to be presented to the SSI transmitter and decoded from the SSI receiver in this format. Below is a brief description of these bit fields together with some implementation specific information. For more in-depth information, refer to the [IEC] standard.

Preamble There are three different preambles available, called B, M and W (sometimes called Z, X and Y respectively). The preambles correspond to unique code sequences on the transmission medium, which never occur during transmission of ordinary data. To transmit these preambles, or to decode which one was received, insert the following preamble values into the data stream or compare the received preambles to these values:

Code (bits 3-0)	Preamble
0b0000 = 0x0	”B”

0b0100 = 0x4	”M”
0b1000 = 0x8	”W”

Table 23.3: *IEC60958 preambles*

The remaining codes are not defined and will never be written to memory by the receiver. Do not insert other codes in the preamble field of data to transmit.

Normally preamble ”M” denotes primary or ’left’ channel, and ”W” denotes secondary or ’right’ channel, samples. ”M” and ”W” words are interleaved one by one, except that every 192:nd ”M” preamble is changed to a ”B” preamble instead, to mark the beginning of a new channel status block, see below.

Audio sample word The audio sample word normally contains an audio sample in linear 2’s complement format. The LSB is sent first, at bit position 8. If less than 20 bits are available, the LSBs are padded with zeros.

Auxiliary data Can be used to transmit 24-bit samples, in that case the audio sample word extends over these bits and the audio LSB is at bit position 4. Other uses are defined in the standard.

Validity flag ’0’ marks that the data is valid and reliable. This bit is recommended to be set to ’1’ if transmitting data not suitable for linear PCM decoding to audio.

User data The default value of this bit is ’0’. Other uses of this bit is described in the standard.

Channel status The channel status is transmitted in 192-bit blocks. The first bit is transferred in the word with preamble ”B”. This first bit also indicates ”consumer” (0) or ”professional” (1) application. The second channel status bit indicates linear PCM samples (0) or other data (1). As can be seen, the CPU is responsible for distributing and gathering of these 192-bit blocks over the data words. For more information on the channel status, see the [IEC] standard.

Parity bit Bits 4 to 31 of each word shall have even parity, i.e., they shall contain an even number of ones and an even number of zeros. Calculation and checking of parity is done automatically by the SSI, i.e, when transmitting, the value of the parity bit in the data stream to the transmitter is don’t care, and when receiving the SSI signals an r958err interrupt if the received parity is wrong.

23.4.1.4.2 IEC60958 receiver data rate detection

The SSI, in conjunction with suitable software, is able to detect the data rate of an incoming IEC60958 data stream. The mode register field rw_rec_cfg.iec60958_ui_len shall contain a value that corresponds to the number of 100 MHz clock cycles per IEC60958 ”unit interval”. The following table gives suitable values for the common data rates:

Sampling rate	value
32 kHz	31
44.1 kHz	23

48 kHz	21
96 kHz	11

Table 23.4: *IEC60958 ui_len values*

If the sampling rate is known beforehand, just write the correct value to the mode register field. If the incoming sample rate is unknown, it can be found by trial-and-error: Start with a low iec60958_ui_len value, and start the receiver. After a short while, an r958err interrupt will be issued if the iec60958_ui_len value is too low, then try a higher value. Note, however, that a too high value may not produce an r958err interrupt. With a too high iec60958_ui_len value, data words may be missed and silently discarded. The receiver can receive an IEC60958 data stream with any non-standard sampling frequency from 32 kHz up to 96 kHz, if an appropriate iec60958_ui_len value is used.

23.4.2 Frame events and frame signals

Note: This chapter is relevant, at least in part, regardless of whether an external frame signal is used or not.

Most commonly, a separate pin on the port is used for frame signalling. The frame signal can be an input or an output.

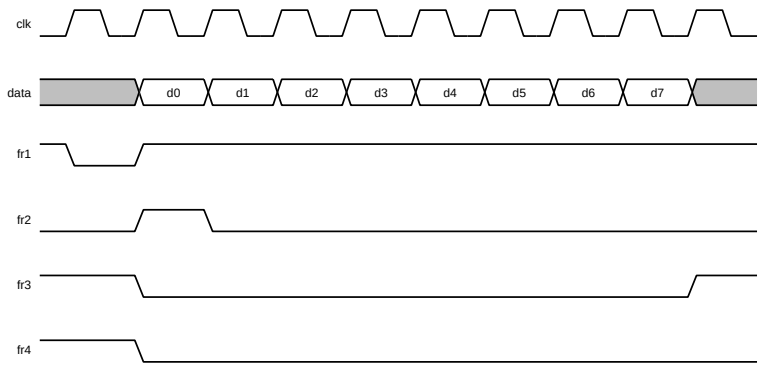


Figure 23.4: *Typical frame signals*

Figure 23.4 shows four typical frame signals. They can be characterized as follows:

- fr1: Negative edge-type frame signal, active one cycle before data.
- fr2: Positive edge-type frame, active at the same time as the first data bit.
- fr3: Negative level-type frame.
- fr4: Edge-type frame, both edges (the frame signal will go high again at the start of the next data word).

The clock edge generating the frame signal can be selected freely for the frame unit relative to the receiver and transmitter. The signal neg-ck-fr-out in figure 23.5 below shows the effect of choosing a different clock edge for the frame signal compared to the data. Of course, the same clock signal must be used for both the unit generating frame and the unit utilizing the frame event, even though polarity can differ.

23.4.2.1 Frame events and their sources

Internally in the SSI, 'frame events' are used to control when a word starts. A frame event is a one-cycle event notifying the exact start of a word. Frame events are produced by: a) an internal frame timer (isochronous mode), b) a frame input signal, and c) the transmitter in bulk mode.

When the receiver or transmitter inside the SSI receives a frame event, they start receiving or transmitting a word. They also pass on the resynchronized frame event, for use e.g. by an output frame generator or clock gating unit. The mode register fields rw_tr_cfg.frm_src, rw_tr_cfg.rate_ctrl, and rw_rec_cfg.frm_src select one of the three sources of frame events for the transmitter and receiver:

internal frame timer The internal frame timer (isochronous mode) is set up using the rw_frm_cfg.wordrate field, and generates frame events periodically with a frequency selected by the mentioned mode register. Set the frame source of receiver and/or transmitter to intern to use the internal frame timer.

external frame input signal In case of frame input, a frame signal fed to the SSI is decoded, and the decoding is controlled by the mode register fields rw_frm_cfg.type and rw_frm_cfg.level. To select an external frame input signal, set the frame source of receiver and/or transmitter to ext.

transmitter bulk mode In transmitter bulk mode, frame events are generated by the transmitter itself as soon as, and only when, data is available for transmission. Bulk mode for the transmitter is selected by the register field rw_tr_cfg.rate_ctrl. The receiver can also be set to receive at the same time as the transmitter transmits in bulk mode. To enable this feature, set rw_rec_cfg.frm_src to tx_bulk.

23.4.2.2 Frame output signal

When a frame signal is output, its shape is configured with the mode register fields rw_frm_cfg.type, rw_frm_cfg.level (same as for frame input decoding) and rw_frm_cfg.early_wend.

The effect of type on the output can be seen in figure 23.4, where fr1, fr2 and fr4 are of type.edge and fr3 is of type.level. The effect of the field level can also be seen in the same figure, fr2 has pos_hi level, fr1 and fr3 have neg_lo level, and fr4 has level configured as both. See also 25.41.

Further, when using a type.level-type frame signal, the field rw_frm_cfg.out_on selects the frame event source that activates the frame output, and rw_frm_cfg.out_off selects which signal that turns off the frame signal.

The timing of when the frame output signal appears relative to the data can also be modified, as is described in the next section.

23.4.2.3 Frame cycle timing

Different protocols demand different timing for the frame signal relative to the data. In order to accomplish this, the two mode register fields rw_frm_cfg.tr_delay and rw_frm_cfg.rec_delay exist. They set the distance in bit clock cycles between frame signal activity and actual data input/output. They are valid for all three frame event sources described above, but the exact meaning of the values differs slightly in the three cases.

23.4.2.3.1 Isochronous mode with frame output signal

Figure 23.5 shows data output and data input sampling points, together with an edge-type frame signal for varying values of tr_delay/rec_delay (0-3). The behavior is analogous for delay values up to 7.

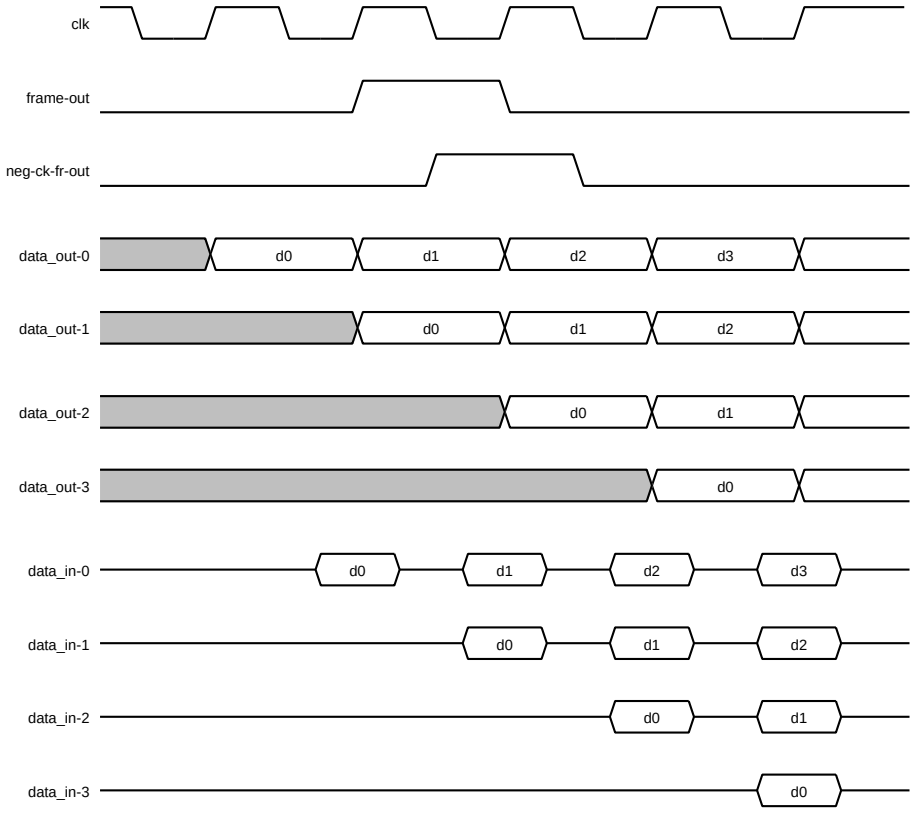


Figure 23.5: *Effect of tr_delay and rec_delay on frame output*

The signal neg-ck-fr-out shows the resulting position of the frame signal if different edge clocking is used for the frame logic compared to the clocking for the transmitter and receiver. The behavior is symmetric if negative edge clocking is used for the transmitter, receiver and/or frame.

Gated output clock This case is quite similar to the above. Only note that the first

clock pulse that is visible externally is the one associated with the first data bit (for receiver or transmitter, depending on rw_cfg.clkgate_ctrl).

External gated clock clocks the frame circuit The only rw_frm_cfg.tr_delay/rec_delay value that is useful for frame output in this case is 0, other values results in that more clock pulses are needed for each data word than there are data bits.

23.4.2.3.2 Frame input signal

Figure 23.6 shows similar examples in the case of frame input:

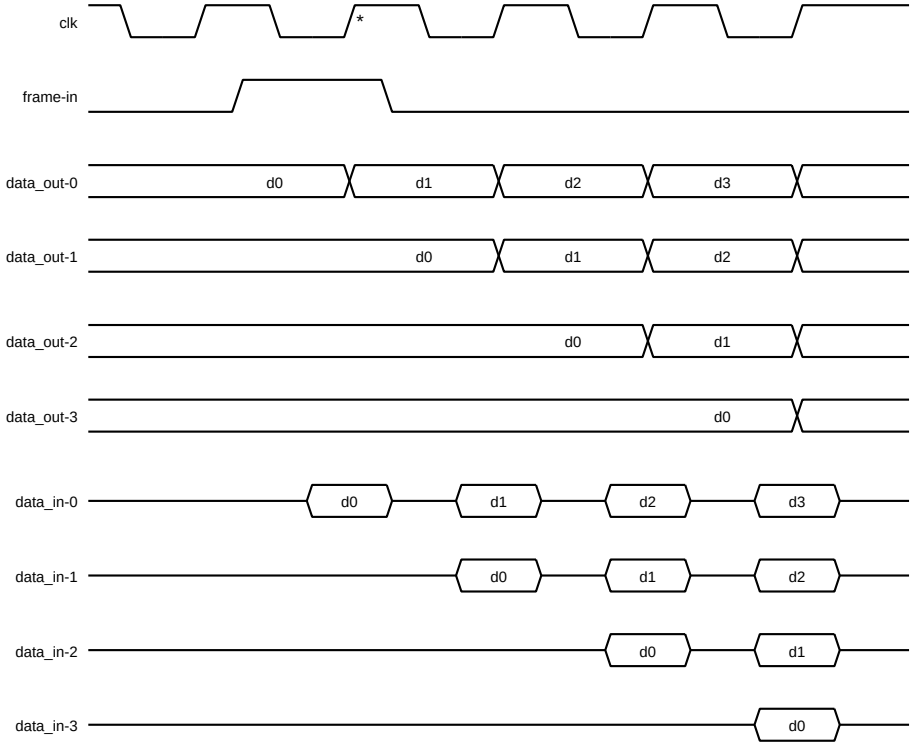


Figure 23.6: *Effect of tr_delay and rec_delay on frame input*

The external frame signal input is sampled on the clock edge marked by an '*'. If both transmitter/receiver and frame circuits use negative edge clocking, the figure is still valid except that the clk signal is inverted, i.e., the edge marked with '*' is then a negative edge. When the clock polarity of the frame circuit is different from the transmitter and/or receiver clock polarity, the following applies:

- The frame signal is sampled on the clock edge drawn as a negative edge just before the edge marked by '*' in the figure.
- To achieve the data in and out timing in figure 23.6, add 1 to the tr_delay/ rec_delay values. E.g. to get the 'data_out-1' behavior, write the value 2 to rw_frm_cfg.tr_delay.

- A tr_delay/rec_delay value of 0 is not allowed.

The first data bit propagates to the output within 2-4 transmitter clock cycles after that reset is released.

Gated output clock When using output clock gating together with an incoming frame signal, a tr_delay/rec_delay value of 0, or different clock phases for frame and the unit controlling clock gating (transmitter or receiver), are not allowed. The first clock edge visible outside the SSI is the one marked by '*'.

An external gated clock clocks the frame circuit With some restrictions a frame input signal can be used together with an external gated clock. To be recognized, the frame signal must be valid one setup time before the edge of the clock. When using normal data timing (see section 23.6.3.1), rw_frm_cfg.tr_delay must always be set to 1. When using early data, it must be set to 0. The field rw_frm_cfg.rec_delay must always be set to 0.

23.4.2.3.3 Transmitter bulk mode and frame output

When the transmitter is running in bulk mode, rw_frm_cfg.tr_delay specifies the distance between frame and data in the same way as in isochronous mode. However if the receiver uses the transmitter as a source of frame events, the field rw_frm_cfg.rec_delay is a little different. It now specifies the delay between transmitter start and receiver start (as opposed to frame-to-receiver in other modes).

Additionally, the mode register field rw_tr_cfg.bulk_wspace also affects the SSI operation in bulk mode. The field specifies the minimum distance between two transmitted words, in bit clock cycles. 0 gives no space between words. The space between words must be at least as many cycles as the maximum of fields rw_frm_cfg.tr_delay and rw_frm_cfg.rec_delay for the frame output and/or the receiver to work as expected (that is, if the frame output and receiver uses the transmitter as source of frame events).

As can be seen in figure 23.7, the effect of rec_delay = 0 (data_in-0 in the figure) is different when tr_delay is 0 and 1, relative to the frame signal (since rec_delay now specifies distance from transmitter start to receiver start). For each case of tr_delay, the sampling point with rec_delay = 0 is shown for same clock phase (data_in-0) and different clock phase (neg-ck-data_in-0) between transmitter and receiver.

Also note that a frame signal clocked on the opposite edge compared to the transmitter now ends up half a cycle before a normal frame signal (as opposed to when isochronous mode is used).

Clock gating For gated clock output, the first clock pulse that is visible externally is the one associated with the first data bit (for receiver or transmitter, depending on rw_cfg.clkgate_ctrl). External, gated clocking is not supported in bulk mode.

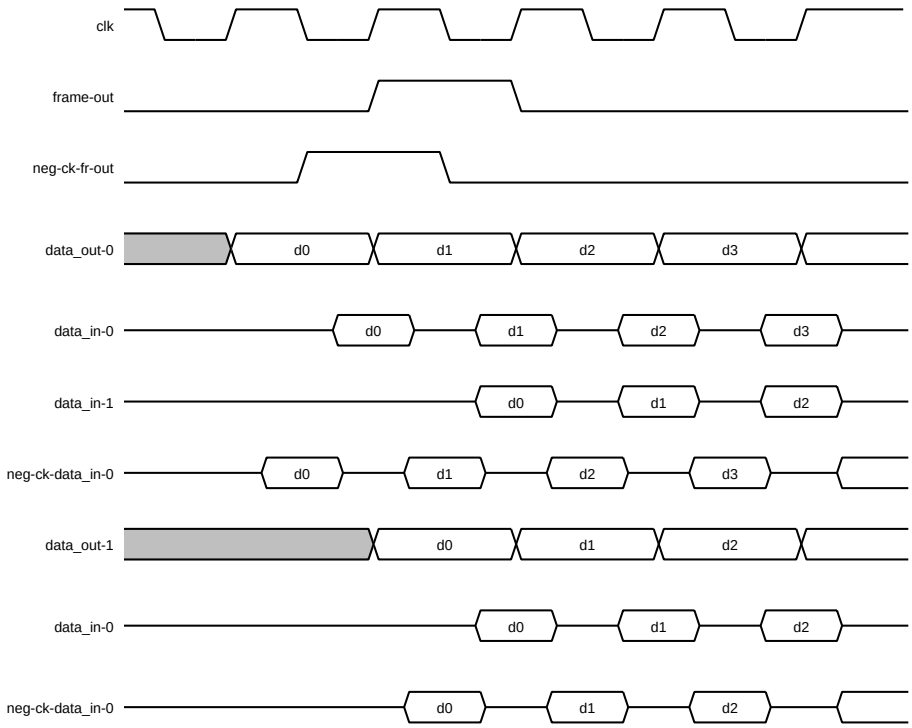


Figure 23.7: *Effect of tr_delay and rec_delay in bulk mode*

23.4.2.4 Special cases

23.4.2.4.1 Simultaneous master and slave

To a limited degree, the SSI can act as a master and a slave at the same time, using two frame signals (e.g. one frame output for the transmitter and a separate frame input for the receiver). However, some options regarding the frame signal type are common for both frame signals in this case, limiting the flexibility.

To support the case with completely independent clock, data and frame signals for receiver and transmitter, the mode register field rw_frm_cfg.fr_in_rxclk exists. When this field is unset, the frame clock is used to decode the incoming frame signal, but setting the mentioned register field instead uses the receiver clock to decode the incoming frame signal. In this way the frame clock can be set to something suitable for producing an outgoing frame signal for the transmitter instead.

23.4.2.4.2 No frame signal

There are three cases in which no physical frame signal is necessary (but can be used anyway, if desired):

gated clock When clock gating is employed, no frame signal is needed since the avail-

ability of a clock edge itself indicates the presence of data.

wiresave mode This mode has been designed specifically to avoid frame signals. See section 23.4.1.3.

IEC60958 mode In this mode special preamble sequences are used to indicate word start, so no dedicated frame signals are needed. See section 23.4.1.4.

23.4.2.4.3 Frame signals in 'highspeed' and 'wiresave' modes

Frame signalling in these modes has not been thoroughly tested and is therefore not supported beyond what has been specified in examples in this document.

23.4.3 Flow control

Flow control can be used to avoid buffer overruns. For the receiver, flow control is most useful in DMA mode. When data is read/written via mode registers, simple flow control is possible.

The use of flow control in conjunction with isochronous transmitter mode is not possible. The rest of this section applies to bulk mode.

23.4.3.1 Highspeed and lowspeed modes

In highspeed and lowspeed modes, flow control is enabled by setting either rw_frm_cfg.frame_pin_use or rw_frm_cfg.status_pin_use to hold. If the corresponding of fields rw_frm_cfg.frame_pin_dir or rw_frm_cfg.status_pin_dir is set to in, the hold signal will be an input and affect the SSI transmitter. Otherwise it will be an output, indicating the status of the SSI receiver. Flow control in both directions is possible by using both the **status** and the **frame** signals.

The polarity of both input and output hold signals is controlled by the field rw_cfg.hold_pol. Positive polarity means that a high level stops the transmitter and indicates an almost full FIFO in the receiver.

hold input The hold input signal is synchronized using the transmitter clock before being used by the SSI. Therefore the hold signal need not be synchronous to any particular clock.

The transmitter finishes the current word before reacting to the hold signal. Due to the synchronization mentioned, a new word may start up to two transmitter clock cycles after the hold signal has been issued.

If the transmitter is using an incoming gated external clock, the synchronization also causes one more word to be sent the next time the clock goes active, if the hold signal is activated while the clock is not running. This happens regardless of how much time passes since the hold signal is asserted.

hold output At which time the hold output signal goes active in DMA mode depends on the amount of data in the DMA FIFO, and on the mode register field

rw_rec_cfg.fifo_thr. In mode register mode, the hold output is high whenever there is a data word in the SSI that the CPU hasn't read yet (except if rw_rec_cfg.fifo_thr == inf, in which case no flow control is used). To have more relevant timing, the hold output signal is internally re-synchronized to the frame clock before being output. This also results in a delay of about two clock cycles of the frame clock before the output changes state. For most protocols this delay is insignificant.

In mode register mode, one full 16-bit word more can be received without causing overrun after the hold signal has gone active internally. If data is transferred back-to-back, the last word that can be accepted might have started already when the hold signal is visible externally. When calculating the threshold value to use in DMA mode, and the corresponding maximum allowed response time for an external transmitter to avoid overrun, consider the following:

- When using the slave receivers, data is written to the DMA FIFO in bursts of 2 or 3 bytes (for 2 or 3 receivers, word length <= 8 bits), or of 4 or 6 bytes (for 2 or 3 receivers, word length > 8 bits).
- When using a word length that is just above a multiple of 16 bits (e.g. 17, 18, 33 or 34 bits), the last part of the word will be written to the DMA FIFO shortly after (1, 2, ... bit times) the previous part.

If both the above cases apply at the same time, up to 6 + 6 bytes will be written to the FIFO with only 1-2 bit clock cycle times in between.

23.4.3.2 Flow control in wiresave mode

The hold pin approach described above can be used also in wiresave mode, but provided that both the transmitter and the receiver is running, xon/xoff handshaking can be used instead. This frees up pins for other purposes. For the transmitter to send xon/xoff metadata, rw_tr_cfg.use_md must be enabled. Regardless of this setting, the transmitter responds to received xon/xoff metadata codes.

The xon/xoff handshaking involves two concurrent processes:

1. As soon as the two first bits of a metadata word having the values 0b11 (xoff) are clocked in, transmission is disabled. No more than just below two serial words may then be transmitted (of which the last word might come immediately or some time later). Thereafter, only metadata is transmitted. When an incoming xon metadata word is detected, ordinary transmission is re-enabled.
2. When the receiver DMA FIFO is getting full, an xoff word is transmitted no later than one full serial word later. When there is more FIFO space again, an xon word is inserted into the transmitted data stream.

The xon/xoff words are inserted and removed transparently to software. All the programmer has to do is to make sure that the metadata to send does not contain the xon and xoff codes (all metadata with LSBs 0b11 are regarded as xoff). Higher level flow control can be managed by manually stopping and resuming the receiver DMA, xoff and xon will then be sent automatically.

external unit reaction time Since there are some delays associated with the xon/xoff signalling, the receiver DMA FIFO must have some margin left when xoff is to be signalled. This margin is configured using rw_rec_cfg.fifo_thr.

The recommended value for rw_rec_cfg.fifo_thr is thr16. Then the system will work with three parallel input data streams, provided that the external unit reacts at least as fast as this SSI when xoff is received (as is described above), and that the transmitter clock frequency is no more than 10% lower than the receiver clock frequency.

To get around difficulties with external unit reaction time, it is possible to increase the threshold to 32 bytes. However, since the DMA FIFO in ETRAX FS is 64 bytes and is not emptied until it holds 33 bytes, this leads to the disadvantage of potential transmission of lots of unnecessary xon/xoff commands.

23.4.4 Clocking

There are two fundamentally different clocking modes: internal and external clock. In the former, a clock reference is generated from the system clock, and in the latter a clock is input to the sync serial ports to clock data and frame in/out. The SSI is divided into three clock regions (receiver, transmitter and frame), for which internal/external clocking can be selected independently. Thus, the transmitter can use an internal clock (which can be output) at the same time as the receiver uses a different, external, clock applied to an input pin.

23.4.4.1 Internal clock

When using internal clocking, the clock frequency base is selected using rw_cfg.base_freq. The base frequency is then fed into a 16-bit clock divider, providing bit clock frequencies from 100 MHz down to around 450 Hz.

The base frequencies are:

- Off
- External serial/timer clock input, **ext_clk**
- 29.493088 MHz
- 32.000 MHz
- 32.768362 MHz
- 100.000 MHz

Note that using the **ext_clk** pin as a base for the internal clock is not the same as normal external clocking of the SSIs. When using the **ext_clk** pin as clock base, the **ext_clk** input is first sampled by the internal 100 MHz clock and then divided.

Some remarks on clock generation and internal clock output:

1. Setting the clock divider, field rw_cfg.clk_div to the value '0' generates a valid internal clock with the frequency selected by rw_cfg.base_freq, but if an off-chip output clock is wanted when rw_cfg.clk_div = 0, the following limitations apply:
   - If base frequency is 100 MHz, rw_cfg.out_clk_src has to be changed to clk100 instead of intern_clk.
   - If rw_cfg.base_freq is set to ext, the frequency of the external clock must be below 50 MHz.
   - The low time of the uninverted output clock (high time of inverted output clock) will always be 10 ns, resulting in a bad duty cycle. Does not apply if rw_cfg.out_clk_src is set to clk100.
2. In cases not covered by case 1 above, the duty cycle of an off-chip output clock will be as close to 50% as possible by using multiples of base_freq periods (each quantified to 10 ns accuracy) as high and low times. This means that the worst duty cycle with a 100 MHz base is 33%, when clk_div = 2.
3. When using the output clock gating option, positive edges of the gated output clock are delayed one base_freq cycle, worsening the duty cycle a little.

The following options can be selected using rw_cfg.out_clk_src, when the internal clock is output on the **clk** pin:

- An internally generated clock, as described above
- "nojitter", the internal clock, but output on the pos edge of **ext_clk**
- The internal 100 MHz clock (This special case must be selected whenever a 100 MHz output clock is wanted, even if the internal clock is set to 100 MHz)
- Constant value (specified by rw_cfg.out_clk_pol)

The nojitter option can provide an output clock with external reference, without the jitter normally imposed by synchronizing the external clock to the internal 100 MHz clock. This only works up to a maximum external clock frequency of 16.67 MHz though.

The polarity of the output clock can be selected using rw_cfg.out_clk_pol. By setting rw_cfg.clk_od_mode, the clock output driver is switched to open drain mode.

Clock gating for an internally generated output clock is enabled by setting rw_cfg.gate_clk. When gated, the output clock is only toggling when a data bit is being transferred. The idle value of a gated clock will be 0 with normal polarity and 1 with inverted polarity. When gating the output clock it must also be decided if the receiver block or the transmitter block shall control when the clock is to be turned on. This is controlled by rw_cfg.clkgate_ctrl.

The most cases the transmitter controls clock gating, receiver control is only used in the case of the SSI acting as an isochronous master-receiver. The unit controlling clock gating must be clocked by the internal clock. Clock gating is only allowed in lowspeed mode.

23.4.4.2 External clock

In external clock mode, a clock is input to an SSI pin and is used to directly clock data and frame signals in to and out from the SSI. Polarity of the external clock can be set independently for transmitter, receiver and frame.

The external clock can be input either from the **clk** pin or from the **ext_clk** pin. This is specified by rw_cfg.clk_in_sel. If the external clock is gated (only toggling when data bits are transferred), the rw_cfg.clkgate_in must be set. This only is allowed in lowspeed mode.

23.4.5 Reset behavior

The SSI block is enabled by rw_cfg.en. When not enabled, the entire SSI, including frame logic and clock logic, is reset. Further, the receiver and transmitter can be enabled separately by rw_rec_cfg.rec_en and rw_tr_cfg.tr_en, respectively. When not enabled, the transmitter and receiver are reset.

The contents of the mode registers is not affected by the reset activated by disabling the enable fields, only by system reset.

More detailed information on how to start up the SSI can be found in section 23.6.3.

23.4.6 Interrupts

The SSI has one interrupt vector output to the CPU, and a number of individually maskable internal interrupt sources. The SSI has the standard set of interrupt mode registers:

- rw_intr_mask
- rw_ack_intr
- r_intr
- r_masked_intr

For descriptions of these registers, see 25.41.

As soon as a non-masked interrupt source is activated, the interrupt to the CPU is signalled. The interrupt handler in software must then acknowledge the interrupt and take any other requested action. Each interrupt source and details about acknowledge etc. is listed below:

trdy (transmitter ready) The transmitter is ready to accept a new data word in mode register driven mode. See section 23.6.2.1 for details on acknowledging this interrupt. In DMA mode, this interrupt will not go active. The trdy interrupt goes active at around the same time as the transmission of the previous word starts.

rdav (receiver data available) A new data word has arrived from the receiver. See section 23.6.2 for details on acknowledging this interrupt in mode register mode. The rdav interrupt goes active slightly after that the last data bit has been clocked in[3].

In DMA mode, this interrupt will go active once for each data word written to the DMA. If the DMA is full, the interrupt comes when there is space again. If or when this interrupt is acknowledged has no consequence for data reception in DMA mode.

tidle (transmitter idle) This interrupt occurs when the transmitter has stopped and the last data has been fully transmitted. See section 23.6.3 on starting and stopping transmission for details.[4]

rstop (receiver stopped) The receiver has been stopped. See section 23.6.3 on starting and stopping transmissions for details.

urun (transmitter underrun error) Data was not supplied to the transmitter quickly enough. See section 23.6.4 on error handling.[5]

orun (receiver overrun error) Data was not read from the mode register quickly enough, or the DMA was full for too long, so data was lost. See also section 23.6.4 on error handling.

md_rec (metadata received) A metadata word was received. This can only happen in wiresave mode and the interrupt can be acknowledged whenever wanted, it is only a notification to software. See section 23.4.1.3.2 for more details.

md_sent (metadata sent) A metadata word is being sent. This happens in wiresave mode and other modes, whenever a DMA data descriptor with the out_eop bit set is being finished (or the rw_tr_data.md was written high, in mode register driven mode). Note that this interrupt comes before the metadata or last word of the data buffer associated with the descriptor has left the SSI.

If a more precise indication is needed of when the last data or metadata has been sent, the rw_tr_cfg.eop_stop field must be set. See section 23.6.3. Also see section 23.4.1.3.2 for more details on when the interrupt is generated in wiresave mode. This interrupt can be acknowledged whenever wanted, it is only a notification to software.

r958err (IEC60958 receiver error) An error occurred in the IEC60958 receiver. When using other than IEC60958 modes, this interrupt must be masked away since its value is then undefined. In IEC60958 mode it means either that an illegal data sequence was input to the SSI, or that a parity error occurred. There is no means to detect which of these cases that occurred. See section 23.4.1.4 on IEC60958.

[3]This does not necessarily mean that the entire SSI transaction is finished yet.

[4] When the transmitter is using an external gated clock, this interrupt will occur after reception of the last data-producing clock edge. The external unit might demand that the serial data line stays active until the next (opposite) clock edge.

[5]In bulk mode using DMA, and with serial word lengths of 128 bits (16 bytes) or shorter, the transmitter does not start until a full serial word is available from the DMA. Thereby underrun errors are prevented from happening. If the serial word length is longer than 128 bits, transmission starts as soon as 16 bytes are available and underrun errors can occur.

23.5 Hardware Interface

23.5.1 External pins

Each SSI has the external pins listed in table 23.5 below.

Pin name	Direction	Description
data	bidir	Data output, or extra data in
din	input	Data input
frame	bidir	Frame or hold in or out, or extra data in
status	bidir	Frame or hold or extra data in or out
clk	bidir	Main clock, in or out
ext_clk	input	Extra external clock input

Table 23.5: *SSI external pins*

These external pins are mapped onto physical pins as described in 16. Note that the **ext_clk** pin is shared between both SSIs, asynchronous serial ports and timers.

All the external pins can be independently used as general inputs/outputs. The values at the pins can always be read using the register r_rec_data.

data This pin is normally used for data output. It is configured using the field rw_tr_cfg.data_pin_use. If using non-wiresave modes and enabling the third slave receiver, the transmitter cannot be used and this pin is instead used as the third input.

din This is the primary pin used for data input. Its only other use is as a general input signal.

frame The primary use for this pin is as a frame input or output, but if desired the **status** pin can be used for this purpose instead. The **frame** pin is controlled by the fields rw_frm_cfg.frame_pin_use and rw_frm_cfg.frame_pin_dir. If not used for frame signalling, **frame** can be used for hold signalling (flow control), in or out. In wiresave mode, if using all three receivers, this signal is used as the third input.

status This pin is primarily intended for frame or hold (flow control) signalling. It is configured using the fields rw_frm_cfg.status_pin_use and rw_frm_cfg.status_pin_dir. If two I2S transmitters are used, this pin is used as the second I2S output instead. If using two or three receivers (in any mode), this pin is used as the second data input.

clk Normally this pin is used as a clock signal, in or out. It is configured using the fields rw_cfg.out_clk_src, rw_cfg.out_clk_pol, rw_cfg.clk_dir and rw_cfg.clk_in_sel. To use this pin as a general I/O, set rw_cfg.out_clk_src to const0, control its direction using rw_cfg.clk_dir and use the field rw_cfg.out_clk_pol to control its output value.

ext_clk This pin can only be used as an extra clock input, or as a general input. Set rw_cfg.clk_in_sel to ext_clk to select this pin as the clock input.

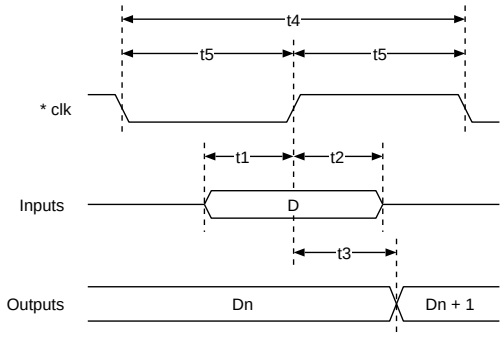
23.5.2 Reset behavior

The SSI mode registers will be initialized by power-on reset to values that result in that all SSI external pins are inputs.

The pin values will not be directly affected by the SSI enable fields, e.g. if some signal(s) are configured to function as general outputs, their values will be unaffected when the SSI is disabled. It is also possible to modify the value and/or direction of SSI signals while the port is disabled.

23.5.3 Timing

The following figures are valid with external output loads of 50 pF.



* clk is the clock on the external pin (**clk** or **ext_clk**), shown non-inverted.
If inverted, the same timing applies but related to the negative edge.

Figure 23.8: *External timing*

Nbr	Name	Description	min	max	unit	notes
t1	tsu	Input setup to clock	12.0	-	ns	1,3,4
t2	th	Input hold time from clock	-2.0	-	ns	1,3
t3	tod	Output delay from clock	-4.0	4.0	ns	3,5
t4	tecp	Clock period; highspeed, wiresave modes	10.0	-	ns	
		Clock period; lowspeeed mode	60.0	-	ns	
t5	tecw	Pulse width of clock; 100 MHz output	4.0	-	ns	
		Pulse width of clock; all other cases	t_nom-2.0	-	ns	2

Table 23.6: *Timing figures, internal clock output*

Nbr	Name	Description	min	max	unit	notes
t1	tsu	Input setup to clock	2	-	ns	1,4
t2	th	Input hold time from clock	2	-	ns	1
t3	tod	Output delay from clock	3.5	12.5	ns	5
t4	tecp	Clock period; highspeed, wiresave modes	9.1	-	ns	
		Clock period; lowspeeed mode	60.0	-	ns	

t5	tecw	Pulse width of clock	4.0	-	ns	-

Table 23.7: *Timing figures, external clock input*

Notes:

1. Timing is valid for input rise/fall times (clock and data) of max 2 ns.
2. t_nom is the nominal high/low time, which can be calculated as described in 23.4.4.1.
3. When using a gated output clock, the clock arrives one rw_cfg.base_freq period later on the external pin. tsu, th and tod will then be changed by this amount.
4. If using an external frame input signal, and a rw_frm_cfg.tr_delay or rw_frm_cfg.rec_delay of 0, an additional setup time of 2 ns is required for the frame signal.
5. Can optionally be adjusted for the **data** pin using rw_extra.dout_delay, see 25.41 for information on how timing is affected.

23.6 Software Interface

The SSI is controlled through a set of registers, see 25.41 for details. To access the registers, fields and register constants from a C program, a set of macros is defined in [MACROS].

The I/O pins of the SSIs are multiplexed with other functions. Before the output pins of the SSIs can be used, the pins must be configured, see 16.

Below is some information on data storage, transfer to/from the SSI, starting and stopping, error handling and metadata codes. Additionally, some examples can be found in chapter 23.7.

23.6.1 Data organization in memory

The SSI works with two data streams, one in each direction. The data formatting depends on the selected word size, how many receivers or transmitters that are enabled, and on the shift direction (MSB or LSB first). Below, the data formatting for the transmitter is described. The formatting for the receiver is identical, just substitute rw_tr_cfg with rw_rec_cfg.

In DMA mode, when the field rw_tr_cfg.sample_size is set to a serial word size of 8 bits or less, one byte is read from DMA for each serial word sent. When the serial word size is up to and including 16 bits, one 16-bit memory word per serial word is read instead.

In mode register mode, with a serial word size up to and including 16 bits, only the rw_tr_cfg.sample_size least significant bits of the mode register are used.

When the serial word size is bigger than 16 bits, two or more 16-bit memory words or mode register data words are used for each serial word sent. The number of 16-bit words needed is simply the serial word length divided by 16, rounded up.

Any padding is always applied in the MSB end of each serial word. E.g., if the serial word length is 36 bits, three 16-bit memory words will be used for each serial word, bits 0 to 35 will be data and bits 36 to 47 will be padding.

If the data is transferred LSB first, the least significant 16-bit memory word shall be sent to the SSI first (i.e., be stored at the lowest memory address if using DMA, or be written first to the mode register in mode register mode). Conversely, if transferring MSB first, the most significant 16-bit word shall be sent to the SSI first. Any padding is in the most significant bits of the first 16-bit word (lowest memory address) in this case.

When more than one transmitter is used, data for the different transmitters is interleaved byte-by-byte (wordsize <=8 bits) or 16-bit-word-by-16-bit-word (otherwise). Data for transmitter 1 is first, followed by data for transmitter 0. The order is reversed for the receiver, here receiver 0 data is first, followed by receiver 1, then receiver 2 (if used).

23.6.1.1 Examples

Consider the case of serial word length = 18 bits, MSB first, three receivers and two transmitters. Reception would then render the following data in memory (DMA mode) or the following sequence of words read from the mode register (mode register mode) as shown in table 23.8.

DMA mem offset	Mode reg read	Contents
0x00	1st, low byte	receiver 0 word 0 bits 17-16 [6]
0x01	1st, high byte	receiver 0 word 0 padding
0x02	2nd, low byte	receiver 1 word 0 bits 17-16 [6]
0x03	2nd, high byte	receiver 1 word 0 padding
0x04	3rd, low byte	receiver 2 word 0 bits 17-16 [6]
0x05	3rd, high byte	receiver 2 word 0 padding
0x06	4th, low byte	receiver 0 word 0 bits 7-0
0x07	4th, high byte	receiver 0 word 0 bits 15-8
0x08	5th, low byte	receiver 1 word 0 bits 7-0
0x09	5th, high byte	receiver 1 word 0 bits 15-8
0x0a	6th, low byte	receiver 2 word 0 bits 7-0
0x0b	6th, high byte	receiver 2 word 0 bits 15-8
0x0c	7th, low byte	receiver 0 word 1 bits 17-16 [6]
0x0d	7th, high byte	receiver 0 word 1 padding

Table 23.8: *Receiver data organization example*

The transmitter would need the following data shown in table 23.9.

DMA mem offset	Mode reg wr	Contents
0x00	1st, low byte	transmitter 1 word 0 bits 17-16 [6]

[6] Bits 17-16 of this word are stored in bits 1-0 of the byte. Bits 7-2 are padding.

0x01	1st, high byte	transmitter 1 word 0 padding
0x02	2nd, low byte	transmitter 0 word 0 bits 17-16 [6]
0x03	2nd, high byte	transmitter 0 word 0 padding
0x04	3rd, low byte	transmitter 1 word 0 bits 7-0
0x05	3rd, high byte	transmitter 1 word 0 bits 15-8
0x06	4th, low byte	transmitter 0 word 0 bits 7-0
0x07	4th, high byte	transmitter 0 word 0 bits 15-8

Table 23.9: *Transmitter data organization example*

The out_eop field in a DMA data descriptor must not be set unless the data of the descriptor ends at a serial word boundary.

23.6.2 Transferring data

This section describes how data is communicated to/from the SSI using DMA or mode register communication. Which to use is selected with the fields rw_tr_cfg.use_dma and rw_rec_cfg.use_dma.

23.6.2.1 Mode register driven mode

When the SSI is controlled directly by the CPU, not using DMA, data to send is written to the register rw_tr_data. The transmitter is ready to accept data when r_intr.trdy has gone high. A write to rw_tr_data will be detected by the SSI and the data will be sent, but first it is necessary to write rw_ack_intr.trdy high, before writing the data, to clear the r_intr.trdy field for the next transfer.

Note that the transmitter expects the first data word to exist in the rw_tr_data mode register at the moment when the transmitter is enabled, even though no r_intr.trdy interrupt is issued at this time.

Received data is read from the register r_rec_data. Data is available when r_intr.rdav is high. When the data has been read, write rw_ack_intr.rdav high to acknowledge this.

If sample size is less than or equal to 16 bits, one write or read is required for each serial word transfer. When the sample size is bigger than 16 bits, more than one read or write is required for each received or transmitted word. A new interrupt has to be awaited before each additional read or write.

23.6.2.1.1 Allowed interrupt latency

At least one 16-bit word of data is buffered inside the SSI for each transmitter and receiver (if serial word size is <= 16 bits, one serial word is buffered). Therefore the worst-case maximum allowed latency from that the trdy or rdav interrupt is issued until data must have been written or read is around 16 cycles of the serial clock (or one serial word length, for shorter serial word lengths). The maximum allowed latency might of course be longer depending on the configuration, e.g. if the word rate is lower, or if

using a bulk mode transmitter with serial word length <= 16 bits (giving no constraint on latency).

When using more than one receiver, data words will appear in pairs or triplets (2 or 3 receivers). The words in each pair or triplet will appear very quickly after each other (after two 100 MHz clock cycles). Therefore, some interrupt routine overhead can be saved by making the rdav interrupt routine check if the interrupt has been retriggered directly after acknowledging the previous interrupt.

Similarly, the transmitter will accept two words quickly if two transmitters are used. Therefore the same overhead saving technique can be used with the trdy interrupt.

23.6.2.2 DMA mode

To start DMA data transfer, first set up the DMA, including data width of 8 or 16 bits, and enable it before the SSI is enabled.[7]

When the transmitter DMA hits an out_eop, and the field rw_tr_cfg.eop_stop is set, transmission will stop after sending the descriptor's associated data. Then the tidle interrupt will be issued. Further, see section 23.6.3.2 below on stopping.

23.6.3 Starting and stopping

23.6.3.1 Enable procedure

The following is a description of how to enable the SSI in a correct way during different clocking conditions. After going through these procedures, the SSI communication will be running.

23.6.3.1.1 Continuous clock or internal clock

When using continuous clocks and/or internal clock, the enable sequence is as follows:

1. Configure the SSI mode registers, keeping rw_cfg.en cleared.
2. Enable DMA or write first transmit data to mode register.
3. Wait until at least three cycles of the selected bit clock type has passed.
4. Set the rw_cfg.en field.

Communication will then start as soon as a frame event occurs.[8]

[7] The SSI transmitter expects the first data to be ready at its input when enabled, therefore it should be checked that the DMA has started filling its FIFO before enabling the SSI. This can be done using the rw_stat.buf field of the connected DMA channel.

[8] It takes 2-3 bit clock cycles for reset to propagate inside the SSI. Therefore an external frame input will not be monitored just after reset.

23.6.3.1.2 Gated external clock

When using a gated external clock, there are two different reset sequences. Which one to use depends on if 'early' or 'normal' data is wanted from the transmitter:

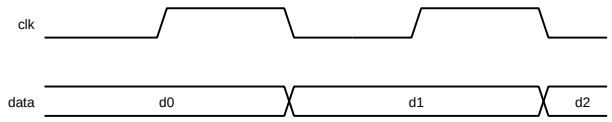


Figure 23.9: *Early data*

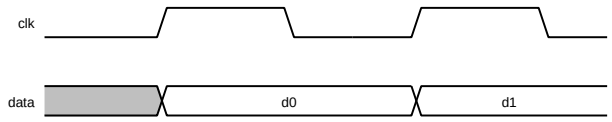


Figure 23.10: *Normal data*

In figure 23.9, the transmitter uses negative clock edges. In figure 23.10, it uses positive edges. The opposite use of clock edges is possible in both cases but is less useful

To get early data, reset as follows:

1. Configure the SSI mode registers, keeping rw_cfg.en cleared, and set the rw_cfg.prepare field.
2. Enable DMA or write first transmit data to mode register.
3. Select internal clock.
4. Wait at least (rw_cfg.clk_div+1)*30 ns
5. Set the rw_cfg.en field.
6. Wait at least (rw_cfg.clk_div+2)*30 ns.
7. Select external clock.
8. clear rw_cfg.prepare
9. Wait at least 30 ns before starting the external clock.

Now the first transmit data bit will be available on the SSI data output, and the first active clock edge detected will clock out the second data bit.

To get normal data, reset as follows:

1. Configure the SSI mode registers, keeping rw_cfg.en cleared.
2. Enable DMA or write first transmit data to mode register.

3. Select internal clock.
4. Wait at least (rw_cfg.clk_div+2)*30 ns
5. Select external clock.
6. Set the rw_cfg.en field.
7. Wait at least 30 ns before starting the external clock.

Now the SSI is ready to receive the first active clock edge. After this edge it will present the first data bit on the SSI data output.

Early or normal data modes are available also with the internal gated clock, but in that case it is a function of the transmitter clock polarity only, the reset sequence is not affected. Also, the early data is in that case generated by the ”invisible” negative edge just before the first visible clock cycle according to figure 23.9 above.

23.6.3.2 Stopping the SSI

There are several options available for stopping the SSI. The most brutal way is to just clear the rw_cfg.en, which will stop transmission immediately. Data might then be lost and the serial protocol may be violated.

There are other ways when a controlled stop is wanted. Both the transmitter and the receiver have some mode register fields for this:

- rw_tr_cfg.eop_stop
- rw_tr_cfg.stop
- rw_rec_cfg.eop_stop
- rw_rec_cfg.stop
- rw_rec_cfg.force_eop

All these fields can be set or cleared at any time.

transmitter, eop_stop When the rw_tr_cfg.eop_stop field is set, transmission will stop after the last data or metadata associated with a descriptor with out_eop set has been transmitted (except if the metadata of the descriptor is ’tx_null’, in which case the current out_eop will be ignored and the transmission will continue. See section 23.4.1.3.2). When the data has been fully transmitted, the tidle interrupt is issued (see also section 23.4.6). At this point there is no data residing inside the transmitter, and the transmitter will not read any more data from DMA. The DMA may have continued reading data though, and stored it in its FIFO.

When the tidle interrupt is acknowledged, transmission will continue. The rw_tr_cfg.eop_stop might be kept set, in which case the transmitter will stop after the next out_eop too. If no more stops are wanted, the field shall be cleared before the interrupt is acknowledged.

Important note: At the same time as the tidle interrupt is acknowledged after an eop_stop (using rw_ack_intr.tidle), the rw_ack_intr.urun field must also be set, even if the urun interrupt hasn't been triggered. Otherwise a false urun interrupt might be issued directly after tidle acknowledge.

The rw_tr_cfg.eop_stop field has no effect in mode-register-driven mode and shall then be kept cleared.

transmitter, stop When the rw_tr_cfg.stop field is set, transmission will stop as soon as there is no more data inside the transmitter, and any word of which some part has been fetched from DMA or mode register has been completely sent. This means that more data might be required from DMA or mode register after writing the rw_tr_cfg.stop field, before the transmitter will stop. When stopped, the tidle interrupt is issued (see also section 23.4.6). At this time there is no data residing inside the transmitter, and the transmitter will not read any more data from DMA. The DMA may have continued reading data though, and stored it in its FIFO.

The recommended way of continuing after stop is to clear the rw_tr_cfg.tr_en (or rw_cfg.en) field, then clear the rw_tr_cfg.stop field, and then restart as described in section 23.6.3.1.

receiver, eop_stop When the rw_rec_cfg.eop_stop field is set, reception will stop after either of the following conditions:

- Metadata is received in wiresave mode, or
- The rw_rec_cfg.force_eop field is set (see below).

When reception has stopped and any data received before the stop point has been transferred to DMA or mode register, the rstop interrupt is issued. When this interrupt is acknowledged, reception will continue. While stopped, any incoming data will be discarded.

receiver, stop When the rw_rec_cfg.stop field is set, reception will stop as soon as possible without losing data before the stop point. For details see the section on this field in 25.41. When reception has been stopped, the rstop interrupt is issued. The recommended way to deal with rstop generated by writing the rw_rec_cfg.stop field is to clear the rw_rec_cfg.rec_en field (or the rw_cfg.en field), and then acknowledge the interrupt. The receiver must not be enabled again until the rstop interrupt has been acknowledged.

receiver, force_eop When the rw_rec_cfg.force_eop field is set, a metadata code will be inserted into the receiver (see the table in section 23.6.5). This results in that after that any pending data has been written to DMA or mode register, an eop will be signalled to DMA along with the metadata code, or that the metadata code will be written to the read data mode register (along with the r_rec_data.md field) as soon as any pending data has been read. See also 25.41, the rw_rec_cfg.force_eop field.[9]

[9] If using the rw_rec_cfg.force_eop while the field rw_rec_cfg.eop_stop is set, the receiver must be disabled when the rstop interrupt is received, before it is acknowledged, to prevent multiple eop cycles to DMA.

23.6.4 Error conditions and recovery

The SSI can signal three different error conditions, described below:

1. IEC60958 error: An error occurred during IEC60958 reception. See sections 23.4.6 on the r958err interrupt and section 23.4.1.4 on IEC60958 (and especially 23.4.1.4.2 on rate detection).

2. transmitter underrun: The CPU or the DMA could not supply data quickly enough for the transmitter, so data wasn't available in time for transmission. For obvious reasons this can happen in isochronous modes (whether internal or external frame is used). Further it can also happen in bulk mode if the serial word length is greater than 16 bits and CPU-controlled (non-DMA) mode is used, since the SSI can then not know beforehand if the CPU will be able to write more than the first part of the word in time (in DMA-controlled bulk mode transmission will not start until the DMA holds one complete serial word).

   When an underrun condition arises the urun interrupt is signalled and at the same time, or shortly thereafter, garbage data is being transmitted.

   At this point, two alternative behaviors are available, controlled by the rw_tr_cfg.urun_stop field. If urun_stop.no was selected, transmission of real data will continue as soon as possible when new data is available. If urun_stop.yes was selected, no more data will be read from DMA or mode register until the urun interrupt is acknowledged. It is recommended that the transmitter is disabled (=reset) before acknowledging this interrupt when the rw_tr_cfg.urun_stop field is active.

   If isochronous modes are used, frame signals will still be generated, clock gating will be activated etc, regardless of the rw_tr_cfg.urun_stop setting. Its only effect is to defer the transfer of more data from DMA or mode register.

3. receiver overrun: Similarly, if the CPU or DMA doesn't remove data quickly enough from the receiver, overrun occurs. This error can occur in any mode, except if flow control is used and is obeyed by the external unit (see section 23.4.3).

   When an overrun condition arises, the orun interrupt is signalled at the same time as received data is lost. The data word that is lost was the one last received, any data buffered inside the SSI is from the time before the orun interrupt.

   Also for overrun there are two alternative behaviors, controlled by the rw_rec_cfg.orun_stop field. If orun_stop.no was selected, reception continues as soon as the DMA or CPU removes data from the SSI again. For orun_stop.yes, reception will not continue until the orun interrupt is acknowledged (although data buffered inside the SSI from before the time when orun was signalled can be read by the CPU or will be written to DMA when non-busy regardless of orun interrupt acknowledge). It is recommended that the receiver is disabled (=reset) after that the last data has been taken care of and before the orun interrupt is acknowledged when the rw_rec_cfg.orun_stop field is set.

   If isochronous modes are used, frame signals will still be generated, clock gating will be activated etc, regardless of the rw_rec_cfg.orun_stop setting. Its only effect is to defer transfer of received data to DMA or mode register.

23.6.5 Wiresave mode metadata codes

In wiresave mode, ”metadata” can be inserted into the stream of ordinary data to mark special events, or points of interest, in the data stream.

The following table lists all possible metadata codes. The xon/xoff codes are only meant to be used by the automatic xon/xoff logic in the SSI, and should not be used as ordinary transmitter metadata. The manual_eop codes should preferably not be sent either, as this might confuse the receiver since it can then not tell for sure if the manual_eop codes come from real manual_eop events or from metadata sent by the transmitter. The other codes may be used depending on the application.

value	sent	md_sent	stored	md_rec	name
all with bit0=1 & bit1=1	yes	no	no	no	xoff
0x0005	yes	no	no	no	xon
0x6001	yes	yes	yes	no	manual_eop [10]
0xa001	yes	yes	yes	no	manual_eop [10]
0xe001	yes	yes	yes	no	manual_eop [10]
0x0009	yes	yes	no	yes	irq
0x0001	no	no	yes	yes	tx_null
0x000d	yes	yes	yes	no	rx_null
all with bit0=0	yes	yes	yes	yes	normal metadata

Table 23.10: *Metadata codes and their effect*

The ’sent’ column is ’yes’ if the metadata code can be sent by the transmitter.

The ’md_sent’ column is ’yes’ if sending that code produces an md_sent interrupt from the transmitter.

The ’stored’ column is ’yes’ for metadata which the receiver stores in the DMA descriptor (storing metadata also includes DMA eop signalling, writing out DMA FIFO contents to memory etc).

The ’md_rec’ column is ’yes’ for codes which causes an md_rec interrupt from the receiver.

All metadata codes with bit 0 = 0 are free to use for general purposes. The unused codes with bit 0 set are reserved for future use and should not be transmitted unless specifically required by an external device. The tx_null metadata is silently discarded by the transmitter and can be used if no metadata sending is wanted even if the ’out_eop’ bit is set in a descriptor. Even if rw_tr_cfg.eop_stop is set, transmission will not stop in case the metadata was tx_null. The rx_null metadata is stored, but no interrupt is signalled from the receiver.

[10] See 25.41, field rw_rec_cfg.stop.

23.7 Configuration examples

23.7.1 I2S

The [I2S] datasheet describes I2S communication in general. Further information (such as clock rate etc.) is to be found in the datasheet of the actual device to be connected.

In this example configuration procedure, the SSI is configured as a master with one output and one input channel, 12.5 MHz bitrate, 16 bits per sample, DMA mode and internal clock output:

1. Set rw_cfg.clk_div to 7 and set rw_cfg.base_freq.f100.
2. Disable all clock gating.
3. Set the clock pin to be an output with rw_cfg.clk_dir.out.
4. Set output clock polarity to rw_cfg.out_clk_pol.pos, output clock source to rw_cfg.out_clk_src.intern_clk.
5. rw_tr_cfg.clk_src, rw_rec_cfg.clk_src and rw_frm_cfg.clk_src to intern
6. Turn off open-drain mode for the **clk** pin.
7. Set rw_tr_cfg.clk_pol to neg, rw_rec_cfg.clk_pol to pos and rw_frm_cfg.clk_pol to neg
8. Use either the **frame** or the **status** pin as a frame signal output. Set the other one to general IO mode.
9. Set rw_frm_cfg.out_on.intern_tb and rw_frm_cfg.out_off.rec
10. Set rw_frm_cfg.type.edge and rw_frm_cfg.level.both.
11. Set rw_frm_cfg.tr_delay and rw_frm_cfg.rec_delay to 2.
12. Set rw_frm_cfg.wordrate to 15.
13. Set the **data** pin to be a normal output by setting rw_tr_cfg.data_pin_use to dout.
14. Turn off open-drain mode for the **data** pin.
15. Set rw_tr_cfg.dual_i2s to no, and set rw_rec_cfg.slave3_en and rw_rec_cfg.slave3_en to no.
16. Set rw_tr_cfg.use_md to no.
17. Set rw_tr_cfg.rate_ctrl to iso,
18. rw_tr_cfg.frm_src and rw_rec_cfg.frm_src to intern
19. Set rw_tr_cfg.mode and rw_rec_cfg.mode to lospeed.
20. Set rw_tr_cfg.use_dma and rw_rec_cfg.use_dma to yes.
21. Set rw_tr_cfg.sh_dir and rw_rec_cfg.sh_dir to msbfirst.
22. Set rw_tr_cfg.sample_size and rw_rec_cfg.sample_size to 15 (16 bits per sample).

Then start communication the usual way, as described in section 23.6.3.

23.7.2 SPI

The following is an example setup procedure for an isochronous SPI master, with both CPHA and CPOL equalling zero:

1. Enable output clock gating by rw_cfg.gate_clk.yes, rw_cfg.clkgate_in.no and rw_cfg.clkgate_ctrl.tr.
2. Set output clock polarity to rw_cfg.out_clk_pol.pos, output clock source to rw_cfg.out_clk_src.intern_clk.
3. rw_tr_cfg.clk_src, rw_rec_cfg.clk_src and rw_frm_cfg.clk_src to intern
4. rw_tr_cfg.clk_pol to neg, rw_rec_cfg.clk_pol to pos and rw_frm_cfg.clk_pol to neg,
5. Use either the **frame** or the **status** pin as a frame signal output. Set the other one to general IO mode.
6. Set rw_frm_cfg.out_on.intern_tb and rw_frm_cfg.out_off.rec,
7. rw_frm_cfg.type.level and rw_frm_cfg.level.neg_lo.
8. Set rw_frm_cfg.tr_delay and rw_frm_cfg.rec_delay to 1.[11]
9. Set rw_tr_cfg.sample_size and rw_rec_cfg.sample_size to 7 (8 bits per word).
10. Set rw_frm_cfg.wordrate to your desired word rate, 7 for back-to-back transmission.
11. Set rw_tr_cfg.rate_ctrl to iso,
12. rw_tr_cfg.frm_src and rw_rec_cfg.frm_src to intern,
13. rw_tr_cfg.mode and rw_rec_cfg.mode to lospeed,
14. rw_tr_cfg.sh_dir and rw_rec_cfg.sh_dir to msbfirst.

Then start communication the usual way, as described in section 23.6.3.

23.7.3 MAX1202

It is possible to use many different timing sequences when communicating with the MAX1202, according to its data sheet. The SSI has been verified using ”external clock mode, 15 clocks/conversion timing”, according to figure 11a in [MAX1202]. If this timing were to be used straight away, it has the awkward side effect of splitting each data sample across two 16-bit memory words (bits 11 to 5 in one word and bits 4 to 0 in the next, together with bits 11 to 5 of the next sample, etc.).

If the communication is to run continuously once it has been started, the splitting of data samples can be avoided by manually generating the first few clock cycles of the first communicated word, using the CPU and the general I/O features of the SSI, before starting the normal SSI communication. Thereby the SSI is fooled into believing that each word starts in the middle of the real word, resulting in that all 12 bits of each sample is stored in the same 16-bit memory location. The following procedure is a description of how to do this.

[11] rw_frm_cfg.tr_delay and rw_frm_cfg.rec_delay values of 1 render frame in the same cycle as data (0 gives data before frame), see section 23.4.2.3.1.

23.7.3.1 Initial configuration

1. Don't use clock gating
2. Set rw_frm_cfg.tr_delay to 2 and rw_frm_cfg.rec_delay to 1.
3. Set rw_frm_cfg.wordrate, rw_tr_cfg.sample_size and rw_rec_cfg.sample_size all to the value 14 (i.e., 15 bits/word, back-to-back transmission).
4. Set rw_tr_cfg.rate_ctrl to iso,
5. rw_tr_cfg.frm_src and rw_rec_cfg.frm_src to intern,
6. rw_tr_cfg.mode and rw_rec_cfg.mode to lospeed,
7. rw_tr_cfg.clk_src, rw_rec_cfg.clk_src and rw_frm_cfg.clk_src to intern
8. rw_tr_cfg.clk_pol to neg, rw_rec_cfg.clk_pol to pos and rw_frm_cfg.clk_pol to pos,
9. rw_tr_cfg.sh_dir and rw_rec_cfg.sh_dir to msbfirst.
10. Set output clock polarity to positive, output clock source to constant 0.
11. Set the **frame** pin to be a general output with value 1.
12. Set the **data** pin to be a general output with value 0.

Now the MAX1202 can be connected/enabled.

23.7.3.2 Starting communication

1. Set the **frame** pin low.
2. Wait tCSS
3. Set the **clk** pin high
4. Wait half a period
5. Set the **clk** pin low again
6. Wait half a period
7. Set rw_cfg.out_clk_src to intern_clk.
8. Enable the SSI, as described elsewhere in this document.

The communication will now continue by itself. There will be three pulses on the **clk** pin before data starts being clocked in to and out from the SSI, this leads to that the conversion result data ends up with bit 0 at bit 0 in memory.

Following this procedure, ”control byte 0” in figure 11a in [MAX1202] will be all zeros, including the start bit. Therefore, the first conversion result will probably be invalid. The data to be output from the SSI must be configured to contain proper start bits and control bytes for the MAX1202 to function.

Also, it might be possible to start the SSI as usual, without manual clock cycle generation, and still get well-aligned data by just putting the start bit at the correct position, see [MAX1202].

23.7.4 I2C

The following is a description of how to use the SSI for I2C communication.

23.7.4.1 Electrical connection

When using I2C communication, externally connect the **data** pin to the **din** pin, see figure 23.11. Also connect pull-up resistors on the **clk** and the **data** pins (for suitable values, see [I2C]).[12]

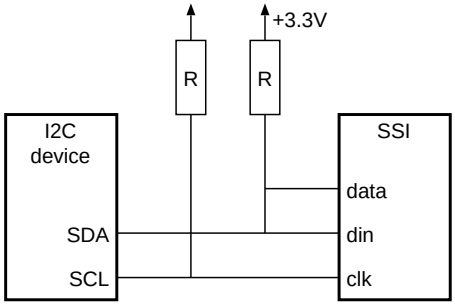


Figure 23.11: *Connection to an I2C device*

23.7.4.2 Data formatting

In order for I2C mode to work, data must be transmitted and received at the same time. Since the data output and input are connected, a copy of the transmitted data is also received simultaneously.

The serial word length used with I2C is 9 bits (8 bits data plus one ack bit). This means that the SSI consumes and produces 16 bits of memory data for each serial word in DMA mode, of which the 9 least significant bits are used. The LSB, bit 0, is the ack bit and bits 8 to 1 are the data.

When data is to be received, the SSI must not drive the **data** pin. Therefore, 1:s have to be stored in every transmitter data bit that is transmitted at the same time as data is to be received. For example, the ack bits that are to be received from the slave have to be set in the transmitter's data. Since the master always controls when data shall be transmitted or received, this is no problem.

23.7.4.3 Initial configuration

Set the register fields rw_tr_cfg.od_mode and rw_cfg.clk_od_mode, the latter only if pull-ups are used on both the **data** and the **clk** pins.

Some more settings:

[12] If it is known that the external I2C device never drives the clock line, the pull-up on **clk** can be omitted and a normal (non-open-drain) output can be used instead. This also means less susceptibility to noise.

- Set rw_tr_cfg.sample_size and rw_rec_cfg.sample_size to 8 (=9 bit data),
- Use internal clocking for both receiver and transmitter,
- Set rw_frm_cfg.tr_delay, rw_frm_cfg.rec_delay and rw_tr_cfg.bulk_wspace to 0,
- Set rw_tr_cfg.rate_ctrl to bulk, rw_tr_cfg.sh_dir and rw_rec_cfg.sh_dir to msbfirst,
- Set rw_rec_cfg.frm_src to tx_bulk.
- Set rw_cfg.gate_clk to yes, and rw_cfg.clkgate_ctrl to tr,
- Set rw_extra.dout_delay to 30 (which will work for both 100 kbit/s and 400 kbit/s communication).
- Set the rw_tr_cfg.tr_en and rw_rec_cfg.rec_en fields (but keep rw_cfg.en disabled).

23.7.4.4 Communication

Before communication is started, also do the following:

- Set rw_cfg.out_clk_src to const0 and rw_cfg.out_clk_pol to neg, thereby forcing the clock signal to a logical 1.
- Set rw_tr_cfg.data_pin_use to gio1, thereby forcing the data signal to a logical 1.

(It might be desirable to do the above two bullets along with the open drain selection, before the pinmux is configured for connecting the SSI to the ETRAX FS pins, to ensure that there are no glitches on the external pins from power-up until communication is started.)

Now communication can begin. Produce a start condition by:

1. Ensure that at least tBUF (see [I2C]) has passed since last communication session.
2. Set rw_tr_cfg.data_pin_use to gio0.
3. Wait tHD;STA
4. Set the clock signal low, by writing pos to rw_cfg.out_clk_pol.

Now enable the transmitter and receiver:

5. Write the first data word to the transmitter (or enable DMA)
6. Set rw_cfg.out_clk_src to intern_clk. Clock is gated so no clock will be produced until the SSI is enabled.
7. Set the rw_cfg.en field.

Now communication will start. The SSI must be stopped again before the next repeated-start or stop condition. In DMA mode, it is most convenient to set EOP of the last descriptor and set the rw_tr_cfg.eop_stop field and make sure the descriptor metadata is something else than 'tx_null' (see 23.4.1.3.2), then a tidle interrupt will be issued when finished.

In mode-register-driven communication, set the rw_tr_cfg.stop field after that the last data has been written to the rw_tr_data.data field, which will also trigger the tidle interrupt when finished.

8. Wait until the tidle interrupt appears
9. Since we now know that the receiver also got all its data, set the rw_rec_cfg.stop field. Now EOP is written to the in-DMA or mode register. Wait for the rstop interrupt (which should come quickly). In DMA mode, it might also be desirable to wait for a DMA in_eop interrupt.
10. Set rw_tr_cfg.data_pin_use to gio1 (for a repeated start condition) or gio0 (for a stop condition).
11. Wait tSU;DAT. This step can be skipped if the previous bullet didn't change the state of the data signal.
12. Clear rw_cfg.en, set rw_cfg.out_clk_src to const0, and rw_cfg.out_clk_pol to neg, all in one write operation, to disable the SSI and to set the clock signal high.
13. Acknowledge the rstop interrupt
14. Wait tSU;STO or tSU;STA
15. Set rw_tr_cfg.data_pin_use to gio0 (for a repeated start condition) or gio1 (for a stop condition).
16. If stop, we're finished now. Otherwise, produce a repeated start condition by going back to bullet 3 above.

An alternative configuration can be used if waiting for the above described delay times is a problem. The idea is to use an inverted output clock, stop the transmission *before* the last byte is transferred, switch to 10 bit serial word length (8 bits data, 1 ack bit, and the repeated start (=1) or stop (=0) condition), and then transfer the last word. When the transmission is finished, the data line is switched to 1 (stop) or 0 (repeated start) immediately. Then the only time the software has to wait is for tHD;STA at start or repeated-start conditions, but on the other hand the SSI has to be stopped and started once more per repeated-start or stop condition. This alternative configuration has not been tested.

23.7.5 Atmel flash memory (fast SPI)

This example shows how to connect and configure the SPI as a fast SPI master, to communicate with an Atmel flash memory (see [ATMEL]).

Both "SPI Mode 0 Compatible" and "SPI Mode 3 Compatible" as described by Atmel in the timing diagrams on page 17 of [ATMEL] are probably possible, but only mode

0 has been verified. Data will be input from and output to the slave during the entire communication cycle, even though the device only produces valid output data and cares about its input data during specific intervals.

23.7.5.1 Hardware connection

Connect the **frame** pin to the ”cs_n” input of the SPI slave, the **clk** pin to the ”sck” input, the **data** pin to ”si” of the slave and **din** to ”so” of the slave, see figure 23.12 below:

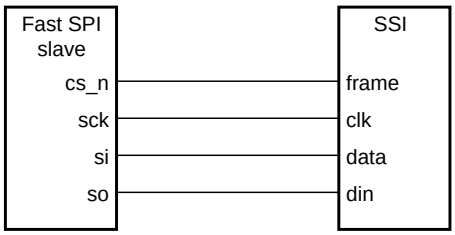


Figure 23.12: *Connection to an SPI slave*

At low clock frequencies, when the cycle time is much longer than the propagation delays involved, in ETRAX FS and in the external device, things are straightforward.

At high frequencies, timing must be analyzed carefully. For the receiver, the following sum of propagation delays has to be calculated:

1. SSI internal clock -> clock on the circuit board
2. circuit board trace delay of the clock signal
3. clock on circuit board -> data out from slave
4. circuit board trace delay of the data signal
5. data on circuit board -> input data inside the SSI

(considering all operating conditions, best to worst case). Knowing the best and worst case delays and the clock cycle time, it is easy to realize if the data is stable at the time of positive or negative clock edges inside the SSI (selected with rw_rec_cfg.clk_pol), and/or if the sampling point needs to be moved a whole clock cycle (using rw_frm_cfg.rec_delay). In some cases it might be necessary to externally delay data in and/or out a few nanoseconds to fulfill all timing requirements.

The Atmel flash cannot run at 100 MHz, but if a similar device is connected that can, an external delay circuit is most probably needed. When running at 100 MHz, data can only be output from the SSI on positive clock edges. Therefore a delay of nominally 5 ns is needed between **data** and ”si” in this case, while programming the SSI to output its data half a period earlier than usual.

23.7.5.2 Initial configuration

1. Set the rw_cfg.gate_clk to no, that field is only used for the normal clock gating in lowspeed mode.

2. Set rw_cfg.out_clk_pol to pos.

3. Set rw_cfg.out_clk_src to intern_clk if frequency is 50 MHz or below, to clk100 if running at 100 MHz.

4. The data width is arbitrary, as long as the total number of bits in a ”burst” is an even multiple of it, of course.

5. Set the rw_frm_cfg.wordrate to the value corresponding to the number of bits per word (e.g. 16 bits/word => set it to 15). Then there will be no space between the transmitted words.

6. Set transmitter, receiver and frame clock sources to internal clock, transmitter and frame clock polarity to negative edge.

7. Normally set rw_rec_cfg.clk_pol to positive edge clocking, since the slave outputs data on the negative edge. For higher clock frequencies, see the discussion in 23.7.5.1 above.

8. Set rw_frm_cfg.tr_delay to 1, since the first data out from the SSI shall appear at the same time as the frame signal (see the ”SI” and ”CS_n” signals in ”waveform 1” on page 17 in [ATMEL], and section 23.4.2.3.1 above).

9. At low frequencies, set rw_frm_cfg.rec_delay to 2. The value 2 is because of Atmel SPI mode 0 (see [Atmel]), where data output (SO) appears one cycle after the transmitted data. For same-cycle input/output SPI devices (and Atmel SPI mode 3 with other clock settings, not shown), use rec_delay = 1. For higher frequencies, see the discussion in 23.7.5.1 above.

10. Set transmitter and receiver shift direction to MSB first.

11. Set rw_tr_cfg.frm_src and rw_rec_cfg.frm_src to intern.

12. Set rw_tr_cfg.rate_ctrl to iso.

13. Set the rw_tr_cfg.eop_stop field and make sure the descriptor metadata is something else than ’tx_null’ (see 23.4.1.3.2).

14. Configure the **frame** pin as a general output, with value 1.

15. Set the fields rw_extra.clkon_en and rw_extra.clkoff_en, to enable the special output clock gating features.

16. Set rw_extra.clkoff_cycles to the number of bits, minus one, to transfer in one ”burst”.

17. Set the rw_cfg.prepare field.

23.7.5.3 Communication

1. Configure the SSI mode registers, keeping rw_cfg.en cleared and rw_cfg.prepare set. Communication will not start until rw_cfg.prepare is cleared, and the special clock gating feature will keep the clock at **clk** off until then.
2. Enable DMA or write first transmit data to mode register.
3. Wait at least (rw_cfg.clk_div+1)*30 ns
4. Set the rw_cfg.en field, keeping rw_cfg.prepare set.
5. Wait at least (rw_cfg.clk_div+2)*30 ns.
6. Set the **frame** pin low and wait the required frame-to-clock time (250 ns) (this delay may overlap bullet 5, ie frame may be lowered any time after bullet 4).
7. Clear the rw_cfg.prepare field. Now transmission will start and the output clock will be enabled. When the number of clock cycles set by rw_extra.clkoff_cycles has passed, the clock will be turned off. If EOP has been set at the correct position in the DMA, the trdy interrupt will be issued when all data has left the SSI.
8. Then wait the desired clock-to-frame time (250 ns) and set frame high.
9. Clear the rw_cfg.en field and set rw_cfg.prepare. After waiting the minimum inter-word gap, the next communication session can be started by repeating the steps from step 2 in this section.

Chapter 24

Electrical and Mechanical Information

24.1 DC Electrical specifications

24.1.1 Absolute maximum ratings

Symbol	Parameter	Min	Max	Unit
V_{DD15}	Core DC supply voltage.	-0.5	1.8	V
V_{DD33}	I/O DC supply voltage.	-0.5	4.8	V
$V_{DD33-15}$	I/O DC supply voltage referenced to core DC supply voltage.	-0.5	4.8	V
V_{in}	DC voltage applied on inputs.	-0.5	6.5	V
V_{out}	DC voltage applied on off-state outputs and I/O, all pins except **sdclk**, **u0vp** and **u0vm**.	-0.5	6.5	V
V_{out}	DC voltage applied on off-state outputs and I/O, **sdclk**, **u0vp** and **u0vm** pins.	-0.5	4.8	V
I_{IO}	Input/output current.	-20	20	mA
T_{stg}	Storage temperature.	-65	150	°C
T_A	Operating temperature.	-40	85	°C
P_D	Power dissipation.	-	2	W

Table 24.1: *Absolute maximum ratings*

24.1.2 ESD protection and latch-up

Test	Description	Standard	Test value	Unit
HBM	Human Body Model ESD test	JESD22-A114-B	2000	V
MM	Machine Model ESD test	JESD22-A115-A	200	V
CDM	Charged Device Model ESD test	JESD22-C101-A	300	V
Latch-up	Latch-up test	JESD78	200	mA

24.1.3 Recommended operating conditions

Symbol	Parameter	Min	Typical	Max	Unit
V_{DD15}	Core DC supply voltage.	1.4	1.5	1.6	V
V_{DD33}	I/O DC supply voltage.	3.0	3.3	3.6	V
V_{in}	DC voltage applied on inputs.	-0.3	-	5.5	V
V_{out}	DC voltage applied on off-state outputs and I/O, all pins except **sdclk**, **u0vp** and **u0vm**.	-0.3	-	5.5	V
V_{out}	DC voltage applied on off-state outputs and I/O, **sdclk**, **u0vp** and **u0vm** pins.	-0.3	-	V_{DD33} + 0.3	V
T_A	Operating temperature (Ambient temperature, no air flow).	-40	25	85	°C
I_{OH}	High level output current.	-	-	-4	mA
I_{OL}	Low level output current.	-	-	4	mA

Table 24.3: *Recommended operating conditions*

24.1.4 DC Electrical characteristics

Symbol	Parameter	Min	Typical	Max	Unit
V_{IH}	High level input voltage, all inputs and I/O except **sdclk**, **u0vp** and **u0vm**.	2.0	-	5.5	V
V_{IH}	High level input voltage, **sdclk** pin.	2.0	-	V_{DD33} + 0.3	V
V_{SEH}	Single ended high level input voltage, **u0vp** and **u0vm** pins.	2.0	-	V_{DD33} + 0.3	V
V_{IL}	Low level input voltage.	-0.3	-	0.8	V
V_{SEL}	Single ended low level input voltage, **u0vp** and **u0vm** pins.	-0.3	-	0.8	V
V_{DI}	Differential input voltage, **u0vp** and **u0vm** pins.	0.2	-	-	V
V_{CM}	Common mode input voltage, **u0vp** and **u0vm** pins (includes V_{DI} range).	0.8	-	2.5	V
V_{OH}	High level output voltage, I_{OH} = -4 mA, all outputs except **u0vp** and **u0vm** pins.	2.4	-	V_{DD33}	V
V_{OH}	High level output voltage, I_{OH} = -1 μA, all outputs except **u0vp** and **u0vm** pins.	V_{DD33} - 0.05	-	V_{DD33}	V
V_{OH}	High level output voltage, R_L = 15 kOhm to V_{SS}, **u0vp** and **u0vm** pins.	2.8	-	V_{DD33}	V
V_{OL}	Low level output voltage, I_{OL} = 4 mA, all outputs except **u0vp** and **u0vm** pins.	0	-	0.4	V
V_{OL}	Low level output voltage, I_{OL} = 1 μA, all outputs except **u0vp** and **u0vm** pins.	0	-	0.05	V
V_{OL}	Low level output voltage, R_L = 1.5 kOhm to V_{DD33}, **u0vp** and **u0vm** pins.	0	-	0.3	V

I_{in}	Input leakage current, all inputs except **trst**, **tck**, **tms** and **tdi**.	-10	-	10	μ A
I_{in}	Input current, **trst tck**, **tms** and **tdi**.	-71	-	10	μ A
I_{ioz}	I/O leakage current.	-10	-	10	μ A
C_{io}	Capacitance on any input, I/O or output pin except **u0vp** or **u0vm**.	-	5	6	pF
C_{io}	Capacitance on **u0vp** or **u0vm**.	-	-	20	pF
I_{DD15}	Core DC Supply current	-	200	950	mA
I_{DD33}	I/O DC Supply current	-	50	-	mA

Table 24.4: *DC Electrical characteristics*

24.1.4.1 Notes on supply current specifications

The supply current for ETRAX FS is widely dependent on the specific application. Typical values specified for I_{DD15} and I_{DD33} are measured with all functional units enabled, a Linux kernel running, communication on both Ethernet ports, and a USB 1.1 application running in the I/O processor. The maximum value for I_{DD15} is measured with an application specifically designed to draw maximum current.

24.1.5 PLL loop filter

The PLL requires a loop filter capacitor between the **plllpf** pin and Vss. Circuit board traces for the loop filter should be kept as short as possible.

Symbol	Parameter	Min	Typical	Max	Unit
C_{lpf}	External loop filter capacitor value	1.425	1.5	1.575	nF

Table 24.5: *PLL loop filter value*

24.1.6 Power up sequence

During power up, V_{DD33} should be applied before or at the same time as V_{DD15}. When powering down, V_{DD15} should be removed before or at the same time as V_{DD33}. The voltage difference between V_{DD33} and V_{DD15} should in no case exceed the values specified by the $V_{DD33-15}$ parameter given in 24.1.1.

24.2 AC Electrical specifications

For AC electrical specification information for the ETRAX FS, please see the respective chapter for each ETRAX FS interface.

24.2.1 Conditions

Timing information for the ETRAX FS is valid under the operating conditions given in the table below.

Condition	Value
T_A	0°C to 85°C
V_{DD15}	1.5 V +/- 0.1 V
V_{DD33}	3.3 V +/- 0.3 V
Capacitive load	50 pF unless explicitly stated otherwise

Table 24.6: *Operating conditions for timing information*

24.3 MTBF

MTBF figures are available upon request. Please contact Axis Communications for further information.

24.4 Pinout

An overview of the ETRAX FS pinout is shown in the figure below. A detailed pinout description is found in chapter 16.

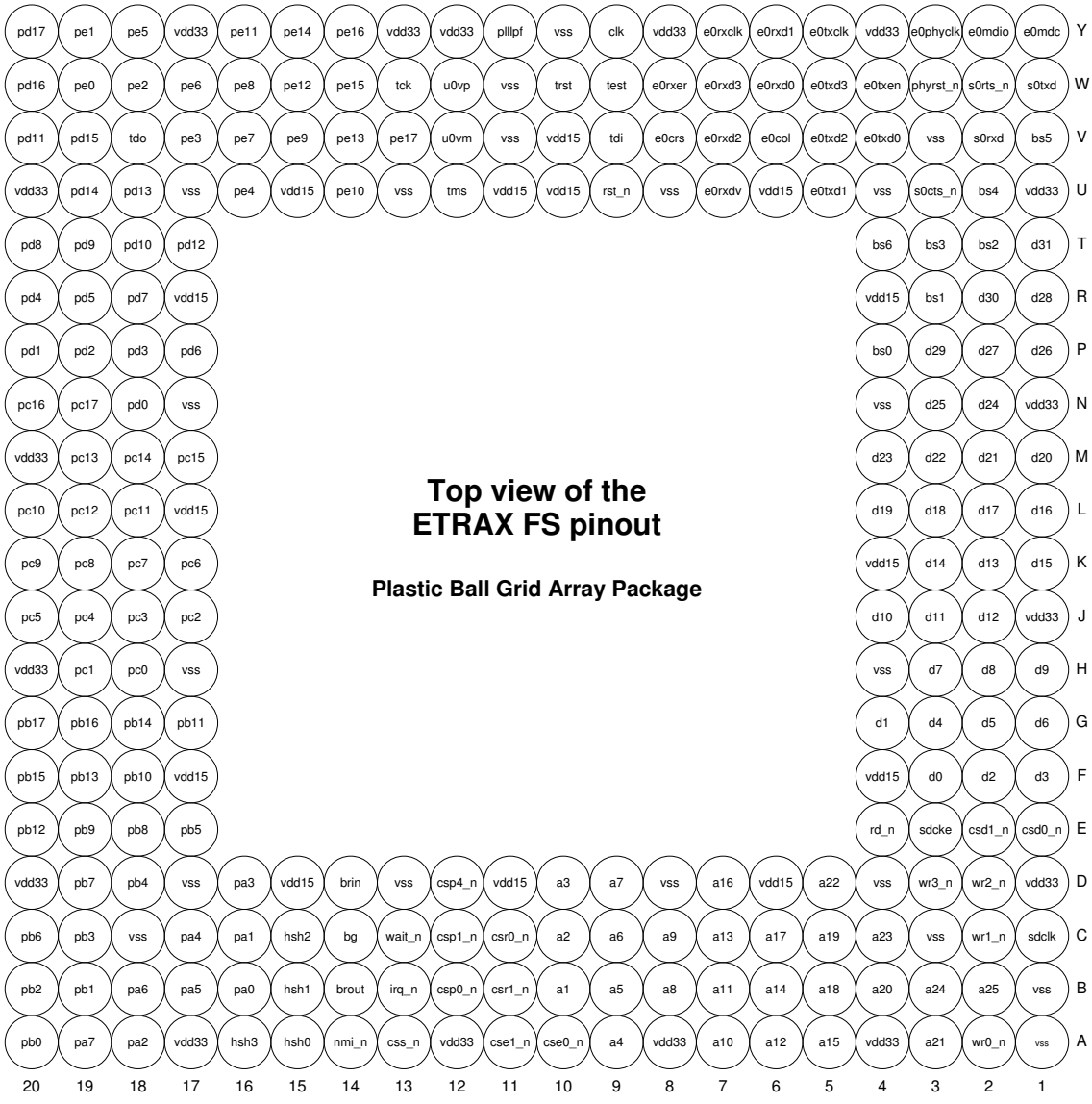


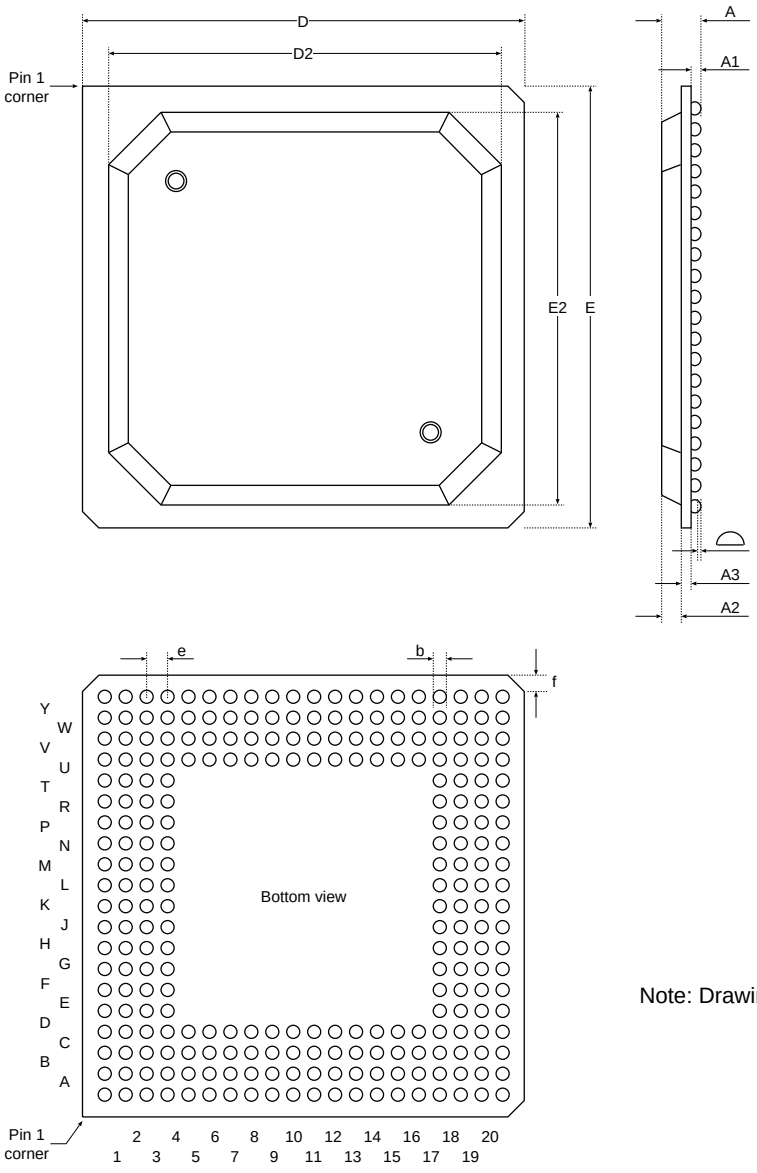
Figure 24.1: *The ETRAX FS pinout*

24.5 Mechanical specifications

24.5.1 Physical dimensions

The package of the ETRAX FS is a 256 lead Plastic Ball Grid Array (PBGA).

ETRAX FS 256 PBGA
Ball Grid Array Package - Mechanical drawing



Dimension		mm
A	Min	2.12
	Nom	2.33
	Max	2.52
A1	Min	0.50
	Nom	0.60
	Max	0.70
A2	Min	1.12
	Nom	1.17
	Max	1.22
A3	Min	0.50
	Nom	0.56
	Max	0.60
⌓	Max	0.15
b	Min	0.60
	Nom	0.76
	Max	0.90
D	Min	26.90
	Nom	27.00
	Max	27.10
D2	Min	23.95
	Nom	24.00
	Max	24.35
E	Min	26.90
	Nom	27.00
	Max	27.10
E2	Min	23.95
	Nom	24.00
	Max	24.35
e	Nom	1.27
f	Nom	0.50

Note: Drawing is not to scale.

Figure 24.2: *The ETRAX FS Plastic Ball Grid Array*

24.5.2 Marking

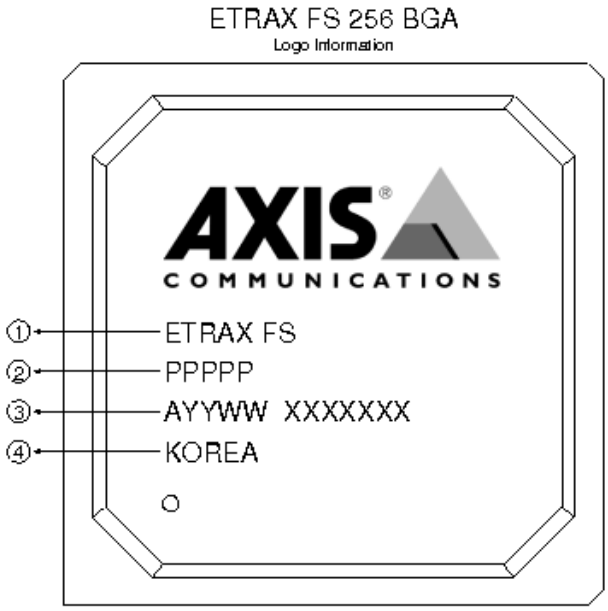


Figure 24.3: *ETRAX FS marking information*

The following list describes the marking on the ETRAX FS:

1. The name of the chip.
2. The Axis five digit part number PPPPP:
   - 24745 for a Pb-free package.
   - 21050 for a conventional package.
3. The third line contains three kinds of information:
   - A: Assembly site code.
   - YYWW: Work week code.
   - XXXXXXX: Fabrication number, up to seven digits.
4. The country where the ETRAX FS chip is produced.

24.5.3 RoHS conformance

The Pb-free version of ETRAX FS, part number 24745 , conforms to the European Union DIRECTIVE/2002/95/EC on the restriction of the use of certain hazardous substances in electrical and electronic equipment (RoHS directive). A declaration of used materials is available upon request. Please Contact Axis Communications for further information.

24.6 Soldering

24.6.1 Recommended soldering profile for Pb-free package

This specification applies to part number 24745.

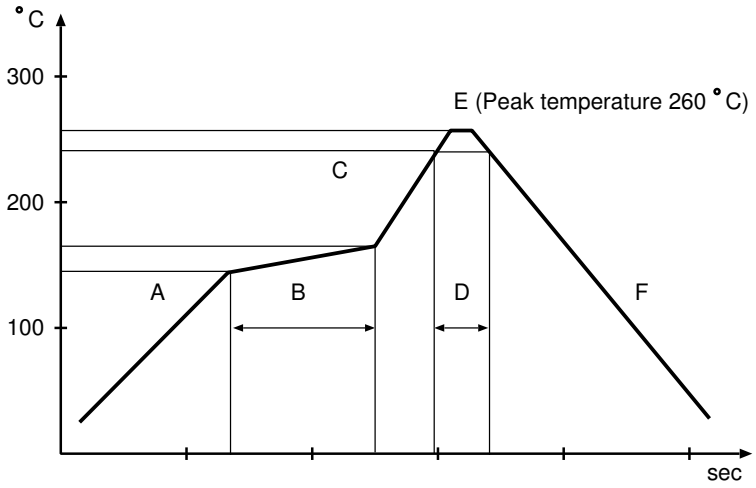


Figure 24.4: *Soldering profile for Pb-free package*

Item	Description	Min	Max	Unit
A	Heat-up	-	3	°C/s
B	Pre-heat	160 60	190 120	°C s
C	Heat-up	-	3	°C/s
D	Maintain	255 7	260 13	°C s
E	Re-flow peak	255	260	°C
F	Cooling down	-	6	°C/s

Table 24.7: *IR re-flow profile for Pb-free package*

24.6.2 Recommended soldering profile for conventional package

This specification applies to part number 21050.

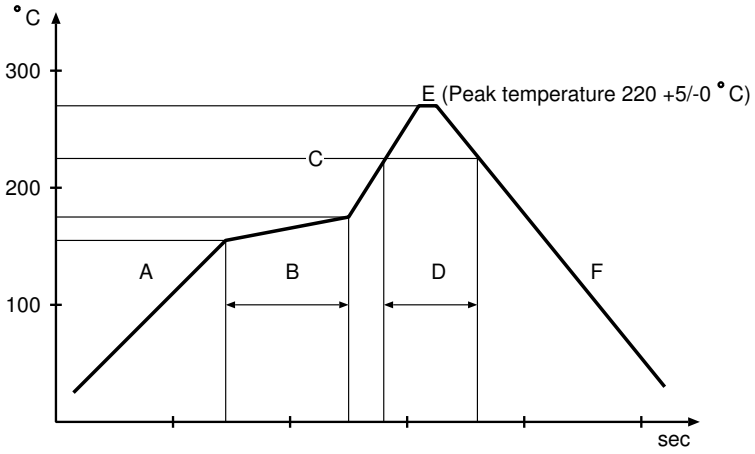


Figure 24.5: *Soldering profile for conventional package*

Item	Description	Min	Max	Unit
A	Heat-up	-	3	°C/s
B	Pre-heat	140 60	160 120	°C s
C	Heat-up	-	3	°C/s
D	Maintain	183 60	- 150	°C s
E	Re-flow peak	220 7	225 13	°C s
F	Cooling down	-	6	°C/s

Table 24.8: *IR re-flow profile for conventional package*

24.7 Delivery and storage

24.7.1 Delivery package

ETRAX FS is delivered in JEDEC CO-029 trays. Each tray carries 40 chips. A tray stack contains 10 filled trays, with an 11th empty tray on top. The tray stack contains 400 units. Each tray stack is placed in a sealed moisture barrier bag, together with one humidity indicator card and two desiccant bags. The moisture barrier bag is packed in a box. Bar code labels to identify the contents are placed on the moisture barrier bag and on the outside of the box.

Other packaging procedures than described above may apply for delivery quantities below 400 units.

24.7.2 Storage time

Maximum storage time in an unopened moisture barrier bag is 1 year.

24.7.3 Factory floor life and rebake procedure

The ETRAX FS is a Level 3 moisture sensitive device according to JEDEC standards J-STD-020A and J-STD-033B. After opening of the moisture barrier bag, factory floor life is limited to 168 h at 30°C/60% RH. If factory floor life time is exceeded, the units must be rebaked at 125°C for a minimum of 17 hours before soldering. The units can be rebaked without removing them from the trays. Rebake should be performed a maximum of 3 times.

Chapter 25

Internal Registers

25.1 Introduction

This chapter contains detailed information about the internal registers and support registers in the ETRAX FS. Each module is presented in a separate sub chapter. The register descriptions are presented as appropriate sets, within which the content of each separate register is specified to bit level.

The description of each module is introduced by a table listing the base addresses of all instances of the module, after which all the registers of the module are listed.

For each register is stated name, offset address within a module instance, default value, read/write capabilities and a description. For vector registers the addresses and, if available, default values are listed. All registers are 32 bit in size, of which all may or may not be used. The bit allocation of each register is then presented in a table as follows:

Bit(s)	Name	Description	Value
Bit no(s)	Name of field	Summary of the function of the field and associated signals.	Significance/value of each bit field.

The value column can list the state names and values of the field or not state anything in which case the value may be 0 to 2^(width of field)-1 unless otherwise stated in the description.

Chapter 25.45 contains a list of all register addresses in ETRAX FS, stating address, scope, instance and register name.

25.2 Notation

All hexadecimal values, including addresses, are preceded by 0x. All other values are given in decimal notation.

25.3 ata

Instance	Base Address
ata	0xb0032000

25.3.1 rw_ctrl2

Address	0x0
Default	0x00000000
Type	Read/Write
Description	ATA control register 2.

Bit(s)	Name	Description	Value
31-30:2	sel	Select ATA bus 0-3.	
29	cs1	Chip select 1 on ATA bus. active: Activate **cs1_n** inactive: Deactivate **cs1_n**	active=1 inactive=0
28	cs0	Chip select 0 on ATA bus. active: Activate **cs0_n** inactive: Deactivate **cs0_n**	active=1 inactive=0
27-25:3	addr	ATA bus device address.	
24	rw	Read/write select. rd: Read wr: Write	rd=1 wr=0
23	trf_mode	Data source/destination mode. dma: Data flows to/from the DMA reg: Register fields are used for data transfer	dma=1 reg=0
22-21:2	hsh	Select ATA handshaking mode. udma: Use UltraDMA mode handshaking dma: Use DMA mode handshaking pio: Use PIO mode handshaking	udma=2 dma=1 pio=0
20	multi	Choose between single or multiword transfer. yes: Keep transferring until the counter reaches zero no: Halt after every word transfer	yes=1 no=0
19	dma_size	Select size of DMA transfers. byte: Select 8-bit transfers word: Select 16-bit transfers	byte=1 word=0
15-0:16	data	Data sent to the device if in register transfer mode.	

25.3.2 rs_stat_data/r_stat_data

Address	0x4/0x8
Default	
Type	Read with side effects/Read
Description	ATA transfer status register.

Bit(s)	Name	Description	Value
17	busy	Indicates if the ATA interface is busy. yes: Busy no: Not busy	yes=1 no=0
16	dav	Set when it is possible to read new data in register mode. yes: Data available no: No data available	yes=1 no=0
15-0:16	data	Data read from ATA device while in register transfer mode.	

25.3.3 rw_ctrl0

Address	0xc
Default	0x00000000
Type	Read/Write
Description	ATA control register 0. This register controls interface enable and reset, and PIO and DMA mode timing. All times are (x + 1)*10 ns.

Bit(s)	Name	Description	Value
31	en	Enable the ATA controller. When disabled, the interface will be kept in a reset state. yes: Enabled no: Disabled	yes=1 no=0
30	rst	Controls the reset signal on the ATA bus. active: Reset active inactive: Reset inactive	active=1 inactive=0
29-24:6	dma_strb	Strobe time in DMA mode.	
23-18:6	dma_hold	Hold time for DMA mode.	
17-12:6	pio_setup	Setup time for PIO mode.	
11-6:6	pio_strb	Strobe time for PIO mode.	
5-0:6	pio_hold	Hold time for PIO mode.	

25.3.4 rw_ctrl1

Address	0x10
Default	
Type	Read/Write
Description	ATA control register 1. This register controls interface timing in UDMA mode. All times are (x + 1)*10 ns.

Bit(s)	Name	Description	Value
7-4:4	udma_tdvs	Data valid setup time for UDMA (tDVS).	
3-0:4	udma_tcyc	Cycle time for UDMA (tCYC - tDVS).	

25.3.5 rw_trf_cnt

Address	0x14
Default	
Type	Read/Write
Description	ATA transfer count register.

Bit(s)	Name	Description	Value
16-0:17	cnt	Number of entities (words or bytes) to transfer. When configured for UltraDMA, the counter value must include the word containing CRC and shall therefore be set to number of payload data entities + 1.	

25.3.6 r_stat_misc

Address	0x18
Default	
Type	Read
Description	ATA miscellaneous status register.

Bit(s)	Name	Description	Value
15-0:16	crc	Calculated CRC from the last UltraDMA transfer.	

25.3.7 rw_intr_mask

Address	0x1c
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts from the ATA interface. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	bus3	Enable/disable bus3 interrupt. Interrupt from ATA bus 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	bus2	Enable/disable bus2 interrupt. Interrupt from ATA bus 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	bus1	Enable/disable bus1 interrupt. Interrupt from ATA bus 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	bus0	Enable/disable bus0 interrupt. Interrupt from ATA bus 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.3.8 rw_ack_intr

Address	0x20
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts from the ATA interface.

Bit(s)	Name	Description	Value
3	bus3	Acknowledge bus3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	bus2	Acknowledge bus2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	bus1	Acknowledge bus1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	bus0	Acknowledge bus0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.3.9 r_intr

Address	0x24
Default	
Type	Read
Description	Interrupts before the mask. Interrupts from the ATA interface. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	bus3	Interrupt bus3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	bus2	Interrupt bus2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	bus1	Interrupt bus1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	bus0	Interrupt bus0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.3.10 r_masked_intr

Address	0x28
Default	
Type	Read
Description	Interrupts after the mask. Interrupts from the ATA interface. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	bus3	Interrupt bus3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	bus2	Interrupt bus2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	bus1	Interrupt bus1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	bus0	Interrupt bus0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.4 bif_core

Instance	Base Address
bif_core	0xb0014000

25.4.1 rw_grp1_cfg

Address	0x0
Default	0x000006cf
Type	Read/Write
Description	Configuration register for chip selects in group 1 (**cse0_n** and **cse1_n**).

Bit(s)	Name	Description	Value
21	mode	Write enable mode. cwe: Common write enable bwe: Bytewise write enable	cwe=1 bwe=0
20	erc_en	Early read complete mode. In early read complete mode, the **rd_n** signal goes high one clock cycle (10 ns) before the end of the bus cycle. Data is still sampled at the end of the bus cycle. yes: Early read complete mode no: Normal mode	yes=1 no=0
19	wr_extend	Write delay mode. In normal mode the write signal goes high 5 ns before the end of the bus cycle. In extended mode it goes high at the end of the bus cycle. yes: Extended mode no: Normal mode	yes=1 no=0
18	bw	Bus width. bw16: 16 bits wide data bus bw32: 32 bits wide data bus	bw16=1 bw32=0
17-16:2	ewb	Early wait states burst. The wait states are added to the early wait states of the first bus cycle in a burst.	
15-14:2	dw	Data setup wait states. The wait states are added in write cycles after the early wait states but before the write goes active. The data bus contains valid output data during the data setup wait states.	
13-12:2	aw	Address recovery wait states. Adds wait states to hold the address bus after chip select goes high.	
11-9:3	zw	Turn-off wait states. The wait states are added after the end of a bus burst, and may overlap with the early wait states of the next bus burst.	
8-6:3	ew	Early wait states. The wait states are added before the read or write goes active.	

5-0:6	lw	Late wait states. The wait states are added while the read or write is active.	

25.4.2 rw_grp2_cfg

Address	0x4
Default	0x000006cf
Type	Read/Write
Description	Configuration register for chip selects in group 2 (**csr0_n** and **csr1_n**).

Bit(s)	Name	Description	Value
21	mode	Write enable mode. cwe: Common write enable bwe: Bytewise write enable	cwe=1 bwe=0
20	erc_en	Early read complete mode. In early read complete mode, the **rd_n** signal goes high one clock cycle (10 ns) before the end of the bus cycle. Data is still sampled at the end of the bus cycle. yes: Early read complete mode no: Normal mode	yes=1 no=0
19	wr_extend	Write delay mode. In normal mode the write signal goes high 5 ns before the end of the bus cycle. In extended mode it goes high at the end of the bus cycle. yes: Extended mode no: Normal mode	yes=1 no=0
18	bw	Bus width. bw16: 16 bits wide data bus bw32: 32 bits wide data bus	bw16=1 bw32=0
17-16:2	ewb	Early wait states burst. The wait states are added to the early wait states of the first bus cycle in a burst.	
15-14:2	dw	Data setup wait states. The wait states are added in write cycles after the early wait states but before the write goes active. The data bus contains valid output data during the data setup wait states.	
13-12:2	aw	Address recovery wait states. Adds wait states to hold the address bus after chip select goes high.	
11-9:3	zw	Turn-off wait states. The wait states are added after the end of a bus burst, and may overlap with the early wait states of the next bus burst.	
8-6:3	ew	Early wait states. The wait states are added before the read or write goes active.	
5-0:6	lw	Late wait states. The wait states are added while the read or write is active.	

25.4.3 rw_grp3_cfg

Address	0x8
Default	0x000006cf
Type	Read/Write
Description	Configuration register for chip selects in group 3 (**csp0_n - csp3_n**).

Bit(s)	Name	Description	Value
31-30:2	gated_csp3	Selects if the **csp3_n** signal is gated with the internal read or write signal. If none the chip select works as normal. none: Not gated wr: Gated with write signal rd: Gated with read signal	none=0 wr=1 rd=2
29-28:2	gated_csp2	Selects if the **csp2_n** signal is gated with the internal read or write signal. If none the chip select works as normal. none: Not gated wr: Gated with write signal rd: Gated with read signal	none=0 wr=1 rd=2
27-26:2	gated_csp1	Selects if the **csp1_n** signal is gated with the internal read or write signal. If none the chip select works as normal. none: Not gated wr: Gated with write signal rd: Gated with read signal	none=0 wr=1 rd=2
25-24:2	gated_csp0	Selects if the **csp0_n** signal is gated with the internal read or write signal. If none the chip select works as normal. none: Not gated wr: Gated with write signal rd: Gated with read signal	none=0 wr=1 rd=2
21	mode	Write enable mode. cwe: Common write enable bwe: Bytewise write enable	cwe=1 bwe=0
20	erc_en	Early read complete mode. In early read complete mode, the **rd_n** signal goes high one clock cycle (10 ns) before the end of the bus cycle. Data is still sampled at the end of the bus cycle. yes: Early read complete mode no: Normal mode	yes=1 no=0
19	wr_extend	Write delay mode. In normal mode the write signal goes high 5 ns before the end of the bus cycle. In extended mode it goes high at the end of the bus cycle. yes: Extended mode no: Normal mode	yes=1 no=0
18	bw	Bus width. bw16: 16 bits wide data bus bw32: 32 bits wide data bus	bw16=1 bw32=0

17-16:2	ewb	Early wait states burst. The wait states are added to the early wait states of the first bus cycle in a burst.	
15-14:2	dw	Data setup wait states. The wait states are added in write cycles after the early wait states but before the write goes active. The data bus contains valid output data during the data setup wait states.	
13-12:2	aw	Address recovery wait states. Adds wait states to hold the address bus after chip select goes high.	
11-9:3	zw	Turn-off wait states. The wait states are added after the end of a bus burst, and may overlap with the early wait states of the next bus burst.	
8-6:3	ew	Early wait states. The wait states are added before the read or write goes active.	
5-0:6	lw	Late wait states. The wait states are added while the read or write is active.	

25.4.4 rw_grp4_cfg

Address	0xc
Default	0x000006cf
Type	Read/Write
Description	Configuration register for chip selects in group 4 (**csp4_n** - **csp6_n** and **css_n**).

Bit(s)	Name	Description	Value
31-30:2	gated_csp6	Selects if the **csp6_n** signal is gated with the internal read or write signal. If none the chip select works as normal. none: Not gated wr: Gated with write signal rd: Gated with read signal	none=0 wr=1 rd=2
29-28:2	gated_csp5	Selects if the **csp5_n** signal is gated with the internal read or write signal. If none the chip select works as normal. none: Not gated wr: Gated with write signal rd: Gated with read signal	none=0 wr=1 rd=2
27-26:2	gated_csp4	Selects if the **csp4_n** signal is gated with the internal read or write signal. If none the chip select works as normal. none: Not gated wr: Gated with write signal rd: Gated with read signal	none=0 wr=1 rd=2
21	mode	Write enable mode. cwe: Common write enable bwe: Bytewise write enable	cwe=1 bwe=0
20	erc_en	Early read complete mode. In early read complete mode, the **rd_n** signal goes high one clock cycle (10 ns) before the end of the bus cycle. Data is still sampled at the end of the bus cycle. yes: Early read complete mode no: Normal mode	yes=1 no=0
19	wr_extend	Write delay mode. In normal mode the write signal goes high 5 ns before the end of the bus cycle. In extended mode it goes high at the end of the bus cycle. yes: Extended mode no: Normal mode	yes=1 no=0
18	bw	Bus width. bw16: 16 bits wide data bus bw32: 32 bits wide data bus	bw16=1 bw32=0
17-16:2	ewb	Early wait states burst. The wait states are added to the early wait states of the first bus cycle in a burst.	
15-14:2	dw	Data setup wait states. The wait states are added in write cycles after the early wait states but before the write goes active. The data bus contains valid output data during the data setup wait states.	

13-12:2	aw	Address recovery wait states. Adds wait states to hold the address bus after chip select goes high.	
11-9:3	zw	Turn-off wait states. The wait states are added after the end of a bus burst, and may overlap with the early wait states of the next bus burst.	
8-6:3	ew	Early wait states. The wait states are added before the read or write goes active.	
5-0:6	lw	Late wait states. The wait states are added while the read or write is active.	

25.4.5 rw_sdram_cfg_grp0

Address	0x10
Default	
Type	Read/Write
Description	Configuration register for group 0 of SDRAM banks.

Bit(s)	Name	Description	Value
19-15:5	grp_sel	Configures how to select between the two groups of SDRAM banks. grp0: Always select group 0 grp1: Always select group 1 bit10: Use address bit 10 to select between groups bit11: Use address bit 11 to select between groups bit12: Use address bit 12 to select between groups bit13: Use address bit 13 to select between groups bit14: Use address bit 14 to select between groups bit15: Use address bit 15 to select between groups bit16: Use address bit 16 to select between groups bit17: Use address bit 17 to select between groups bit18: Use address bit 18 to select between groups bit19: Use address bit 19 to select between groups bit20: Use address bit 20 to select between groups bit21: Use address bit 21 to select between groups bit22: Use address bit 22 to select between groups bit23: Use address bit 23 to select between groups bit24: Use address bit 24 to select between groups bit25: Use address bit 25 to select between groups bit26: Use address bit 26 to select between groups bit27: Use address bit 27 to select between groups bit28: Use address bit 28 to select between groups bit29: Use address bit 29 to select between groups	grp0=0 grp1=1 bit10=10 bit11=11 bit12=12 bit13=13 bit14=14 bit15=15 bit16=16 bit17=17 bit18=18 bit19=19 bit20=20 bit21=21 bit22=22 bit23=23 bit24=24 bit25=25 bit26=26 bit27=27 bit28=28 bit29=29
14	sh16	Selects between normal and address shifted 16-bit mode. yes: Shifted 16-bit mode no: Normal mode	yes=1 no=0
13	wmm	Wide module mode. In wide module mode, all 8 **dqm** outputs are used in each group. The use of **dqm7** - **dqm4** or **dqm3** - **dqm0** is selected by the highest address bit in the selected column address range (ca field). The ca field should in this case be set to one bit higher than the highest column address bit to the SDRAM. This mode is used with 64-bit wide SDRAM modules that do not have separate chip selects for the upper and lower half of the data bus. yes: Wide module mode no: Normal mode	yes=1 no=0

12-10:3	sh	Row address shift. For bw == bw16 and sh16 == no: 0 == internal address 24-9 is shifted down to pin **a16 - a1**. 7 == internal address 29-16 is shifted down to pin **a14 - a1**. For bw == bw32 or sh16 == yes: 0 == internal address 23-9 is shifted down to pin **a16 - a2**. 7 == internal address 29-16 is shifted down to pin **a15 - a2**.	
9	bw	Bus width. bw16: 16 bits wide data bus bw32: 32 bits wide data bus	bw16=1 bw32=0
8	type	SDRAM type select. bank4: Four banks bank2: Two banks	bank4=1 bank2=0
7-5:3	ca	Column address range. This selects how many bits that are used for the column address. 0 == Up to and including address bit 8, 7 == up to and including bit 15.	
4-0:5	bank_sel	Selects which address bit that will be used to select between bank 0 and 1. For four-bank SDRAMs the specified address bit and the next higher order address bit will be used to select between banks 0 - 3. bit9: Use address bit 9 bit10: Use address bit 10 bit11: Use address bit 11 bit12: Use address bit 12 bit13: Use address bit 13 bit14: Use address bit 14 bit15: Use address bit 15 bit16: Use address bit 16 bit17: Use address bit 17 bit18: Use address bit 18 bit19: Use address bit 19 bit20: Use address bit 20 bit21: Use address bit 21 bit22: Use address bit 22 bit23: Use address bit 23 bit24: Use address bit 24 bit25: Use address bit 25 bit26: Use address bit 26 bit27: Use address bit 27 bit28: Use address bit 28 bit29: Use address bit 29	bit9=9 bit10=10 bit11=11 bit12=12 bit13=13 bit14=14 bit15=15 bit16=16 bit17=17 bit18=18 bit19=19 bit20=20 bit21=21 bit22=22 bit23=23 bit24=24 bit25=25 bit26=26 bit27=27 bit28=28 bit29=29

25.4.6 rw_sdram_cfg_grp1

Address	0x14
Default	
Type	Read/Write
Description	Configuration register for group 1 of SDRAM banks.

Bit(s)	Name	Description	Value
14	sh16	Selects between normal and address shifted 16-bit mode. yes: Shifted 16-bit mode no: Normal mode	yes=1 no=0
13	wmm	Wide module mode. In wide module mode, all 8 **dqm** outputs are used in each group. The use of **dqm7** - **dqm4** or **dqm3** - **dqm0** is selected by the highest address bit in the selected column address range (ca field). The ca field should in this case be set to one bit higher than the highest column address bit to the SDRAM. This mode is used with 64-bit wide SDRAM modules that do not have separate chip selects for the upper and lower half of the data bus. yes: Wide module mode no: Normal mode	yes=1 no=0
12-10:3	sh	Row address shift. For bw == bw16 and sh16 == no: 0 == internal address 24-9 is shifted down to pin **a16** - **a1**. 7 == internal address 29-16 is shifted down to pin **a14** - **a1**. For bw == bw32 or sh16 == yes: 0 == internal address 23-9 is shifted down to pin **a16** - **a2**. 7 == internal address 29-16 is shifted down to pin **a15** - **a2**.	
9	bw	Bus width. bw16: 16 bits wide data bus bw32: 32 bits wide data bus	bw16=1 bw32=0
8	type	SDRAM type select. bank4: Four banks bank2: Two banks	bank4=1 bank2=0
7-5:3	ca	Column address range. This selects how many bits that are used for the column address. 0 == Up to and including address bit 8, 7 == up to and including bit 15.	

4-0:5	bank_sel	Selects which address bit that will be used to select between bank 0 and 1. For four-bank SDRAMs the specified address bit and the next higher order address bit will be used to select between banks 0 - 3. bit9: Use address bit 9 bit10: Use address bit 10 bit11: Use address bit 11 bit12: Use address bit 12 bit13: Use address bit 13 bit14: Use address bit 14 bit15: Use address bit 15 bit16: Use address bit 16 bit17: Use address bit 17 bit18: Use address bit 18 bit19: Use address bit 19 bit20: Use address bit 20 bit21: Use address bit 21 bit22: Use address bit 22 bit23: Use address bit 23 bit24: Use address bit 24 bit25: Use address bit 25 bit26: Use address bit 26 bit27: Use address bit 27 bit28: Use address bit 28 bit29: Use address bit 29	bit9=9 bit10=10 bit11=11 bit12=12 bit13=13 bit14=14 bit15=15 bit16=16 bit17=17 bit18=18 bit19=19 bit20=20 bit21=21 bit22=22 bit23=23 bit24=24 bit25=25 bit26=26 bit27=27 bit28=28 bit29=29

25.4.7 rw_sdram_timing

Address	0x18
Default	
Type	Read/Write
Description	Timing parameters for the SDRAM interface. The first write to this register after reset starts the SDRAM interface.

Bit(s)	Name	Description	Value
18	sdclk	Disable SDRAM clock. When disabled, the **sdclk** output is constantly low. yes: Disable SDRAM clock no: Normal mode	yes=1 no=0
17	sdcke	Disable SDRAM clock enable signal. When disabled, the **sdcke** output is constantly low. yes: Disable SDRAM cke no: Normal mode	yes=1 no=0
16	cpd	Chip PLL disabled mode. Ensures that enough refreshes are issued when the PLL is bypassed. yes: Chip PLL disabled mode no: Normal	yes=1 no=0
15-14:2	ref	The SDRAM refresh interval. e7800ns: Every 7800 ns e15us: Every 15 μs off: Disable	e7800ns=2 e15us=1 off=0
13	pde	SDRAM self refresh exit delay. The number of delay cycles are pde + 1.	
12-11:2	dpl	Data write to precharge delay. The number of delay cycles are dpl.	
10-9:2	rc	Row cycle time. The number of refresh cycles are rc + 6.	
8-6:3	rp	RAS precharge delay. The number of delay cycles are rp + 1.	
5-3:3	rcd	RAS to CAS delay. The number of delay cycles are rcd + 1.	
2-0:3	cl	CAS latency. The number of latency cycles are cl.	

25.4.8 rw_sdram_cmd

Address	0x1c
Default	
Type	Read/Write
Description	SDRAM commands during initialization.

Bit(s)	Name	Description	Value
17-3:15	mrs_data	Data output during SDRAM mode register set cycle.	
2-0:3	cmd	Initiate an SDRAM command cycle. The available commands are nop, auto refresh (ref), precharge all (pre) and mode register set (mrs). slf: Enter/exit self refresh mode mrs: Mode register set pre: Precharge all ref: Auto refresh nop: Deselect	slf=4 mrs=3 pre=2 ref=1 nop=0

25.4.9 rs_sdram_ref_stat/r_sdram_ref_stat

Address	0x20/0x24
Default	0x00000001
Type	Read with side effects/Read
Description	SDRAM refresh status register. Read with side effect will set the ok field to yes.

Bit(s)	Name	Description	Value
0	ok	Indicates if SDRAM refresh cycles have been issued at the desired rate or not. If the bus is unavailable to the SDRAM controller for a consecutive time of more than eight refresh intervals, the average DRAM refresh rate may become degraded. yes: Normal refresh rate no: Refresh rate may have become degraded	yes=1 no=0

25.5 bif_dma

Instance	Base Address
bif_dma	0xb0016000

25.5.1 rw_ch0_ctrl

Address	0x0
Default	0x00000000
Type	Read/Write
Description	External DMA control register.

Bit(s)	Name	Description	Value
19	wr_all	If this field is set to yes, the external DMA write cycles will be performed as 32-bit cycles regardless of the actual data width. Only the number of bytes corresponding to the actual data width will contain valid data. no: Bus cycle width equal to actual data width yes: 32-bit bus cycle width	no=0 yes=1
18	rate_en	Rate control enable. If enabled, the channel bandwidth is limited by the clock frequency of the rate clock input. no: Rate clock disabled yes: Rate clock enabled	no=0 yes=1
17-16:2	bus_mode	Enable master mode or slave mode operation. off: Channel disabled master: Master mode enabled slave: Slave mode enabled	off=0 master=1 slave=2
15-14:2	tc_in_mode	Select mode for the terminal count input. off: No tc_in force: All transfers terminate after one burst norm: Active high tc_in from pin inv: Active low tc_in from pin	off=0 force=1 norm=2 inv=3
13-11:3	tc_in_pin	Selects which pin **hsh0 - hsh7** to use as tc_in for this channel.	
10-9:2	dreq_mode	Select mode for the dreq input. off: No dreq force: Constant dreq norm: Active high dreq from pin inv: Active low dreq from pin	off=0 force=1 norm=2 inv=3
8-6:3	dreq_pin	Selects which pin **hsh0 - hsh7** to use as dreq for this channel.	

5	cnt	Selects if the transfer counter shall cause any effects or not. no: Transfer counter ignored yes: Transfer counter takes effect	no=0 yes=1
4	end_pad	Selects what will happen with the data in the remaining cycles of a burst, if the transfer counter expires in the middle of the burst. no: Continue to send data until end of burst yes: Pad the remaining cycles with 0	no=0 yes=1
3	cont	Enable/disable continuous transfer mode. no: Disabled yes: Enabled	no=0 yes=1
2	burst_len	Burst length in number of bus cycles. burst1: 1 bus cycle per burst burst8: 8 bus cycles per burst	burst1=0 burst8=1
1-0:2	bw	Bus width for the DMA transfers. bw8: 8-bit bus width bw16: 16-bit bus width bw32: 32-bit bus width	bw8=0 bw16=1 bw32=2

25.5.2 rw_ch0_addr

Address	0x4
Default	
Type	Read/Write
Description	External DMA address register.

Bit(s)	Name	Description	Value
31-0:32	addr	Address used for the external DMA accesses. Bit 31, 30, 1 and 0 are ignored by the external DMA unit.	

25.5.3 rw_ch0_start

Address	0x8
Default	0x00000000
Type	Read/Write
Description	External DMA start/stop register.

Bit(s)	Name	Description	Value
0	run	Start and stop the external DMA. When read, this field shows if the channel is stopped or running. It may indicate a stopped condition during the last burst of the DMA transfer. Use the run field of the r_ch0_stat register instead to check if the current DMA transfer is completed. no: Channel is stopped yes: Channel is started	no=0 yes=1

25.5.4 rw_ch0_cnt

Address	0xc
Default	
Type	Read/Write
Description	Transfer counter start value.

Bit(s)	Name	Description	Value
15-0:16	start_cnt	Transfer counter start value. A start value of 0 gives 65536 transfers.	

25.5.5 r_ch0_stat

Address	0x10
Default	
Type	Read
Description	Status for the external DMA channel.

Bit(s)	Name	Description	Value
31	run	This bit shows if the channel is stopped or running. It will indicate a running status until the last burst of the transfer is completed. no: Channel is stopped yes: Channel is running	no=0 yes=1
15-0:16	cnt	Current value of the transfer counter.	

25.5.6 rw_ch1_ctrl

Address	0x20
Default	0x00000000
Type	Read/Write
Description	External DMA control register.

Bit(s)	Name	Description	Value
18	rate_en	Rate control enable. If enabled, the channel bandwidth is limited by the clock frequency of the rate clock input. no: Rate clock disabled yes: Rate clock enabled	no=0 yes=1
17-16:2	bus_mode	Enable master mode or slave mode operation. off: Channel disabled master: Master mode enabled slave: Slave mode enabled	off=0 master=1 slave=2
15-14:2	tc_in_mode	Select mode for the terminal count input. off: No tc_in force: All transfers terminate after one burst norm: Active high tc_in from pin inv: Active low tc_in from pin	off=0 force=1 norm=2 inv=3
13-11:3	tc_in_pin	Selects which pin **hsh0 - hsh7** to use as tc_in for this channel.	
10-9:2	dreq_mode	Select mode for the dreq input. off: No dreq force: Constant dreq norm: Active high dreq from pin inv: Active low dreq from pin	off=0 force=1 norm=2 inv=3
8-6:3	dreq_pin	Selects which pin **hsh0 - hsh7** to use as dreq for this channel.	
5	cnt	Selects if the transfer counter shall cause any effects or not. no: Transfer counter ignored yes: Transfer counter takes effect	no=0 yes=1
4	end_discard	Selects what will happen with the data in the remaining cycles of a burst, if the transfer counter expires in the middle of the burst. no: Receive the remaining data yes: Discard the remaining data	no=0 yes=1
3	cont	Enable/disable continuous transfer mode. no: Disabled yes: Enabled	no=0 yes=1
2	burst_len	Burst length in number of bus cycles. burst1: 1 bus cycle per burst burst8: 8 bus cycles per burst	burst1=0 burst8=1

1-0:2	bw	Bus width for the DMA transfers. bw8: 8-bit bus width bw16: 16-bit bus width bw32: 32-bit bus width	bw8=0 bw16=1 bw32=2

25.5.7 rw_ch1_addr

Address	0x24
Default	
Type	Read/Write
Description	External DMA address register.

Bit(s)	Name	Description	Value
31-0:32	addr	Address used for the external DMA accesses. Bit 31, 30, 1 and 0 are ignored by the external DMA unit.	

25.5.8 rw_ch1_start

Address	0x28
Default	0x00000000
Type	Read/Write
Description	External DMA start/stop register.

Bit(s)	Name	Description	Value
0	run	Start and stop the external DMA. When read, this field shows if the channel is stopped or running. It may indicate a stopped condition during the last burst of the DMA transfer. Use the run field of the r_ch1_stat register instead to check if the current DMA transfer is completed. no: Channel is stopped yes: Channel is started	no=0 yes=1

25.5.9 rw_ch1_cnt

Address	0x2c
Default	
Type	Read/Write
Description	Transfer counter start value.

Bit(s)	Name	Description	Value
15-0:16	start_cnt	Transfer counter start value. A start value of 0 gives 65536 transfers.	

25.5.10 r_ch1_stat

Address	0x30
Default	
Type	Read
Description	Status for the external DMA channel.

Bit(s)	Name	Description	Value
31	run	This bit shows if the channel is stopped or running. It will indicate a running status until the last burst of the transfer is completed. no: Channel is stopped yes: Channel is running	no=0 yes=1
15-0:16	cnt	Current value of the transfer counter.	

25.5.11 rw_ch2_ctrl

Address	0x40
Default	0x00000000
Type	Read/Write
Description	External DMA control register.

Bit(s)	Name	Description	Value
19	wr_all	If this field is set to yes, the external DMA write cycles will be performed as 32-bit cycles regardless of the actual data width. Only the number of bytes corresponding to the actual data width will contain valid data. no: Bus cycle width equal to actual data width yes: 32-bit bus cycle width	no=0 yes=1
18	rate_en	Rate control enable. If enabled, the channel bandwidth is limited by the clock frequency of the rate clock input. no: Rate clock disabled yes: Rate clock enabled	no=0 yes=1
17-16:2	bus_mode	Enable master mode or slave mode operation. off: Channel disabled master: Master mode enabled slave: Slave mode enabled	off=0 master=1 slave=2
15-14:2	tc_in_mode	Select mode for the terminal count input. off: No tc_in force: All transfers terminate after one burst norm: Active high tc_in from pin inv: Active low tc_in from pin	off=0 force=1 norm=2 inv=3
13-11:3	tc_in_pin	Selects which pin **hsh0 - hsh7** to use as tc_in for this channel.	
10-9:2	dreq_mode	Select mode for the dreq input. off: No dreq force: Constant dreq norm: Active high dreq from pin inv: Active low dreq from pin	off=0 force=1 norm=2 inv=3
8-6:3	dreq_pin	Selects which pin **hsh0 - hsh7** to use as dreq for this channel.	
5	cnt	Selects if the transfer counter shall cause any effects or not. no: Transfer counter ignored yes: Transfer counter takes effect	no=0 yes=1
4	end_pad	Selects what will happen with the data in the remaining cycles of a burst, if the transfer counter expires in the middle of the burst. no: Continue to send data until end of burst yes: Pad the remaining cycles with 0	no=0 yes=1

3	cont	Enable/disable continuous transfer mode. no: Disabled yes: Enabled	no=0 yes=1
2	burst_len	Burst length in number of bus cycles. burst1: 1 bus cycle per burst burst8: 8 bus cycles per burst	burst1=0 burst8=1
1-0:2	bw	Bus width for the DMA transfers. bw8: 8-bit bus width bw16: 16-bit bus width bw32: 32-bit bus width	bw8=0 bw16=1 bw32=2

25.5.12 rw_ch2_addr

Address	0x44
Default	
Type	Read/Write
Description	External DMA address register.

Bit(s)	Name	Description	Value
31-0:32	addr	Address used for the external DMA accesses. Bit 31, 30, 1 and 0 are ignored by the external DMA unit.	

25.5.13 rw_ch2_start

Address	0x48
Default	0x00000000
Type	Read/Write
Description	External DMA start/stop register.

Bit(s)	Name	Description	Value
0	run	Start and stop the external DMA. When read, this field shows if the channel is stopped or running. It may indicate a stopped condition during the last burst of the DMA transfer. Use the run field of the r_ch2_stat register instead to check if the current DMA transfer is completed. no: Channel is stopped yes: Channel is started	no=0 yes=1

25.5.14 rw_ch2_cnt

Address	0x4c
Default	
Type	Read/Write
Description	Transfer counter start value.

Bit(s)	Name	Description	Value
15-0:16	start_cnt	Transfer counter start value. A start value of 0 gives 65536 transfers.	

25.5.15 r_ch2_stat

Address	0x50
Default	
Type	Read
Description	Status for the external DMA channel.

Bit(s)	Name	Description	Value
31	run	This bit shows if the channel is stopped or running. It will indicate a running status until the last burst of the transfer is completed. no: Channel is stopped yes: Channel is running	no=0 yes=1
15-0:16	cnt	Current value of the transfer counter.	

25.5.16 rw_ch3_ctrl

Address	0x60
Default	0x00000000
Type	Read/Write
Description	External DMA control register.

Bit(s)	Name	Description	Value
18	rate_en	Rate control enable. If enabled, the channel bandwidth is limited by the clock frequency of the rate clock input. no: Rate clock disabled yes: Rate clock enabled	no=0 yes=1
17-16:2	bus_mode	Enable master mode or slave mode operation. off: Channel disabled master: Master mode enabled slave: Slave mode enabled	off=0 master=1 slave=2
15-14:2	tc_in_mode	Select mode for the terminal count input. off: No tc_in force: All transfers terminate after one burst norm: Active high tc_in from pin inv: Active low tc_in from pin	off=0 force=1 norm=2 inv=3
13-11:3	tc_in_pin	Selects which pin **hsh0 - hsh7** to use as tc_in for this channel.	
10-9:2	dreq_mode	Select mode for the dreq input. off: No dreq force: Constant dreq norm: Active high dreq from pin inv: Active low dreq from pin	off=0 force=1 norm=2 inv=3
8-6:3	dreq_pin	Selects which pin **hsh0 - hsh7** to use as dreq for this channel.	
5	cnt	Selects if the transfer counter shall cause any effects or not. no: Transfer counter ignored yes: Transfer counter takes effect	no=0 yes=1
4	end_discard	Selects what will happen with the data in the remaining cycles of a burst, if the transfer counter expires in the middle of the burst. no: Receive the remaining data yes: Discard the remaining data	no=0 yes=1
3	cont	Enable/disable continuous transfer mode. no: Disabled yes: Enabled	no=0 yes=1
2	burst_len	Burst length in number of bus cycles. burst1: 1 bus cycle per burst burst8: 8 bus cycles per burst	burst1=0 burst8=1

1-0:2	bw	Bus width for the DMA transfers. bw8: 8-bit bus width bw16: 16-bit bus width bw32: 32-bit bus width	bw8=0 bw16=1 bw32=2

25.5.17 rw_ch3_addr

Address	0x64
Default	
Type	Read/Write
Description	External DMA address register.

Bit(s)	Name	Description	Value
31-0:32	addr	Address used for the external DMA accesses. Bit 31, 30, 1 and 0 are ignored by the external DMA unit.	

25.5.18 rw_ch3_start

Address	0x68
Default	0x00000000
Type	Read/Write
Description	External DMA start/stop register.

Bit(s)	Name	Description	Value
0	run	Start and stop the external DMA. When read, this field shows if the channel is stopped or running. It may indicate a stopped condition during the last burst of the DMA transfer. Use the run field of the r_ch3_stat register instead to check if the current DMA transfer is completed. no: Channel is stopped yes: Channel is started	no=0 yes=1

25.5.19 rw_ch3_cnt

Address	0x6c
Default	
Type	Read/Write
Description	Transfer counter start value.

Bit(s)	Name	Description	Value
15-0:16	start_cnt	Transfer counter start value. A start value of 0 gives 65536 transfers.	

25.5.20 r_ch3_stat

Address	0x70
Default	
Type	Read
Description	Status for the external DMA channel.

Bit(s)	Name	Description	Value
31	run	This bit shows if the channel is stopped or running. It will indicate a running status until the last burst of the transfer is completed. no: Channel is stopped yes: Channel is running	no=0 yes=1
15-0:16	cnt	Current value of the transfer counter.	

25.5.21 rw_intr_mask

Address	0x80
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts from external DMA channels. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	ext_dma3	Enable/disable ext_dma3 interrupt. Interrupt from external DMA channel 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	ext_dma2	Enable/disable ext_dma2 interrupt. Interrupt from external DMA channel 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	ext_dma1	Enable/disable ext_dma1 interrupt. Interrupt from external DMA channel 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	ext_dma0	Enable/disable ext_dma0 interrupt. Interrupt from external DMA channel 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.5.22 rw_ack_intr

Address	0x84
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts from external DMA channels.

Bit(s)	Name	Description	Value
3	ext_dma3	Acknowledge ext_dma3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	ext_dma2	Acknowledge ext_dma2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	ext_dma1	Acknowledge ext_dma1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	ext_dma0	Acknowledge ext_dma0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.5.23 r_intr

Address	0x88
Default	
Type	Read
Description	Interrupts before the mask. Interrupts from external DMA channels. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	ext_dma3	Interrupt ext_dma3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	ext_dma2	Interrupt ext_dma2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	ext_dma1	Interrupt ext_dma1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	ext_dma0	Interrupt ext_dma0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.5.24 r_masked_intr

Address	0x8c
Default	
Type	Read
Description	Interrupts after the mask. Interrupts from external DMA channels. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	ext_dma3	Interrupt ext_dma3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	ext_dma2	Interrupt ext_dma2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	ext_dma1	Interrupt ext_dma1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	ext_dma0	Interrupt ext_dma0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.5.25 rw_pin0_cfg

Address	0xa0
Default	0x00000000
Type	Read/Write
Description	Configuration register for external DMA/slave mode handshake pin **hsh0** output.

Bit(s)	Name	Description	Value
9-7:3	slave_mode	Pin usage when in slave mode. off: The output is off, and the pin can be used as input as_master: The pin maintains its master mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low sreq: Used as slave request, active high sreq_inv: Used as slave request, active low	off=0 as_master=1 lo=2 hi=3 tc=4 tc_inv=5 sreq=6 sreq_inv=7
6-5:2	slave_ch	Slave mode channel connected to the pin.	
4-2:3	master_mode	Pin usage when in master mode. off: The output is off, and the pin can be used as input as_slave: The pin maintains its slave mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low dack: Used as DMA acknowledge, active high dack_inv: Used as DMA acknowledge, active low	off=0 as_slave=1 lo=2 hi=3 tc=4 tc_inv=5 dack=6 dack_inv=7
1-0:2	master_ch	External DMA channel connected to the pin in master mode.	

25.5.26 rw_pin1_cfg

Address	0xa4
Default	0x00000000
Type	Read/Write
Description	Configuration register for external DMA/slave mode handshake pin **hsh1** output.

Bit(s)	Name	Description	Value
9-7:3	slave_mode	Pin usage when in slave mode. off: The output is off, and the pin can be used as input as_master: The pin maintains its master mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low sreq: Used as slave request, active high sreq_inv: Used as slave request, active low	off=0 as_master=1 lo=2 hi=3 tc=4 tc_inv=5 sreq=6 sreq_inv=7
6-5:2	slave_ch	Slave mode channel connected to the pin.	
4-2:3	master_mode	Pin usage when in master mode. off: The output is off, and the pin can be used as input as_slave: The pin maintains its slave mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low dack: Used as DMA acknowledge, active high dack_inv: Used as DMA acknowledge, active low	off=0 as_slave=1 lo=2 hi=3 tc=4 tc_inv=5 dack=6 dack_inv=7
1-0:2	master_ch	External DMA channel connected to the pin in master mode.	

25.5.27 rw_pin2_cfg

Address	0xa8
Default	0x00000000
Type	Read/Write
Description	Configuration register for external DMA/slave mode handshake pin **hsh2** output.

Bit(s)	Name	Description	Value
9-7:3	slave_mode	Pin usage when in slave mode. off: The output is off, and the pin can be used as input as_master: The pin maintains its master mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low sreq: Used as slave request, active high sreq_inv: Used as slave request, active low	off=0 as_master=1 lo=2 hi=3 tc=4 tc_inv=5 sreq=6 sreq_inv=7
6-5:2	slave_ch	Slave mode channel connected to the pin.	
4-2:3	master_mode	Pin usage when in master mode. off: The output is off, and the pin can be used as input as_slave: The pin maintains its slave mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low dack: Used as DMA acknowledge, active high dack_inv: Used as DMA acknowledge, active low	off=0 as_slave=1 lo=2 hi=3 tc=4 tc_inv=5 dack=6 dack_inv=7
1-0:2	master_ch	External DMA channel connected to the pin in master mode.	

25.5.28 rw_pin3_cfg

Address	0xac
Default	0x00000000
Type	Read/Write
Description	Configuration register for external DMA/slave mode handshake pin **hsh3** output.

Bit(s)	Name	Description	Value
9-7:3	slave_mode	Pin usage when in slave mode. off: The output is off, and the pin can be used as input as_master: The pin maintains its master mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low sreq: Used as slave request, active high sreq_inv: Used as slave request, active low	off=0 as_master=1 lo=2 hi=3 tc=4 tc_inv=5 sreq=6 sreq_inv=7
6-5:2	slave_ch	Slave mode channel connected to the pin.	
4-2:3	master_mode	Pin usage when in master mode. off: The output is off, and the pin can be used as input as_slave: The pin maintains its slave mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low dack: Used as DMA acknowledge, active high dack_inv: Used as DMA acknowledge, active low	off=0 as_slave=1 lo=2 hi=3 tc=4 tc_inv=5 dack=6 dack_inv=7
1-0:2	master_ch	External DMA channel connected to the pin in master mode.	

25.5.29 rw_pin4_cfg

Address	0xb0
Default	0x00000000
Type	Read/Write
Description	Configuration register for external DMA/slave mode handshake pin **hsh4** output.

Bit(s)	Name	Description	Value
9-7:3	slave_mode	Pin usage when in slave mode. off: The output is off, and the pin can be used as input as_master: The pin maintains its master mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low sreq: Used as slave request, active high sreq_inv: Used as slave request, active low	off=0 as_master=1 lo=2 hi=3 tc=4 tc_inv=5 sreq=6 sreq_inv=7
6-5:2	slave_ch	Slave mode channel connected to the pin.	
4-2:3	master_mode	Pin usage when in master mode. off: The output is off, and the pin can be used as input as_slave: The pin maintains its slave mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low dack: Used as DMA acknowledge, active high dack_inv: Used as DMA acknowledge, active low	off=0 as_slave=1 lo=2 hi=3 tc=4 tc_inv=5 dack=6 dack_inv=7
1-0:2	master_ch	External DMA channel connected to the pin in master mode.	

25.5.30 rw_pin5_cfg

Address	0xb4
Default	0x00000000
Type	Read/Write
Description	Configuration register for external DMA/slave mode handshake pin **hsh5** output.

Bit(s)	Name	Description	Value
9-7:3	slave_mode	Pin usage when in slave mode. off: The output is off, and the pin can be used as input as_master: The pin maintains its master mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low sreq: Used as slave request, active high sreq_inv: Used as slave request, active low	off=0 as_master=1 lo=2 hi=3 tc=4 tc_inv=5 sreq=6 sreq_inv=7
6-5:2	slave_ch	Slave mode channel connected to the pin.	
4-2:3	master_mode	Pin usage when in master mode. off: The output is off, and the pin can be used as input as_slave: The pin maintains its slave mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low dack: Used as DMA acknowledge, active high dack_inv: Used as DMA acknowledge, active low	off=0 as_slave=1 lo=2 hi=3 tc=4 tc_inv=5 dack=6 dack_inv=7
1-0:2	master_ch	External DMA channel connected to the pin in master mode.	

25.5.31 rw_pin6_cfg

Address	0xb8
Default	0x00000000
Type	Read/Write
Description	Configuration register for external DMA/slave mode handshake pin **hsh6** output.

Bit(s)	Name	Description	Value
9-7:3	slave_mode	Pin usage when in slave mode. off: The output is off, and the pin can be used as input as_master: The pin maintains its master mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low sreq: Used as slave request, active high sreq_inv: Used as slave request, active low	off=0 as_master=1 lo=2 hi=3 tc=4 tc_inv=5 sreq=6 sreq_inv=7
6-5:2	slave_ch	Slave mode channel connected to the pin.	
4-2:3	master_mode	Pin usage when in master mode. off: The output is off, and the pin can be used as input as_slave: The pin maintains its slave mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low dack: Used as DMA acknowledge, active high dack_inv: Used as DMA acknowledge, active low	off=0 as_slave=1 lo=2 hi=3 tc=4 tc_inv=5 dack=6 dack_inv=7
1-0:2	master_ch	External DMA channel connected to the pin in master mode.	

25.5.32 rw_pin7_cfg

Address	0xbc
Default	0x00000000
Type	Read/Write
Description	Configuration register for external DMA/slave mode handshake pin **hsh7** output.

Bit(s)	Name	Description	Value
9-7:3	slave_mode	Pin usage when in slave mode. off: The output is off, and the pin can be used as input as_master: The pin maintains its master mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low sreq: Used as slave request, active high sreq_inv: Used as slave request, active low	off=0 as_master=1 lo=2 hi=3 tc=4 tc_inv=5 sreq=6 sreq_inv=7
6-5:2	slave_ch	Slave mode channel connected to the pin.	
4-2:3	master_mode	Pin usage when in master mode. off: The output is off, and the pin can be used as input as_slave: The pin maintains its slave mode function lo: Driven low hi: Driven high tc: Used as tc_out, active high tc_inv: Used as tc_out, active low dack: Used as DMA acknowledge, active high dack_inv: Used as DMA acknowledge, active low	off=0 as_slave=1 lo=2 hi=3 tc=4 tc_inv=5 dack=6 dack_inv=7
1-0:2	master_ch	External DMA channel connected to the pin in master mode.	

25.5.33 r_pin_stat

Address	0xc0
Default	
Type	Read
Description	Input values on the external DMA/slave mode handshake pins.

Bit(s)	Name	Description	Value
7	pin7	Value on pin **hsh7**	
6	pin6	Value on pin **hsh6**	
5	pin5	Value on pin **hsh5**	
4	pin4	Value on pin **hsh4**	
3	pin3	Value on pin **hsh3**	
2	pin2	Value on pin **hsh2**	
1	pin1	Value on pin **hsh1**	
0	pin0	Value on pin **hsh0**	

25.6 bif_slave

Instance	Base Address
bif_slave	0xb0018000

25.6.1 rw_slave_cfg

Address	0x0
Default	0x00000000
Type	Read/Write
Description	General slave mode configuration register.

Bit(s)	Name	Description	Value
6	dis	Disable the slave mode interface. no: Interface enabled yes: Interface disabled	no=0 yes=1
5	loopback	In loopback mode, the master can access its own external slave mode registers. This mode is mainly intended for production test. no: Loopback disabled yes: Loopback enabled	no=0 yes=1
4	boot_rdy	This field is set by the boot code to indicate that there is a boot record ready in the internal RAM of the slave. Used by master and slave boot methods. no: Boot record not ready yes: Boot record ready	no=0 yes=1
3	use_slave_id	Select if the slave ID address will be used or ignored. no: Slave ID address is ignored yes: Slave ID address is used	no=0 yes=1
2-0:3	slave_id	Slave ID address. If use_slave_id is set to yes, the slave ID address is compared against address bits 8 - 6 to qualify an access to the external slave mode registers.	

25.6.2 r_slave_mode

Address	0x4
Default	
Type	Read
Description	This register shows the access mode for each slave channel.

Bit(s)	Name	Description	Value
3	ch3_mode	Access mode for slave channel 3. addr: Address register dma: DMA channel	addr=0 dma=1
2	ch2_mode	Access mode for slave channel 2. addr: Address register dma: DMA channel	addr=0 dma=1
1	ch1_mode	Access mode for slave channel 1. addr: Address register dma: DMA channel	addr=0 dma=1
0	ch0_mode	Access mode for slave channel 0. addr: Address register dma: DMA channel	addr=0 dma=1

25.6.3 rw_ch0_cfg

Address	0x10
Default	0x00000000
Type	Read/Write
Description	Configuration register for slave mode channel 0.

Bit(s)	Name	Description	Value
5-4:2	data_cs	Enable the use of a separate chip select input (**csp0_n**) for the sequential data register. Note that even if this is enabled, the sequential data register can still be reached via the **css_n** chip select. no: No separate chip select active_lo: **csp0_n** pin used, active low active_hi: **csp0_n** pin used, active high	no=0 active_lo=2 active_hi=3
3	access_ctrl	Selects if the access mode for the channel is controlled by the current bus master via the external slave mode registers, or by the access_mode field. master: Access mode controlled by bus master mode_reg: Access mode controlled by mode register	master=0 mode_reg=1
2	access_mode	Selects the access mode for the channel. When read, it only reads back the register value. The real access mode for the channel is read from the ch0_mode field of the r_slave_mode register. addr: Address register dma: DMA channel	addr=0 dma=1
1-0:2	rd_hold	Sets the data hold time after a read from the external slave mode registers. 10 ns will be needed for the fastest burst mode. 0 ns can be chosen if the application requires faster data turn off time. The data hold time can also be set by the bus master via the external slave mode cfg register rw_ch0_ctrl. master: Data hold time set by bus master t10ns: 10 ns data hold time t0ns: 0 ns data hold time	master=0 t10ns=2 t0ns=3

25.6.4 rw_ch1_cfg

Address	0x14
Default	0x00000000
Type	Read/Write
Description	Configuration register for slave mode channel 1.

Bit(s)	Name	Description	Value
5-4:2	data_cs	Enable the use of a separate chip select input (**csp1_n**) for the sequential data register. Note that even if this is enabled, the sequential data register can still be reached via the **css_n** chip select. no: No separate chip select active_lo: **csp1_n** pin used, active low active_hi: **csp1_n** pin used, active high	no=0 active_lo=2 active_hi=3
3	access_ctrl	Selects if the access mode for the channel is controlled by the current bus master via the external slave mode registers, or by the access_mode field. master: Access mode controlled by bus master mode_reg: Access mode controlled by mode register	master=0 mode_reg=1
2	access_mode	Selects the access mode for the channel. When read, it only reads back the register value. The real access mode for the channel is read from the ch1_mode field of the r_slave_mode register. addr: Address register dma: DMA channel	addr=0 dma=1
1-0:2	rd_hold	Sets the data hold time after a read from the external slave mode registers. 10 ns will be needed for the fastest burst mode. 0 ns can be chosen if the application requires faster data turn off time. The data hold time can also be set by the bus master via the external slave mode cfg register rw_ch1_ctrl. master: Data hold time set by bus master t10ns: 10 ns data hold time t0ns: 0 ns data hold time	master=0 t10ns=2 t0ns=3

25.6.5 rw_ch2_cfg

Address	0x18
Default	0x00000000
Type	Read/Write
Description	Configuration register for slave mode channel 2.

Bit(s)	Name	Description	Value
5-4:2	data_cs	Enable the use of a separate chip select input (**csp2_n**) for the sequential data register. Note that even if this is enabled, the sequential data register can still be reached via the **css_n** chip select. no: No separate chip select active_lo: **csp2_n** pin used, active low active_hi: **csp2_n** pin used, active high	no=0 active_lo=2 active_hi=3
3	access_ctrl	Selects if the access mode for the channel is controlled by the current bus master via the external slave mode registers, or by the access_mode field. master: Access mode controlled by bus master mode_reg: Access mode controlled by mode register	master=0 mode_reg=1
2	access_mode	Selects the access mode for the channel. When read, it only reads back the register value. The real access mode for the channel is read from the ch2_mode field of the r_slave_mode register. addr: Address register dma: DMA channel	addr=0 dma=1
1-0:2	rd_hold	Sets the data hold time after a read from the external slave mode registers. 10 ns will be needed for the fastest burst mode. 0 ns can be chosen if the application requires faster data turn off time. The data hold time can also be set by the bus master via the external slave mode cfg register rw_ch2_ctrl. master: Data hold time set by bus master t10ns: 10 ns data hold time t0ns: 0 ns data hold time	master=0 t10ns=2 t0ns=3

25.6.6 rw_ch3_cfg

Address	0x1c
Default	0x00000000
Type	Read/Write
Description	Configuration register for slave mode channel 3.

Bit(s)	Name	Description	Value
5-4:2	data_cs	Enable the use of a separate chip select input (**csp3_n**) for the sequential data register. Note that even if this is enabled, the sequential data register can still be reached via the **css_n** chip select. no: No separate chip select active_lo: **csp3_n** pin used, active low active_hi: **csp3_n** pin used, active high	no=0 active_lo=2 active_hi=3
3	access_ctrl	Selects if the access mode for the channel is controlled by the current bus master via the external slave mode registers, or by the access_mode field. master: Access mode controlled by bus master mode_reg: Access mode controlled by mode register	master=0 mode_reg=1
2	access_mode	Selects the access mode for the channel. When read, it only reads back the register value. The real access mode for the channel is read from the ch3_mode field of the r_slave_mode register. addr: Address register dma: DMA channel	addr=0 dma=1
1-0:2	rd_hold	Sets the data hold time after a read from the external slave mode registers. 10 ns will be needed for the fastest burst mode. 0 ns can be chosen if the application requires faster data turn off time. The data hold time can also be set by the bus master via the external slave mode cfg register rw_ch3_ctrl. master: Data hold time set by bus master t10ns: 10 ns data hold time t0ns: 0 ns data hold time	master=0 t10ns=2 t0ns=3

25.6.7 rw_arb_cfg

Address	0x20
Default	0x00000000
Type	Read/Write
Description	Configuration register for external bus arbitration.

Bit(s)	Name	Description	Value
12	dram_ctrl	Selects single master DRAM control or shared DRAM control. shared: DRAM control shared between bus masters local: DRAM control not shared	shared=0 local=1
11-10:2	settle_time	Selects the settle time for the bus arbitration. t50ns: 50 ns settle time t40ns: 40 ns settle time t30ns: 30 ns settle time t20ns: 20 ns settle time	t50ns=0 t40ns=1 t30ns=2 t20ns=3
9	acquire	Bus acquirement policy. Selects if the bus interface always will try to acquire the bus, of if it will only try it when it requires access to the external bus. on_access: Only when access to the external bus is required always: Always try to acquire the external bus	on_access=0 always=1
8-7:2	release	Bus release policy. Selects when the bus may be released to another bus master. no: The bus is never released at_idle: Release when the bus interface becomes idle burst_end: Release at the end of each bus burst	no=0 at_idle=2 burst_end=3
6-4:3	bg_mode	Operating mode for the **bg** pin. norm: Normal operation (active high) inv: Inverted operation (active low) lo: Constantly driven low hi: Constantly driven high z: Constantly high-z	norm=0 inv=1 lo=2 hi=3 z=4
3-1:3	brout_mode	Operating mode for the **brout** pin. norm: Normal operation (active high) inv: Inverted operation (active low) lo: Constantly driven low hi: Constantly driven high z: Constantly high-z	norm=0 inv=1 lo=2 hi=3 z=4
0	brin_mode	Operating mode for the **brin** pin. norm: Normal operation (active high) inv: Inverted operation (active low)	norm=0 inv=1

25.6.8 r_arb_stat

Address	0x24
Default	
Type	Read
Description	Bus arbitration status register.

Bit(s)	Name	Description	Value
4	bg	Shows the value on the **bg** pin.	
3	brout	Shows the value on the **brout** pin.	
2	brin	shows the value on the **brin** pin.	
1	mode	Shows the current operation mode. master: Master mode slave: Slave mode	master=0 slave=1
0	init_mode	Shows the initial operation mode after system reset. master: Master mode slave: Slave mode	master=0 slave=1

25.6.9 rw_intr_mask

Address	0x40
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts from the external bus arbitration. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
1	bus_acquire	Enable/disable bus_acquire interrupt. Interrupt when the bus has been acquired. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	bus_release	Enable/disable bus_release interrupt. Interrupt when the bus has been released. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.6.10 rw_ack_intr

Address	0x44
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts from the external bus arbitration.

Bit(s)	Name	Description	Value
1	bus_acquire	Acknowledge bus_acquire interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	bus_release	Acknowledge bus_release interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.6.11 r_intr

Address	0x48
Default	
Type	Read
Description	Interrupts before the mask. Interrupts from the external bus arbitration. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
1	bus_acquire	Interrupt bus_acquire active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	bus_release	Interrupt bus_release active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.6.12 r_masked_intr

Address	0x4c
Default	
Type	Read
Description	Interrupts after the mask. Interrupts from the external bus arbitration. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
1	bus_acquire	Interrupt bus_acquire active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	bus_release	Interrupt bus_release active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.7 bif_slave_ext

Instance	Base Address
bif_slave_ext	0xac000000

25.7.1 r_ch0_seq_data

Address	0x0
Default	
Type	Read
Description	This register is used for sequential data access through the slave mode interface. An access to this register will increment the address register of the channel by 4 if the channel is configured for address increment. A read to the register will initiate a new internal read within the slave. Note: This register may only be read when the dav field in the r_ch0_stat register is set to yes.

Bit(s)	Name	Description	Value
31-0:32	data	Sequential data.	

25.7.2 r_ch0_data

Address	0x4
Default	
Type	Read
Description	This register is used for single data access through the slave mode interface. An access to this register will not affect the address register of the channel, and will not generate a new read internally in the slave.

Bit(s)	Name	Description	Value
31-0:32	data	Data.	

25.7.3 rw_ch0_addr

Address	0x8
Default	
Type	Read/Write
Description	This register sets the address that will be used for the internal data access in the slave. The address increments with 4 for each access to the r_ch0_seq_data register if the channel is configured for address increment. (If address increment is selected, the address increments even if the channel is in DMA mode). A write to the register initiates an internal read to the address within the slave. Bit 31 is used to select snooped (bit 31 == 0) or non-snooped (bit 31 == 1) accesses. Bit 1 and 0 in the register are not used by the internal access. Note: This register may only be written when the dav field in the r_ch0_stat register is set to yes. When the channel is set to DMA mode, the register may be written at any time after the first time dav is set, or immediately if the dav field was set to yes before entering the DMA mode.

Bit(s)	Name	Description	Value
31-0:32	addr	Address.	

25.7.4 r_ch0_stat

Address	0xc
Default	0x00000001
Type	Read
Description	Channel status.

Bit(s)	Name	Description	Value
31-4:28	Reserved	Reserved.	
3	boot	This field reflects the value of the boot_rdy field in the internal mode register rw_slave_cfg. This field is duplicated in the status registers for all four channels.	
2	access_mode	Shows if the channel is using DMA or the channel address register. The value in rw_ch0_ctrl can be overridden by an internal mode register. This field shows the actual access mode. addr: Address register dma: DMA channel	addr=0 dma=1
1	addr_incr	Shows whether address increment is used or not. no: No address increment yes: Use address increment	no=0 yes=1
0	dav	Data busy/ready signal. This field indicates if the data in the data registers for the channel is valid. The address register of the channel may only be written when the field is set to yes. When the channel is in DMA mode, the address register may be written at any time after the first time the field is set to yes, or immediately if the field was set to yes before entering the DMA mode. The sequential data register of the channel may only be read when the field is set to yes. The field is set to yes at reset even though there is no valid data at that time. no: Busy yes: Data available	no=0 yes=1

25.7.5 rw_ch0_ctrl

Address	0xc
Default	0x00000000
Type	Read/Write
Description	

Bit(s)	Name	Description	Value
31-3:29	Reserved	Reserved.	
2	access_mode	Selects if the channel is using DMA or the channel address register. The value can be overridden by an internal mode register. addr: Address register dma: DMA channel	addr=0 dma=1
1	addr_incr	Selects whether to use address increment or not. no: No address increment yes: Use address increment	no=0 yes=1
0	rd_hold	Sets the minimum data hold time after a read to the external slave mode registers. 10 ns will be needed for the fastest burst mode. 0 ns can be chosen if the application requires faster data turn off time. t10ns: 10 ns data hold time t0ns: 0 ns data hold time	t10ns=0 t0ns=1

25.7.6 rw_ch1_seq_data

Address	0x10
Default	
Type	Read/Write
Description	This register is used for sequential data access through the slave mode interface. An access to this register will increment the address register of the channel by 4 if the channel is configured for address increment. Note: This register may only be written when the data_rdy field in the r_ch1_stat register is set to yes.

Bit(s)	Name	Description	Value
31-0:32	data	Sequential data.	

25.7.7 rw_ch1_data

Address	0x14
Default	
Type	Read/Write
Description	This register is used for single data access through the slave mode interface. An access to this register will not affect the address register of the channel. Note: This register may only be written when the data_rdy field in the r_ch1_stat register is set to yes.

Bit(s)	Name	Description	Value
31-0:32	data	Data.	

25.7.8 rw_ch1_addr

Address	0x18
Default	
Type	Read/Write
Description	This register sets the address that will be used for the internal data access in the slave. The address increments with 4 for each access to the rw_ch1_seq_data register if the channel is configured for address increment. (If address increment is selected, the address increments even if the channel is in DMA mode). Bit 31 is used to select snooped (bit 31 == 0) or non-snooped (bit 31 == 1) accesses. Bit 1 and 0 in the register are not used by the internal access. Note: This register may only be written when the data_rdy field in the r_ch1_stat register is set to yes. When the channel is in DMA mode, the register may be written at any time after the first time data_rdy is set, or immediately if the data_rdy field was set to yes before entering the DMA mode.

Bit(s)	Name	Description	Value
31-0:32	addr	Address.	

25.7.9 rw_ch1_ctrl

Address	0x1c
Default	0x00000000
Type	Read/Write
Description	

Bit(s)	Name	Description	Value
31-3:29	Reserved	Reserved.	
2	access_mode	Selects if the channel is using DMA or the channel address register. The value can be overridden by an internal mode register. addr: Address register dma: DMA channel	addr=0 dma=1
1	addr_incr	Selects whether to use address increment or not. no: No address increment yes: Use address increment	no=0 yes=1
0	rd_hold	Sets the minimum data hold time after a read to the external slave mode registers. 10 ns will be needed for the fastest burst mode. 0 ns can be chosen if the application requires faster data turn off time. t10ns: 10 ns data hold time t0ns: 0 ns data hold time	t10ns=0 t0ns=1

25.7.10 r_ch1_stat

Address	0x1c
Default	0x00000001
Type	Read
Description	Channel status.

Bit(s)	Name	Description	Value
31-4:28	Reserved	Reserved.	
3	boot	This field reflects the value of the boot_rdy field in the internal mode register rw_slave_cfg. This field is duplicated in the status registers for all four channels.	
2	access_mode	Shows if the channel is using DMA or the channel address register. The value in rw_ch1_ctrl can be overridden by an internal mode register. This field shows the actual access mode. addr: Address register dma: DMA channel	addr=0 dma=1
1	addr_incr	Shows whether address increment is used or not. no: No address increment yes: Use address increment	no=0 yes=1
0	data_rdy	Data busy/ready signal. The data registers and the address register of the channel may only be written when this field is set to yes. When the channel is in DMA mode, the address register may be written at any time after the first time the field is set to yes, or immediately if the field was set to yes before entering the DMA mode. no: Busy yes: Data ready	no=0 yes=1

25.7.11 r_ch2_seq_data

Address	0x20
Default	
Type	Read
Description	This register is used for sequential data access through the slave mode interface. An access to this register will increment the address register of the channel by 4 if the channel is configured for address increment. A read to the register will initiate a new internal read within the slave. Note: This register may only be read when the dav field in the r_ch2_stat register is set to yes.

Bit(s)	Name	Description	Value
31-0:32	data	Sequential data.	

25.7.12 r_ch2_data

Address	0x24
Default	
Type	Read
Description	This register is used for single data access through the slave mode interface. An access to this register will not affect the address register of the channel, and will not generate a new read internally in the slave.

Bit(s)	Name	Description	Value
31-0:32	data	Data.	

25.7.13 rw_ch2_addr

Address	0x28
Default	
Type	Read/Write
Description	This register sets the address that will be used for the internal data access in the slave. The address increments with 4 for each access to the r_ch2_seq_data register if the channel is configured for address increment. (If address increment is selected, the address increments even if the channel is in DMA mode). A write to the register initiates an internal read to the address within the slave. Bit 31 is used to select snooped (bit 31 == 0) or non-snooped (bit 31 == 1) accesses. Bit 1 and 0 in the register are not used by the internal access. Note: This register may only be written when the dav field in the r_ch2_stat register is set to yes. When the channel is set to DMA mode, the register may be written at any time after the first time dav is set, or immediately if the dav field was set to yes before entering the DMA mode.

Bit(s)	Name	Description	Value
31-0:32	addr	Address.	

25.7.14 r_ch2_stat

Address	0x2c
Default	0x00000001
Type	Read
Description	Channel status.

Bit(s)	Name	Description	Value
31-4:28	Reserved	Reserved.	
3	boot	This field reflects the value of the boot_rdy field in the internal mode register rw_slave_cfg. This field is duplicated in the status registers for all four channels.	
2	access_mode	Shows if the channel is using DMA or the channel address register. The value in rw_ch2_ctrl can be overridden by an internal mode register. This field shows the actual access mode. addr: Address register dma: DMA channel	addr=0 dma=1
1	addr_incr	Shows whether address increment is used or not. no: No address increment yes: Use address increment	no=0 yes=1
0	dav	Data busy/ready signal. This field indicates if the data in the data registers for the channel is valid. The address register of the channel may only be written when the field is set to yes. When the channel is in DMA mode, the address register may be written at any time after the first time the field is set to yes, or immediately if the field was set to yes before entering the DMA mode. The sequential data register of the channel may only be read when the field is set to yes. The field is set to yes at reset even though there is no valid data at that time. no: Busy yes: Data available	no=0 yes=1

25.7.15 rw_ch2_ctrl

Address	0x2c
Default	0x00000000
Type	Read/Write
Description	

Bit(s)	Name	Description	Value
31-3:29	Reserved	Reserved.	
2	access_mode	Selects if the channel is using DMA or the channel address register. The value can be overridden by an internal mode register. addr: Address register dma: DMA channel	addr=0 dma=1
1	addr_incr	Selects whether to use address increment or not. no: No address increment yes: Use address increment	no=0 yes=1
0	rd_hold	Sets the minimum data hold time after a read to the external slave mode registers. 10 ns will be needed for the fastest burst mode. 0 ns can be chosen if the application requires faster data turn off time. t10ns: 10 ns data hold time t0ns: 0 ns data hold time	t10ns=0 t0ns=1

25.7.16 rw_ch3_seq_data

Address	0x30
Default	
Type	Read/Write
Description	This register is used for sequential data access through the slave mode interface. An access to this register will increment the address register of the channel by 4 if the channel is configured for address increment. Note: This register may only be written when the data_rdy field in the r_ch3_stat register is set to yes.

Bit(s)	Name	Description	Value
31-0:32	data	Sequential data.	

25.7.17 rw_ch3_data

Address	0x34
Default	
Type	Read/Write
Description	This register is used for single data access through the slave mode interface. An access to this register will not affect the address register of the channel. Note: This register may only be written when the data_rdy field in the r_ch3_stat register is set to yes.

Bit(s)	Name	Description	Value
31-0:32	data	Data.	

25.7.18 rw_ch3_addr

Address	0x38
Default	
Type	Read/Write
Description	This register sets the address that will be used for the internal data access in the slave. The address increments with 4 for each access to the rw_ch3_seq_data register if the channel is configured for address increment. (If address increment is selected, the address increments even if the channel is in DMA mode). Bit 31 is used to select snooped (bit 31 == 0) or non-snooped (bit 31 == 1) accesses. Bit 1 and 0 in the register are not used by the internal access. Note: This register may only be written when the data_rdy field in the r_ch3_stat register is set to yes. When the channel is in DMA mode, the register may be written at any time after the first time data_rdy is set, or immediately if the data_rdy field was set to yes before entering the DMA mode.

Bit(s)	Name	Description	Value
31-0:32	addr	Address.	

25.7.19 r_ch3_stat

Address	0x3c
Default	0x00000001
Type	Read
Description	Channel status.

Bit(s)	Name	Description	Value
31-4:28	Reserved	Reserved.	
3	boot	This field reflects the value of the boot_rdy field in the internal mode register rw_slave_cfg. This field is duplicated in the status registers for all four channels.	
2	access_mode	Shows if the channel is using DMA or the channel address register. The value in rw_ch3_ctrl can be overridden by an internal mode register. This field shows the actual access mode. addr: Address register dma: DMA channel	addr=0 dma=1
1	addr_incr	Shows whether address increment is used or not. no: No address increment yes: Use address increment	no=0 yes=1
0	data_rdy	Data busy/ready signal. The data registers and the address register of the channel may only be written when this field is set to yes. When the channel is in DMA mode, the address register may be written at any time after the first time the field is set to yes, or immediately if the field was set to yes before entering the DMA mode. no: Busy yes: Data ready	no=0 yes=1

25.7.20 rw_ch3_ctrl

Address	0x3c
Default	0x00000000
Type	Read/Write
Description	

Bit(s)	Name	Description	Value
31-3:29	Reserved	Reserved.	
2	access_mode	Selects if the channel is using DMA or the channel address register. The value can be overridden by an internal mode register. addr: Address register dma: DMA channel	addr=0 dma=1
1	addr_incr	Selects whether to use address increment or not. no: No address increment yes: Use address increment	no=0 yes=1
0	rd_hold	Sets the minimum data hold time after a read to the external slave mode registers. 10 ns will be needed for the fastest burst mode. 0 ns can be chosen if the application requires faster data turn off time. t10ns: 10 ns data hold time t0ns: 0 ns data hold time	t10ns=0 t0ns=1

25.8 config

Instance	Base Address
config	0xb003c000

25.8.1 r_bootsel

Address	0x0
Default	
Type	Read
Description	Chip boot select register. This register contains the value sampled on pins **bs6** - **bs0** when **rst_n** is released.

Bit(s)	Name	Description	Value
6	flash_bw	Bus width for EPROM/Flash PROM select 0 and 1. bw16: 16 bit bus width bw32: 32 bit bus width	bw16=0 bw32=1
5	pll	PLL mode. no: PLL off (bypass mode) yes: PLL on	no=0 yes=1
4	user	User defined boot mode.	
3	full_duplex	Half/full duplex select for network boot. no: Half duplex yes: Full duplex	no=0 yes=1
2-0:3	boot_mode	Boot mode. nor: NOR flash boot net_rx: Network RX boot net_tx_rx: Network TX/RX boot nand: NAND flash boot ser: Serial flash boot master: Master booted by slave slave: Slave booted by master none: No boot	nor=0 net_rx=1 net_tx_rx=2 nand=3 ser=4 master=5 slave=6 none=7

25.8.2 rw_clk_ctrl

Address	0x4
Default	0x00000002
Type	Read/Write
Description	Global clock control register. This register is used to turn on/off different on-chip clock regions. By default only the CPU and essential system functionality to access the internal boot ROM are enabled after reset. Depending on which boot mode that is used, different blocks will be turned on by the boot code.

Bit(s)	Name	Description	Value
9	fix_io	Fixed function I/O block 100 MHz clock region control. This bit controls the synchronous serial ports 0 and 1, asynchronous serial ports 0 to 3, ATA port and Ethernet port 1. no: Turn off clock region yes: Turn on clock region	no=0 yes=1
8	bif	External bus interface 100 MHz clock region control. no: Turn off clock region yes: Turn on clock region	no=0 yes=1
7	dma89_strcop	DMA channel 8, channel 9 and crypto accelerator 100MHz clock regions control. no: Turn off clock region yes: Turn on clock region	no=0 yes=1
6	dma67	DMA channels 6 and 7 100 MHz clock region control. no: Turn off clock region yes: Turn on clock region	no=0 yes=1
5	dma45	DMA channels 4 and 5 100 MHz clock region control. no: Turn off clock region yes: Turn on clock region	no=0 yes=1
4	dma23	DMA channels 2 and 3 100 MHz clock region control. no: Turn off clock region yes: Turn on clock region	no=0 yes=1
3	dma01_eth0	DMA channel 0, channel 1 and Ethernet port 0 100 MHz clock region control. no: Turn off clock region yes: Turn on clock region	no=0 yes=1
2	iop	I/O processor 200 MHz clock region control. no: Turn off clock region yes: Turn on clock region	no=0 yes=1
1	cpu	CPU 200 MHz clock region control. When turned off the CPU clock region including MMUs, caches and debug support is turned off. no: Turn off clock region yes: Turn on clock region	no=0 yes=1

0	pll	PLL operation control. This field controls if the PLL is enabled or if it is turned off and bypassed. When the PLL is turned off the internal 400 MHz clock is reduced to 12 MHz. This will result in that all 200 MHz clock regions will be clocked at 6 MHz and all 100 MHz regions at 3 MHz. no: PLL turned off yes: PLL turned on	no=0 yes=1

25.8.3 rw_pad_ctrl

Address	0x8
Default	0x00000000
Type	Read/Write
Description	Global pad control.

Bit(s)	Name	Description	Value
1	phyrst_n	Controls the **phyrst_n** external reset output.	
0	usb_susp	USB transceiver suspend control. no: Set transceiver in normal mode yes: Set transceiver in suspend mode	no=0 yes=1

25.9 cris

Bank
0

25.9.1 rw_gc_cfg

Support register	s0
Default	0x00000000
Type	Read/Write
Description	General configuration register.

Bit(s)	Name	Description	Value
6	gp	Makes gc_r0 through gc_r3 writable only from Guru mode.Once set, this bit may only be cleared from Guru mode. no: gc_r0 through gc_r3 writable from Guru and Kernel mode yes: gc_r0 through gc_r3 only writable from Guru mode	no=0 yes=1
5	gk	Makes the break 15 instruction cause a Guru mode exception instead of a normal break 15 exception (vector 0x1f). Once set, this bit may only be cleared from Guru mode. no: Break 15 exception yes: Guru mode exception	no=0 yes=1
4	gb	Makes hardware breakpoints and single step exceptions cause a Guru mode exception instead of a breakpoint/single step exception (vector 0xC/0x3). Once set, this bit may only be cleared from Guru mode. no: Breakpoint/single step exceptions yes: Guru mode exceptions	no=0 yes=1
3	dm	Enable data MMU. no: Disable yes: Enable	no=0 yes=1
2	im	Enable instruction MMU. no: Disable yes: Enable	no=0 yes=1
1	dc	Enable data cache. The cache is normally enabled by the boot ROM. no: Disable yes: Enable	no=0 yes=1
0	ic	Enable instruction cache. The cache is normally enabled by the boot ROM. no: Disable yes: Enable	no=0 yes=1

25.9.2 rw_gc_ccs

Support register	s1
Default	
Type	Read/Write
Description	Guru mode saved CCS.

25.9.3 rw_gc_srs

Support register	s2
Default	
Type	Read/Write
Description	Guru mode saved SRS.

Bit(s)	Name	Description	Value
7-0:8	srs	Guru mode saved SRS.	

25.9.4 rw_gc_nrp

Support register	s3
Default	
Type	Read/Write
Description	Guru mode saved NRP.

25.9.5 rw_gc_exs

Support register	s4
Default	
Type	Read/Write
Description	Guru mode Exception Status.

25.9.6 rw_gc_eda

Support register	s5
Default	
Type	Read/Write
Description	Guru mode Exception Data Address.

25.9.7 rw_gc_r0

Support register	s8
Default	
Type	Read/Write
Description	Guru mode scratchpad register 0.

25.9.8 rw_gc_r1

Support register	s9
Default	
Type	Read/Write
Description	Guru mode scratchpad register 1.

25.9.9 rw_gc_r2

Support register	s10
Default	
Type	Read/Write
Description	Guru mode scratchpad register 2.

25.9.10 rw_gc_r3

Support register	s11
Default	
Type	Read/Write
Description	Guru mode scratchpad register 3.

25.10 cris_bp

Bank
3

25.10.1 rw_bp_ctrl

Support register	s0
Default	0x00000000
Type	Read/Write
Description	Breakpoint control register.

Bit(s)	Name	Description	Value
25	d5_wpwr	Enable/disable data watchpoint 5 for write accesses. yes: Enable no: Disable	yes=1 no=0
24	d5_wprd	Enable/disable data watchpoint 5 for read accesses. yes: Enable no: Disable	yes=1 no=0
23	d5_bpwr	Enable/disable data breakpoint 5 for write accesses. yes: Enable no: Disable	yes=1 no=0
22	d5_bprd	Enable/disable data breakpoint 5 for read accesses. yes: Enable no: Disable	yes=1 no=0
21	d4_wpwr	Enable/disable data watchpoint 4 for write accesses. yes: Enable no: Disable	yes=1 no=0
20	d4_wprd	Enable/disable data watchpoint 4 for read accesses. yes: Enable no: Disable	yes=1 no=0
19	d4_bpwr	Enable/disable data breakpoint 4 for write accesses. yes: Enable no: Disable	yes=1 no=0
18	d4_bprd	Enable/disable data breakpoint 4 for read accesses. yes: Enable no: Disable	yes=1 no=0
17	d3_wpwr	Enable/disable data watchpoint 3 for write accesses. yes: Enable no: Disable	yes=1 no=0
16	d3_wprd	Enable/disable data watchpoint 3 for read accesses. yes: Enable no: Disable	yes=1 no=0

15	d3_bpwr	Enable/disable data breakpoint 3 for write accesses. yes: Enable no: Disable	yes=1 no=0
14	d3_bprd	Enable/disable data breakpoint 3 for read accesses. yes: Enable no: Disable	yes=1 no=0
13	d2_wpwr	Enable/disable data watchpoint 2 for write accesses. yes: Enable no: Disable	yes=1 no=0
12	d2_wprd	Enable/disable data watchpoint 2 for read accesses. yes: Enable no: Disable	yes=1 no=0
11	d2_bpwr	Enable/disable data breakpoint 2 for write accesses. yes: Enable no: Disable	yes=1 no=0
10	d2_bprd	Enable/disable data breakpoint 2 for read accesses. yes: Enable no: Disable	yes=1 no=0
9	d1_wpwr	Enable/disable data watchpoint 1 for write accesses. yes: Enable no: Disable	yes=1 no=0
8	d1_wprd	Enable/disable data watchpoint 1 for read accesses. yes: Enable no: Disable	yes=1 no=0
7	d1_bpwr	Enable/disable data breakpoint 1 for write accesses. yes: Enable no: Disable	yes=1 no=0
6	d1_bprd	Enable/disable data breakpoint 1 for read accesses. yes: Enable no: Disable	yes=1 no=0
5	d0_wpwr	Enable/disable data watchpoint 0 for write accesses. yes: Enable no: Disable	yes=1 no=0
4	d0_wprd	Enable/disable data watchpoint 0 for read accesses. yes: Enable no: Disable	yes=1 no=0
3	d0_bpwr	Enable/disable data breakpoint 0 for write accesses. yes: Enable no: Disable	yes=1 no=0
2	d0_bprd	Enable/disable data breakpoint 0 for read accesses. yes: Enable no: Disable	yes=1 no=0
1	i0_wp	Enable/disable instruction watchpoint 0. yes: Enable no: Disable	yes=1 no=0

0	i0_bp	Enable/disable instruction breakpoint 0. yes: Enable no: Disable	yes=1 no=0

25.10.2 rw_bp_i0_start

Support register	s1
Default	
Type	Read/Write
Description	Instruction break/watch-point 0 start address.

25.10.3 rw_bp_i0_end

Support register	s2
Default	
Type	Read/Write
Description	Instruction break/watch-point 0 end address.

25.10.4 rw_bp_d0_start

Support register	s3
Default	
Type	Read/Write
Description	Data break/watch-point 0 start address.

25.10.5 rw_bp_d0_end

Support register	s4
Default	
Type	Read/Write
Description	Data break/watch-point 0 end address.

25.10.6 rw_bp_d1_start

Support register	s5
Default	
Type	Read/Write
Description	Data break/watch-point 1 start address.

25.10.7 rw_bp_d1_end

Support register	s6
Default	
Type	Read/Write
Description	Data break/watch-point 1 end address.

25.10.8 rw_bp_d2_start

Support register	s7
Default	
Type	Read/Write
Description	Data break/watch-point 2 start address.

25.10.9 rw_bp_d2_end

Support register	s8
Default	
Type	Read/Write
Description	Data break/watch-point 2 end address.

25.10.10 rw_bp_d3_start

Support register	s9
Default	
Type	Read/Write
Description	Data break/watch-point 3 start address.

25.10.11 rw_bp_d3_end

Support register	s10
Default	
Type	Read/Write
Description	Data break/watch-point 3 end address.

25.10.12 rw_bp_d4_start

Support register	s11
Default	
Type	Read/Write
Description	Data break/watch-point 4 start address.

25.10.13 rw_bp_d4_end

Support register	s12
Default	
Type	Read/Write
Description	Data break/watch-point 4 end address.

25.10.14 rw_bp_d5_start

Support register	s13
Default	
Type	Read/Write
Description	Data break/watch-point 5 start address.

25.10.15 rw_bp_d5_end

Support register	s14
Default	
Type	Read/Write
Description	Data break/watch-point 5 end address.

25.11 dma

Instance	Base Address
dma0	0xb0000000
dma1	0xb0002000
dma2	0xb0004000
dma3	0xb0006000
dma4	0xb0008000
dma5	0xb000a000
dma6	0xb000c000
dma7	0xb000e000
dma8	0xb0010000
dma9	0xb0012000

25.11.1 rw_data

Address	0x0
Default	
Type	Read/Write
Description	Data descriptor. This register holds the pointer to the current data descriptor. Should normally not be used by SW. SW may use the value read from this register to observe list progression, but only if the corresponding context-descriptor is never re-read, eg. by an ack_pkt+restore stream command from the HW-client.

25.11.2 rw_data_next

Address	0x4
Default	
Type	Read/Write
Description	Data descriptor next. This register holds the pointer to the next data descriptor of the current data descriptor. Should normally not be used by SW.

25.11.3 rw_data_buf

Address	0x8
Default	
Type	Read/Write
Description	Data descriptor buffer. This register holds the pointer to the first byte of the data buffer of the current data descriptor. Should normally not be used by SW.

25.11.4 rw_data_ctrl

Address	0xc
Default	
Type	Read/Write
Description	Data descriptor control. This register holds the ctrl field of the current data descriptor. Should normally not be used by SW.

Bit(s)	Name	Description	Value
5	wait	Wait for command. Finish processing this buffer, then wait for command from the client. Out channel only. Ignored by in channel. yes: Wait enabled no: Wait disabled	yes=1 no=0
4	intr	Interrupt. Indicates if an interrupt is generated when this data descriptor has been processed and the next data descriptor is loaded. If eol is set, this field indicates if an interrupt is generated when this data descriptor has been processed. yes: Interrupt enabled no: Interrupt disabled	yes=1 no=0
3	out_eop	Out channel end of packet. This is the last data descriptor of the packet to be transmitted. Out channel only. Ignored by in channel. yes: End of packet no: Not end of packet	yes=1 no=0
0	eol	End of list. This is the last data descriptor of the list. yes: End of list no: Not end of list	yes=1 no=0

25.11.5 rw_data_stat

Address	0x10
Default	
Type	Read/Write
Description	Data descriptor status. This register holds the status field of the current data descriptor. Should normally not be used by SW.

Bit(s)	Name	Description	Value
3	in_eop	In channel end of packet. This is the last descriptor of the received packet. In channel only. Ignored by out channel. yes: End of packet no: Not end of packet	yes=1 no=0

25.11.6 rw_data_md

Address	0x14
Default	
Type	Read/Write
Description	Data descriptor meta data. This register holds the meta data of the current data descriptor. Should normally not be used by SW.

Bit(s)	Name	Description	Value
15-0:16	md	Meta data. Client read and write meta data area.	

25.11.7 rw_data_md_s

Address	0x18
Default	
Type	Read/Write
Description	Sampled data descriptor meta data. This register holds the meta data field 0 sampled from the client. Should normally not be used by SW.

Bit(s)	Name	Description	Value
15-0:16	md_s	Meta data. Sampled meta data area.	

25.11.8 rw_data_after

Address	0x1c
Default	
Type	Read/Write
Description	Data descriptor after. This register holds the pointer to the byte after the last byte of the data buffer. Should normally not be used by SW.

25.11.9 rw_ctxt

Address	0x20
Default	
Type	Read/Write
Description	Context descriptor. This register holds the pointer to the current context descriptor. Should normally not be used by SW.

25.11.10 rw_ctxt_next

Address	0x24
Default	
Type	Read/Write
Description	Context descriptor next. This register holds the pointer to the next context descriptor of the context descriptor list. Should normally not be used by SW.

25.11.11 rw_ctxt_ctrl

Address	0x28
Default	
Type	Read/Write
Description	Context control. This register holds the ctrl field of the current context descriptor. Should normally not be used by SW.

Bit(s)	Name	Description	Value
7	en	Enable. Indicates if this context descriptor is enabled. yes: Enabled no: Disabled	yes=1 no=0
6	store_mode	Store mode. Indicates if the context descriptor may be stored at any time during data transfer, or at wait_ack only. The buffer space of two consecutive buffers must be at least 64 bytes to guarantee full DMA performance, when a context descriptor is to be stored at any time. Out channel only. Ignored by in channel. anytime: The context descriptor may be saved at anytime only_at_wait: wait_ack is used to store the context descriptor	anytime=1 only_at_wait=0
4	intr	Interrupt. Indicates if an interrupt is generated when this context descriptor has been processed, and the next or n:th context descriptor is loaded. yes: Interrupt enabled no: Interrupt disabled	yes=1 no=0
0	eol	End of list. Indicates if this is the last context descriptor of the list. yes: End of list no: Not end of list	yes=1 no=0

25.11.12 rw_ctxt_stat

Address	0x2c
Default	
Type	Read/Write
Description	Context descriptor status. This register holds the status field of the current context descriptor. Should normally not be used by SW.

Bit(s)	Name	Description	Value
7	dis	Disabled. Indicates if this context descriptor is disabled by the DMA. The priority of this bit is higher than rw_ctxt_ctrl.en. yes: Disabled no: Not disabled	yes=1 no=0

25.11.13 rw_ctxt_md0

Address	0x30
Default	
Type	Read/Write
Description	Context meta data 0. This register holds meta data of the current context descriptor. Should normally not be used by SW.

Bit(s)	Name	Description	Value
15-0:16	md0	Meta data. Client read and write meta data area.	

25.11.14 rw_ctxt_md0_s

Address	0x34
Default	
Type	Read/Write
Description	Sampled context meta data 0. This register holds the meta data field 0 sampled from the client. Should normally not be used by SW.

Bit(s)	Name	Description	Value
15-0:16	md0_s	Sampled meta data area.	

25.11.15 rw_ctxt_md1

Address	0x38
Default	
Type	Read/Write
Description	Context meta data 1. This register holds meta data of the current context descriptor. Should normally not be used by SW.

25.11.16 rw_ctxt_md1_s

Address	0x3c
Default	
Type	Read/Write
Description	Sampled context meta data 1. This register holds the meta data field 1 sampled from the client. Should normally not be used by SW.

25.11.17 rw_ctxt_md2

Address	0x40
Default	
Type	Read/Write
Description	Context meta data 2. This register holds meta data of the current context descriptor. Should normally not be used by SW.

25.11.18 rw_ctxt_md2_s

Address	0x44
Default	
Type	Read/Write
Description	Sampled context meta data 2. This register holds the meta data field 2 sampled from the client. Should normally not be used by SW.

25.11.19 rw_ctxt_md3

Address	0x48
Default	
Type	Read/Write
Description	Context meta data 3. This register holds meta data of the current context descriptor. Should normally not be used by SW.

25.11.20 rw_ctxt_md3_s

Address	0x4c
Default	
Type	Read/Write
Description	Sampled context meta data 3. This register holds the meta data field 3 sampled from the client. Should normally not be used by SW.

25.11.21 rw_ctxt_md4

Address	0x50
Default	
Type	Read/Write
Description	Context meta data 4. This register holds meta data of the current context descriptor. Should normally not be used by SW.

25.11.22 rw_ctxt_md4_s

Address	0x54
Default	
Type	Read/Write
Description	Sampled context meta data 4. This register holds the meta data field 4 sampled from the client. Should normally not be used by SW.

25.11.23 rw_saved_data

Address	0x58
Default	
Type	Read/Write
Description	Saved data descriptor. This register holds the pointer to the last saved data descriptor of the current context descriptor. Should normally not be used by SW.

25.11.24 rw_saved_data_buf

Address	0x5c
Default	
Type	Read/Write
Description	Saved data descriptor buffer. This register holds the pointer to the last saved buffer position of the current context descriptor. Should normally not be used by SW.

25.11.25 rw_group

Address	0x60
Default	
Type	Read/Write
Description	Group descriptor. This register holds the pointer to the current group descriptor.

25.11.26 rw_group_next

Address	0x64
Default	
Type	Read/Write
Description	Group descriptor next. This register holds the pointer to the next group descriptor at the same level of the group descriptor hierarchy. Should normally not be used by SW.

25.11.27 rw_group_ctrl

Address	0x68
Default	
Type	Read/Write
Description	Group descriptor control. This register holds the ctrl field of the current group descriptor. Should normally not be used by SW.

Bit(s)	Name	Description	Value
7	en	Enable. Indicates if this group descriptor is enabled. yes: Enabled no: Disabled	yes=1 no=0
4	intr	Interrupt. Indicates if an interrupt is generated when this group descriptor has been processed, and the next, upper or lower group descriptor is loaded. yes: Interrupt enabled no: Interrupt disabled	yes=1 no=0
2	bol	Bottom of list. Indicates if the current group descriptor is at the lowest level. yes: Bottom of list no: Not bottom of list	yes=1 no=0
1	tol	Top of list. Indicates if the current group descriptor is at the highest level. yes: Top of list no: Not top of list	yes=1 no=0
0	eol	End of list. Indicates if this is the last group descriptor of the list. yes: End of list no: Not end of list	yes=1 no=0

25.11.28 rw_group_stat

Address	0x6c
Default	
Type	Read/Write
Description	Group descriptor status. This register holds the status field of the current group descriptor. Should normally not be used by SW.

Bit(s)	Name	Description	Value
7	dis	Disable. Indicates if this group descriptor is disabled by the DMA. The priority of this bit is higher than rw_group_ctrl.en. yes: Disabled no: Not disabled	yes=1 no=0

25.11.29 rw_group_md

Address	0x70
Default	
Type	Read/Write
Description	Group descriptor meta data. This register holds the meta data of the current group descriptor. Should normally not be used by SW.

Bit(s)	Name	Description	Value
15-0:16	md	Meta data. Client read and write meta data area.	

25.11.30 rw_group_md_s

Address	0x74
Default	
Type	Read/Write
Description	Sampled group meta data. This register holds the group meta data sampled from the client. Should normally not be used by SW.

Bit(s)	Name	Description	Value
15-0:16	md_s	Sampled meta data.	

25.11.31 rw_group_up

Address	0x78
Default	
Type	Read/Write
Description	Group descriptor up. This register holds the up pointer of the current group descriptor. Should normally not be used by SW.

25.11.32 rw_group_down

Address	0x7c
Default	
Type	Read/Write
Description	Group descriptor down. This register holds the down pointer of the current group descriptor.

25.11.33 rw_cmd

Address	0x80
Default	0x00000000
Type	Read/Write
Description	Command. The DMA channel command register.

Bit(s)	Name	Description	Value
0	cont_data	Continue data descriptor processing. If the DMA has reached the end of a data descriptor list, it reloads the last data descriptor to check if the end of list has been removed. If so the data processing continues. yes: Continue data list processing no: No operation	yes=1 no=0

25.11.34 rw_cfg

Address	0x84
Default	0x00000000
Type	Read/Write
Description	Configuration. The DMA channel configuration register.

Bit(s)	Name	Description	Value
1	stop	Stop DMA channel. When set the DMA channel completes any ongoing memory operation and then stops. When cleared the DMA channel continues its normal function. yes: Stop the DMA channel no: Normal operation	yes=1 no=0
0	en	Enable. Enable DMA channel. When set the DMA channel performs its normal function. When cleared the DMA channel is reset, and remains in this state until enable is set. yes: Enable the DMA channel no: Disable and reset the DMA channel	yes=1 no=0

25.11.35 rw_stat

Address	0x88
Default	0x00000101
Type	Read/Write
Description	The DMA channel status. Note that this register cannot be set by stream command set_reg or by SW since the DMA updates this register each cycle.

Bit(s)	Name	Description	Value
31-24:8	buf	The amount of data buffered in the DMA.	
15-8:8	stream_cmd_src	Stream command source. The source of the executed stream command. idle: No active command client: The client is source intern: Internal source sw: The SW is source	idle=1 client=2 intern=4 sw=8
7-5:3	list_state	The DMA data list status. data_at_eol: The DMA channel has finished the data list. Note that there still may be buffered data in the DMA	data_at_eol=1
4-0:5	mode	The DMA channel status mode of operation. rst: The DMA channel is reset stopped: The DMA channel has stopped running: The DMA channel is running	rst=1 stopped=2 running=4

25.11.36 rw_intr_mask

Address	0x8c
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. DMA channel interrupts. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	stream_cmd	Enable/disable stream_cmd interrupt. Stream command done interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	in_eop	Enable/disable in_eop interrupt. In channel end of packet interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	data	Enable/disable data interrupt. Data descriptor interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	ctxt	Enable/disable ctxt interrupt. Context descriptor interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	group	Enable/disable group interrupt. Group descriptor interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.11.37 rw_ack_intr

Address	0x90
Default	
Type	Read/Write
Description	Acknowledge interrupts. DMA channel interrupts.

Bit(s)	Name	Description	Value
4	stream_cmd	Acknowledge stream_cmd interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
3	in_eop	Acknowledge in_eop interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	data	Acknowledge data interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	ctxt	Acknowledge ctxt interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	group	Acknowledge group interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.11.38 r_intr

Address	0x94
Default	
Type	Read
Description	Interrupts before the mask. DMA channel interrupts. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	stream_cmd	Interrupt stream_cmd active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	in_eop	Interrupt in_eop active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	data	Interrupt data active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	ctxt	Interrupt ctxt active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	group	Interrupt group active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.11.39 r_masked_intr

Address	0x98
Default	
Type	Read
Description	Interrupts after the mask. DMA channel interrupts. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	stream_cmd	Interrupt stream_cmd active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	in_eop	Interrupt in_eop active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	data	Interrupt data active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	ctxt	Interrupt ctxt active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	group	Interrupt group active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.11.40 rw_stream_cmd

Address	0x9c
Default	0x00000000
Type	Read/Write
Description	Stream command. This register holds the commands executing on the DMA channel. May only be written to if busy is not set.

Bit(s)	Name	Description	Value
31	busy	Busy. Indicates if the DMA channel is busy. yes: The DMA channel is busy no: The DMA channel is not busy	yes=1 no=0
23-16:8	n	Offset parameter to the command cmd.load_c_n.	

9-0:10	cmd	Command. DMA channel stream commands. store_descr: Store descriptor(nop) store_md: Store meta data store_c: Store context descriptor store_g: Store group descriptor array: When next_en, stop at eol next_en: Find the next enabled descriptor copy_next: Make the down pointer of the upper group, point to the current next group next_pkt: Go to the first data descriptor of the next packet dis_c: Disable the current context descriptor dis_g: Disable the current group descriptor copy_up: Copy current up pointer when loading the next group descriptor save_down: Make the down pointer of the upper group, point to current group save_up: Make the up pointer of the lower group, point to current group update_down: Make the down pointer of the group point to the loaded context restore: Restore context descriptor burst: Start bursting set_reg: Used to modify pointer registers only. reg(n) <= context_md_wr[47:16], where n == cmd[23:16], and mapped to registers according to the 'addr' field in this file. For non pointer registers and non rw registers the command is a no operation. ack_pkt: Acknowledge packet(continue burst) load_d: Load data descriptor set_w_size1: Set word size to 1 byte set_w_size2: Set word size to 2 bytes set_w_size4: Set word size to 4 bytes load_c: Load context descriptor load_c_next: Load next context descriptor load_c_n: Load n:th context descriptor load_g: Load group descriptor load_g_next: Load next group descriptor load_g_up: Load upper group descriptor load_g_down: Load lower group descriptor	store_descr=0 store_md=1 store_c=2 store_g=4 array=8 next_en=16 copy_next=16 next_pkt=16 dis_c=16 dis_g=32 copy_up=32 save_down=32 save_up=32 update_down=32 restore=32 burst=32 set_reg=80 ack_pkt=256 load_d=320 set_w_size1=400 set_w_size2=416 set_w_size4=448 load_c=512 load_c_next=576 load_c_n=640 load_g=768 load_g_next=832 load_g_up=896 load_g_down=960

25.12 eth

Instance	Base Address
eth0	0xb0034000
eth1	0xb0036000

25.12.1 rw_ma0_lo

Address	0x0
Default	0x00000000
Type	Read/Write
Description	Bit 31 to 0 of station address MA0.

Bit(s)	Name	Description	Value
31-0:32	addr	Bit 31 to 0 of station address MA0.	

25.12.2 rw_ma0_hi

Address	0x4
Default	0x00000000
Type	Read/Write
Description	Bit 47 to 32 of station address MA0.

Bit(s)	Name	Description	Value
15-0:16	addr	Bit 47 to 32 of station address MA0.	

25.12.3 rw_ma1_lo

Address	0x8
Default	0x00000000
Type	Read/Write
Description	Bit 31 to 0 of station address MA1.

Bit(s)	Name	Description	Value
31-0:32	addr	Bit 31 to 0 of station address MA1.	

25.12.4 rw_ma1_hi

Address	0xc
Default	0x00000000
Type	Read/Write
Description	Bit 47 to 32 of station address MA1.

Bit(s)	Name	Description	Value
15-0:16	addr	Bit 47 to 32 of station address MA1.	

25.12.5 rw_ga_lo

Address	0x10
Default	0x00000000
Type	Read/Write
Description	Bit 31 to 0 of the group address table.

Bit(s)	Name	Description	Value
31-0:32	table	Bit 31 to 0 of the group address table.	

25.12.6 rw_ga_hi

Address	0x14
Default	0x00000000
Type	Read/Write
Description	Bit 63 to 32 of the group address table.

Bit(s)	Name	Description	Value
31-0:32	table	Bit 63 to 32 of the group address table.	

25.12.7 rw_gen_ctrl

Address	0x18
Default	0x00000000
Type	Read/Write
Description	General control register for the Ethernet interface.

Bit(s)	Name	Description	Value
5	flow_ctrl	This field turns flow control on or off. no: Flow control disable yes: Flow control enable	no=0 yes=1
4	loopback	This field turns internal loop back mode on or off. In loop back mode, the transmitted frames are directly passed to the receiver and the MII interface is disabled. no: Normal mode yes: Internal loop back mode	no=0 yes=1
3	protocol	Selects access protocol. ether: Ethernet (CSMA/CD) access protocol hsh: Handshake protocol	ether=0 hsh=1
2-1:2	phy	Selects physical connection protocol. mii_clk: MII with 25 MHz clock output on phyclk pin mii: MII interface mii_arec: MII with address recognized signal on phyclk pin	mii_clk=0 mii=1 mii_arec=2
0	en	General enable for the network interface. no: Network interface disabled yes: Network interface enabled	no=0 yes=1

25.12.8 rw_rec_ctrl

Address	0x1c
Default	
Type	Read/Write
Description	Control register for the Ethernet receiver.

Bit(s)	Name	Description	Value
8	max_size	This bit determines the maximum packet size for Ethernet packets. size1518: 1518 bytes maximum packet size size1522: 1522 bytes maximum packet size	size1518=0 size1522=1
7	duplex	Selects full or half duplex. half: Half duplex full: Full duplex	half=0 full=1
6	bad_crc	Receive or discard packets with crc or alignment errors. discard: Discard frames with errors rec: Receive all frames	discard=0 rec=1
5	oversize	Receive or discard oversized packets. discard: Discard oversized frames rec: Receive all frames	discard=0 rec=1
4	undersize	Receive or discard undersized packets. discard: Discard undersized frames rec: Receive all frames	discard=0 rec=1
3	broadcast	Receive or discard broadcast frames (address 0xffffffffffff). discard: Discard broadcast frames rec: Receive all frames	discard=0 rec=1
2	individual	Selects if the group address table will also match individual addresses or not. no: The group address table ony matches group addresses yes: The group address table matches all addresses	no=0 yes=1
1	ma1	Use station address MA1 no: MA1 address disabled yes: MA1 address enabled	no=0 yes=1
0	ma0	Use station address MA0 no: MA0 address disabled yes: MA0 address enabled	no=0 yes=1

25.12.9 rw_tr_ctrl

Address	0x20
Default	
Type	Read/Write
Description	Control register for the Ethernet transmitter.

Bit(s)	Name	Description	Value
6	ignore_crs	When set to no the transmitter will reset the interframe gap timer if **crs** is asserted during the first 64 bits of the interframe gap. If set to yes the transmitter will ignore **crs** while measuring the interframe gap, meaning that after **crs** is deasserted the transmitter will wait for 96 bits and then start to transmit. no: Check **crs** in interframe gap yes: Ignore **crs** in interframe gap	no=0 yes=1
5	hsh_delay	Add 2 μs delay after acknowledge in handshake access protocol mode. This mode bit has no effect in CSMA/CD (Ethernet) access protocol mode. no: No delay yes: 2 μs delay	no=0 yes=1
4	cancel	This field cancels a pending frame. If set to yes, the transmitter completes the current transmission attempt (if any), and then stops. The excessive retry condition is then entered, regardless of whether a transmission was in progress. no: No action yes: Cancel pending transmission attempts	no=0 yes=1
3	ignore_col	Selects whether to use or ignore the **col** signal. This field only takes effect in half duplex Ethernet mode. It should be set to no for normal operation. no: Normal operation yes: Collision detect ignored	no=0 yes=1
2	retry	Enable or disable Ethernet transmission retries after a collision. They should be enabled for normal Ethernet operation. no: Transmission retries disabled yes: Transmission retries enabled	no=0 yes=1
1	pad	If this field is set to yes, a frame shorter than 60 bytes (excluding preamble, start of frame delimiter and CRC) is padded to 60 bytes. The pad consists of all 0's. no: Do not pad yes: Pad short frames	no=0 yes=1
0	crc	Selects whether to add CRC to the end of the transmitted packets. no: Do not add CRC yes: Add CRC	no=0 yes=1

25.12.10 rw_clr_err

Address	0x24
Default	
Type	Read/Write
Description	This register clears the excessive retry and underrun conditions. The register is transient. Its value reverts to 0 after action is taken.

Bit(s)	Name	Description	Value
0	clr	Clears the excessive retry and underrun conditions. no: No action yes: Clear error conditions	no=0 yes=1

25.12.11 rw_mgm_ctrl

Address	0x28
Default	0x00000000
Type	Read/Write
Description	This register controls the management port of the MII interface. It also controls the values on the MII output pins when the network interface is disabled.

Bit(s)	Name	Description	Value
8	txen	This field is output on the **txen** pin when the network controller is disabled. When read, this field reflects the value set in the register. To read the actual value on the pin, use the r_stat register.	
7-4:4	txdata	This field is output on **txd3 - txd0** when the network controller is disabled or in internal loopback mode. When read, this field reflects the value set in the register. To read the actual values on the pins, use the r_stat register.	
3	phyclk	This field is output on the **phyclk** pin when the network controller is disabled. When read, this field reflects the value set in the register. To read the actual value on the pin, use the r_stat register.	
2	mdc	This field is output on the **mdc** pin.	
1	mdoe	This field controls the ouput enable of the **mdio** pin. no: Output disabled yes: Output enabled	no=0 yes=1
0	mdio	This field is output on the **mdio** pin. When read, this field reflects the value of the register. To read the actual value on the pin, use the r_stat register.	

25.12.12 r_stat

Address	0x2c
Default	
Type	Read
Description	Network interface status.

Bit(s)	Name	Description	Value
18	rxclk	Value on the **rxclk** pin.	
17	rxdv	Value on the **rxdv** pin.	
16	rxer	Value on the **rxer** pin.	
15-12:4	rxdata	Values on the **rxd3 - rxd0** pins.	
11	txclk	Value on the **txclk** pin.	
10	crs	Value on the **crs** pin.	
9	col	Value on the **col** pin.	
8	txen	Value on the **txen** pin.	
7-4:4	txdata	Values on the **txd3 - txd0** pins.	
3	phyclk	Value on the **phyclk** pin.	
2	urun	This bit is set if the transmitter is stopped due to a detected underrun. The underrun condition is cleared by the rw_clr_err register. no: No underrun yes: Transmitter stopped due to underrun	no=0 yes=1
1	exc_col	This bit is set if the transmitter is stopped due to excessive transmission retries. The excessive retry condition is cleared by the rw_clr_err register. no: No excessive retries yes: Transmitter stopped due to excessive retries	no=0 yes=1
0	mdio	Value on the **mdio** pin	

25.12.13 rs_rec_cnt/r_rec_cnt

Address	0x30/0x34
Default	
Type	Read with side effects/Read
Description	Receive error counters.

Bit(s)	Name	Description	Value
31-24:8	congestion	This field gives the number of otherwise correct frames that were not received due to a receiver overrun condition.	
23-16:8	oversize	This field gives the number of oversized frames.	
15-8:8	align_err	This field gives the number of frames with alignment errors.	
7-0:8	crc_err	This field gives the number of frames with CRC errors.	

25.12.14 rs_tr_cnt/r_tr_cnt

Address	0x38/0x3c
Default	
Type	Read with side effects/Read
Description	Transmit statistics counters.

Bit(s)	Name	Description	Value
31-24:8	deferred	This field gives the number of deferred transmit frames.	
23-16:8	late_col	This field gives the number of frames that were involved in late collisions.	
15-8:8	mult_col	This field gives the number of frames that were involved in more than one collision.	
7-0:8	single_col	This field gives the number of frames that were involved in exactly one collision.	

25.12.15 rs_phy_cnt/r_phy_cnt

Address	0x40/0x44
Default	
Type	Read with side effects/Read
Description	Physical layer error counters.

Bit(s)	Name	Description	Value
15-8:8	sqe_err	When the chip is configured for 10 Mb Ethernet, this field gives the number of transmit frames for which the sqe test signal was not recognized.	
7-0:8	carrier_loss	This field gives the number of transmit frames for which the carrier sense signal was not constantly present during the transmission.	

25.12.16 rw_test_ctrl

Address	0x48
Default	0x00000000
Type	Read/Write
Description	This register controls various internal test modes associated with the network interface.

Bit(s)	Name	Description	Value
2	backoff	Turn network transmitter backoff test mode on/off. When enabling the backoff test mode, the backoff counter will be decremented every tx_clk cycle. This will result in a shorter backoff time. This mode should only be used during testing. When configured for loopback mode this bit also enables the **col** signal. no: Backoff test off yes: Backoff test on	no=0 yes=1
1	snmp	Turn error and statistics counter test mode on/off. Makes it possible to use the snmp_inc field described below to increment and test the error and statistics counters. no: Normal mode yes: Counter test mode on	no=0 yes=1
0	snmp_inc	Error and statistics counter test mode increment clock. In test mode, a 0->1 transition makes all error and statistics counters count up by one. no: Clock signal equal to 0 yes: Clock signal equal to 1	no=0 yes=1

25.12.17 rw_intr_mask

Address	0x4c
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts from the Ethernet interface. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
13	mdio	Enable/disable mdio interrupt. Generated when the **mdio** pin is low. This interrupt should be masked off during normal transfers over the MDIO interface. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
12	exc_col	Enable/disable exc_col interrupt. Generated when an excessive collision has occured. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
11	urun	Enable/disable urun interrupt. Generated when transmitter underrun has occured. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
10	orun	Enable/disable orun interrupt. Generated when receiver overrun has occured. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
9	sqe_test_err	Enable/disable sqe_test_err interrupt. Interrupt generated when the Ethernet SQE test error counter reaches 128. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
8	carrier_loss	Enable/disable carrier_loss interrupt. Generated when the Ethernet carrier loss counter reaches 128. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
7	deferred	Enable/disable deferred interrupt. Generated when the Ethernet deferred counter reaches 128. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
6	late_col	Enable/disable late_col interrupt. Generated when the Ethernet late collision counter reaches 128. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

5	mult_col	Enable/disable mult_col interrupt. Generated when the Ethernet multiple collision counter reaches 128. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	single_col	Enable/disable single_col interrupt. Generated when the Ethernet single collision counter reaches 128. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	congestion	Enable/disable congestion interrupt. Generated when the Ethernet congestion counter reaches 128. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	oversize	Enable/disable oversize interrupt. Generated when the Ethernet oversize counter reaches 128. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	align	Enable/disable align interrupt. Generated when the Ethernet alignment error counter reaches 128. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	crc	Enable/disable crc interrupt. Generated when the Ethernet CRC error counter reaches 128. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.12.18 rw_ack_intr

Address	0x50
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts from the Ethernet interface.

Bit(s)	Name	Description	Value
13	mdio	Acknowledge mdio interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
12	exc_col	Acknowledge exc_col interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
11	urun	Acknowledge urun interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
10	orun	Acknowledge orun interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
9	sqe_test_err	Acknowledge sqe_test_err interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
8	carrier_loss	Acknowledge carrier_loss interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
7	deferred	Acknowledge deferred interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
6	late_col	Acknowledge late_col interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
5	mult_col	Acknowledge mult_col interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
4	single_col	Acknowledge single_col interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
3	congestion	Acknowledge congestion interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	oversize	Acknowledge oversize interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

1	align	Acknowledge align interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	crc	Acknowledge crc interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.12.19 r_intr

Address	0x54
Default	
Type	Read
Description	Interrupts before the mask. Interrupts from the Ethernet interface. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
13	mdio	Interrupt mdio active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	exc_col	Interrupt exc_col active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	urun	Interrupt urun active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	orun	Interrupt orun active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	sqe_test_err	Interrupt sqe_test_err active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	carrier_loss	Interrupt carrier_loss active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	deferred	Interrupt deferred active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	late_col	Interrupt late_col active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	mult_col	Interrupt mult_col active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
4	single_col	Interrupt single_col active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

3	congestion	Interrupt congestion active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	oversize	Interrupt oversize active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	align	Interrupt align active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	crc	Interrupt crc active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.12.20 r_masked_intr

Address	0x58
Default	
Type	Read
Description	Interrupts after the mask. Interrupts from the Ethernet interface. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
13	mdio	Interrupt mdio active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	exc_col	Interrupt exc_col active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	urun	Interrupt urun active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	orun	Interrupt orun active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	sqe_test_err	Interrupt sqe_test_err active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	carrier_loss	Interrupt carrier_loss active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	deferred	Interrupt deferred active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	late_col	Interrupt late_col active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	mult_col	Interrupt mult_col active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
4	single_col	Interrupt single_col active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	congestion	Interrupt congestion active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

2	oversize	Interrupt oversize active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	align	Interrupt align active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	crc	Interrupt crc active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.13 gio

Instance	Base Address
gio	0xb001a000

25.13.1 rw_pa_dout

Address	0x0
Default	
Type	Read/Write
Description	General port **pa** data out. When read, it only reads back the register values. To read the actual port data, use r_pa_din.

Bit(s)	Name	Description	Value
7-0:8	data	Data out.	

25.13.2 r_pa_din

Address	0x4
Default	
Type	Read
Description	General port **pa** data in.

Bit(s)	Name	Description	Value
7-0:8	data	Data in.	

25.13.3 rw_pa_oe

Address	0x8
Default	0x00000000
Type	Read/Write
Description	General port **pa** output enable.

Bit(s)	Name	Description	Value
7-0:8	oe	Output enables.	

25.13.4 rw_intr_cfg

Address	0xc
Default	0x00000000
Type	Read/Write
Description	Configuration register for the interrupts on port **pa**.

Bit(s)	Name	Description	Value
23-21:3	pa7	configuration for interrupt on **pa7**. off: Interrupt turned off hi: Level triggered active high lo: Level triggered active low set: Interrupt always set posedge: Positive edge triggered negedge: Negative edge triggered anyedge: Triggered on both edges	off=0 hi=1 lo=2 set=3 posedge=5 negedge=6 anyedge=7
20-18:3	pa6	configuration for interrupt on **pa6**. off: Interrupt turned off hi: Level triggered active high lo: Level triggered active low set: Interrupt always set posedge: Positive edge triggered negedge: Negative edge triggered anyedge: Triggered on both edges	off=0 hi=1 lo=2 set=3 posedge=5 negedge=6 anyedge=7
17-15:3	pa5	configuration for interrupt on **pa5**. off: Interrupt turned off hi: Level triggered active high lo: Level triggered active low set: Interrupt always set posedge: Positive edge triggered negedge: Negative edge triggered anyedge: Triggered on both edges	off=0 hi=1 lo=2 set=3 posedge=5 negedge=6 anyedge=7
14-12:3	pa4	configuration for interrupt on **pa4**. off: Interrupt turned off hi: Level triggered active high lo: Level triggered active low set: Interrupt always set posedge: Positive edge triggered negedge: Negative edge triggered anyedge: Triggered on both edges	off=0 hi=1 lo=2 set=3 posedge=5 negedge=6 anyedge=7

11-9:3	pa3	configuration for interrupt on **pa3**. off: Interrupt turned off hi: Level triggered active high lo: Level triggered active low set: Interrupt always set posedge: Positive edge triggered negedge: Negative edge triggered anyedge: Triggered on both edges	off=0 hi=1 lo=2 set=3 posedge=5 negedge=6 anyedge=7
8-6:3	pa2	configuration for interrupt on **pa2**. off: Interrupt turned off hi: Level triggered active high lo: Level triggered active low set: Interrupt always set posedge: Positive edge triggered negedge: Negative edge triggered anyedge: Triggered on both edges	off=0 hi=1 lo=2 set=3 posedge=5 negedge=6 anyedge=7
5-3:3	pa1	configuration for interrupt on **pa1**. off: Interrupt turned off hi: Level triggered active high lo: Level triggered active low set: Interrupt always set posedge: Positive edge triggered negedge: Negative edge triggered anyedge: Triggered on both edges	off=0 hi=1 lo=2 set=3 posedge=5 negedge=6 anyedge=7
2-0:3	pa0	configuration for interrupt on **pa0**. off: Interrupt turned off hi: Level triggered active high lo: Level triggered active low set: Interrupt always set posedge: Positive edge triggered negedge: Negative edge triggered anyedge: Triggered on both edges	off=0 hi=1 lo=2 set=3 posedge=5 negedge=6 anyedge=7

25.13.5 rw_intr_mask

Address	0x10
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts from port **pa**. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
7	pa7	Enable/disable pa7 interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
6	pa6	Enable/disable pa6 interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
5	pa5	Enable/disable pa5 interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	pa4	Enable/disable pa4 interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	pa3	Enable/disable pa3 interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	pa2	Enable/disable pa2 interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	pa1	Enable/disable pa1 interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	pa0	Enable/disable pa0 interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.13.6 rw_ack_intr

Address	0x14
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts from port **pa**.

Bit(s)	Name	Description	Value
7	pa7	Acknowledge pa7 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
6	pa6	Acknowledge pa6 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
5	pa5	Acknowledge pa5 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
4	pa4	Acknowledge pa4 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
3	pa3	Acknowledge pa3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	pa2	Acknowledge pa2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	pa1	Acknowledge pa1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	pa0	Acknowledge pa0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.13.7 r_intr

Address	0x18
Default	
Type	Read
Description	Interrupts before the mask. Interrupts from port **pa**. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
7	pa7	Interrupt pa7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	pa6	Interrupt pa6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	pa5	Interrupt pa5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
4	pa4	Interrupt pa4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	pa3	Interrupt pa3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	pa2	Interrupt pa2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	pa1	Interrupt pa1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	pa0	Interrupt pa0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.13.8 r_masked_intr

Address	0x1c
Default	
Type	Read
Description	Interrupts after the mask. Interrupts from port **pa**. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
7	pa7	Interrupt pa7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	pa6	Interrupt pa6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	pa5	Interrupt pa5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
4	pa4	Interrupt pa4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	pa3	Interrupt pa3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	pa2	Interrupt pa2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	pa1	Interrupt pa1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	pa0	Interrupt pa0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.13.9 rw_pb_dout

Address	0x20
Default	
Type	Read/Write
Description	General port **pb** data out. When read, it only reads back the register values. To read the actual port data, use r_pb_din.

Bit(s)	Name	Description	Value
17-0:18	data	Data out.	

25.13.10 r_pb_din

Address	0x24
Default	
Type	Read
Description	General port **pb** data in.

Bit(s)	Name	Description	Value
17-0:18	data	Data in.	

25.13.11 rw_pb_oe

Address	0x28
Default	0x00000000
Type	Read/Write
Description	General port **pb** output enable.

Bit(s)	Name	Description	Value
17-0:18	oe	Output enables.	

25.13.12 rw_pc_dout

Address	0x30
Default	
Type	Read/Write
Description	General port **pc** data out. When read, it only reads back the register values. To read the actual port data, use r_pc_din.

Bit(s)	Name	Description	Value
17-0:18	data	Data out.	

25.13.13 r_pc_din

Address	0x34
Default	
Type	Read
Description	General port **pc** data in.

Bit(s)	Name	Description	Value
17-0:18	data	Data in.	

25.13.14 rw_pc_oe

Address	0x38
Default	0x00000000
Type	Read/Write
Description	General port **pc** output enable.

Bit(s)	Name	Description	Value
17-0:18	oe	Output enables.	

25.13.15 rw_pd_dout

Address	0x40
Default	
Type	Read/Write
Description	General port **pd** data out. When read, it only reads back the register values. To read the actual port data, use r_pd_din.

Bit(s)	Name	Description	Value
17-0:18	data	Data out.	

25.13.16 r_pd_din

Address	0x44
Default	
Type	Read
Description	General port **pd** data in.

Bit(s)	Name	Description	Value
17-0:18	data	Data in.	

25.13.17 rw_pd_oe

Address	0x48
Default	0x00000000
Type	Read/Write
Description	General port **pd** output enable.

Bit(s)	Name	Description	Value
17-0:18	oe	Output enables.	

25.13.18 rw_pe_dout

Address	0x50
Default	
Type	Read/Write
Description	General port **pe** data out. When read, it only reads back the register values. To read the actual port data, use r_pe_din.

Bit(s)	Name	Description	Value
17-0:18	data	Data out.	

25.13.19 r_pe_din

Address	0x54
Default	
Type	Read
Description	General port **pe** data in.

Bit(s)	Name	Description	Value
17-0:18	data	Data in.	

25.13.20 rw_pe_oe

Address	0x58
Default	0x00000000
Type	Read/Write
Description	General port **pe** output enable.

Bit(s)	Name	Description	Value
17-0:18	oe	Output enables.	

25.14 intr_vect

Instance	Base Address
irq	0xb001c000

25.14.1 rw_mask

Address	0x0
Default	0x00000000
Type	Read/Write
Description	Interrupt vector mask. Enables or disables all interrupts from a particular source.

Bit(s)	Name	Description	Value
29	ext	Enable/disable External IRQ pin interrupts. Vector number 0x4e. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
28	bif_dma	Enable/disable Bus interface DMA interrupts. Vector number 0x4d. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
27	bif_arb	Enable/disable Bus interface arbiter interrupts. Vector number 0x4c. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
26	timer	Enable/disable Timers interrupts. Vector number 0x4b. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
25	eth1	Enable/disable Ethernet port 1 interrupts. Vector number 0x4a. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
24	eth0	Enable/disable Ethernet port 0 interrupts. Vector number 0x49. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
22	ser3	Enable/disable Serial port 3 interrupts. Vector number 0x47. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
21	ser2	Enable/disable Serial port 2 interrupts. Vector number 0x46. yes: Enable interrupts no: Disable interrupts	yes=1 no=0

20	ser1	Enable/disable Serial port 1 interrupts. Vector number 0x45. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
19	ser0	Enable/disable Serial port 0 interrupts. Vector number 0x44. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
18	sser1	Enable/disable Synchronous serial port 1 interrupts. Vector number 0x43. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
17	sser0	Enable/disable Synchronous serial port 0 interrupts. Vector number 0x42. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
16	ata	Enable/disable ATA interface interrupts. Vector number 0x41. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
15	dma9	Enable/disable dma channel 9 interrupts. Vector number 0x40. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
14	dma8	Enable/disable dma channel 8 interrupts. Vector number 0x3f. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
13	dma7	Enable/disable dma channel 7 interrupts. Vector number 0x3e. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
12	dma6	Enable/disable dma channel 6 interrupts. Vector number 0x3d. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
11	dma5	Enable/disable dma channel 5 interrupts. Vector number 0x3c. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
10	dma4	Enable/disable dma channel 4 interrupts. Vector number 0x3b. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
9	dma3	Enable/disable dma channel 3 interrupts. Vector number 0x3a. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
8	dma2	Enable/disable dma channel 2 interrupts. Vector number 0x39. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
7	dma1	Enable/disable dma channel 1 interrupts. Vector number 0x38. yes: Enable interrupts no: Disable interrupts	yes=1 no=0

6	dma0	Enable/disable dma channel 0 interrupts. Vector number 0x37. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
5	iop3	Enable/disable I/O processor port 3 interrupts. Vector number 0x36. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
4	iop2	Enable/disable I/O processor port 2 interrupts. Vector number 0x35. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
3	iop1	Enable/disable I/O processor port 1 interrupts. Vector number 0x34. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
2	iop0	Enable/disable I/O processor port 0 interrupts. Vector number 0x33. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
1	gen_io	Enable/disable general I/O interrupts. Vector number 0x32. yes: Enable interrupts no: Disable interrupts	yes=1 no=0
0	memarb	Enable/disable memory arbiter breakpoints interrupts. Vector number 0x31. yes: Enable interrupts no: Disable interrupts	yes=1 no=0

25.14.2 r_vect

Address	0x4
Default	
Type	Read
Description	Unmasked vectors. Interrupt status before the vector mask. May be used to tell if there are active interrupts for a particular vector even if it that vector is masked.

Bit(s)	Name	Description	Value
29	ext	Set if there are active External IRQ pin interrupts before vector mask. Vector number 0x4e. yes: Active interrupts no: No active interrupts	yes=1 no=0
28	bif_dma	Set if there are active Bus interface DMA interrupts before vector mask. Vector number 0x4d. yes: Active interrupts no: No active interrupts	yes=1 no=0
27	bif_arb	Set if there are active Bus interface arbiter interrupts before vector mask. Vector number 0x4c. yes: Active interrupts no: No active interrupts	yes=1 no=0
26	timer	Set if there are active Timers interrupts before vector mask. Vector number 0x4b. yes: Active interrupts no: No active interrupts	yes=1 no=0
25	eth1	Set if there are active Ethernet port 1 interrupts before vector mask. Vector number 0x4a. yes: Active interrupts no: No active interrupts	yes=1 no=0
24	eth0	Set if there are active Ethernet port 0 interrupts before vector mask. Vector number 0x49. yes: Active interrupts no: No active interrupts	yes=1 no=0
22	ser3	Set if there are active Serial port 3 interrupts before vector mask. Vector number 0x47. yes: Active interrupts no: No active interrupts	yes=1 no=0
21	ser2	Set if there are active Serial port 2 interrupts before vector mask. Vector number 0x46. yes: Active interrupts no: No active interrupts	yes=1 no=0
20	ser1	Set if there are active Serial port 1 interrupts before vector mask. Vector number 0x45. yes: Active interrupts no: No active interrupts	yes=1 no=0

19	ser0	Set if there are active Serial port 0 interrupts before vector mask. Vector number 0x44. yes: Active interrupts no: No active interrupts	yes=1 no=0
18	sser1	Set if there are active Synchronous serial port 1 interrupts before vector mask. Vector number 0x43. yes: Active interrupts no: No active interrupts	yes=1 no=0
17	sser0	Set if there are active Synchronous serial port 0 interrupts before vector mask. Vector number 0x42. yes: Active interrupts no: No active interrupts	yes=1 no=0
16	ata	Set if there are active ATA interface interrupts before vector mask. Vector number 0x41. yes: Active interrupts no: No active interrupts	yes=1 no=0
15	dma9	Set if there are active dma channel 9 interrupts before vector mask. Vector number 0x40. yes: Active interrupts no: No active interrupts	yes=1 no=0
14	dma8	Set if there are active dma channel 8 interrupts before vector mask. Vector number 0x3f. yes: Active interrupts no: No active interrupts	yes=1 no=0
13	dma7	Set if there are active dma channel 7 interrupts before vector mask. Vector number 0x3e. yes: Active interrupts no: No active interrupts	yes=1 no=0
12	dma6	Set if there are active dma channel 6 interrupts before vector mask. Vector number 0x3d. yes: Active interrupts no: No active interrupts	yes=1 no=0
11	dma5	Set if there are active dma channel 5 interrupts before vector mask. Vector number 0x3c. yes: Active interrupts no: No active interrupts	yes=1 no=0
10	dma4	Set if there are active dma channel 4 interrupts before vector mask. Vector number 0x3b. yes: Active interrupts no: No active interrupts	yes=1 no=0
9	dma3	Set if there are active dma channel 3 interrupts before vector mask. Vector number 0x3a. yes: Active interrupts no: No active interrupts	yes=1 no=0

8	dma2	Set if there are active dma channel 2 interrupts before vector mask. Vector number 0x39. yes: Active interrupts no: No active interrupts	yes=1 no=0
7	dma1	Set if there are active dma channel 1 interrupts before vector mask. Vector number 0x38. yes: Active interrupts no: No active interrupts	yes=1 no=0
6	dma0	Set if there are active dma channel 0 interrupts before vector mask. Vector number 0x37. yes: Active interrupts no: No active interrupts	yes=1 no=0
5	iop3	Set if there are active I/O processor port 3 interrupts before vector mask. Vector number 0x36. yes: Active interrupts no: No active interrupts	yes=1 no=0
4	iop2	Set if there are active I/O processor port 2 interrupts before vector mask. Vector number 0x35. yes: Active interrupts no: No active interrupts	yes=1 no=0
3	iop1	Set if there are active I/O processor port 1 interrupts before vector mask. Vector number 0x34. yes: Active interrupts no: No active interrupts	yes=1 no=0
2	iop0	Set if there are active I/O processor port 0 interrupts before vector mask. Vector number 0x33. yes: Active interrupts no: No active interrupts	yes=1 no=0
1	gen_io	Set if there are active general I/O interrupts before vector mask. Vector number 0x32. yes: Active interrupts no: No active interrupts	yes=1 no=0
0	memarb	Set if there are active memory arbiter breakpoints interrupts before vector mask. Vector number 0x31. yes: Active interrupts no: No active interrupts	yes=1 no=0

25.14.3 r_masked_vect

Address	0x8
Default	
Type	Read
Description	Masked vectors. Interrupt status after the vector mask. If more than one vector is active and unmasked at the same time, this register may be used to tell which these vectors are. The value of this register can be expressed as: r_masked_vect = r_vect & rw_mask;

Bit(s)	Name	Description	Value
29	ext	Set if there are active External IRQ pin interrupts after vector mask. Vector number 0x4e. yes: Active interrupts no: No active interrupts	yes=1 no=0
28	bif_dma	Set if there are active Bus interface DMA interrupts after vector mask. Vector number 0x4d. yes: Active interrupts no: No active interrupts	yes=1 no=0
27	bif_arb	Set if there are active Bus interface arbiter interrupts after vector mask. Vector number 0x4c. yes: Active interrupts no: No active interrupts	yes=1 no=0
26	timer	Set if there are active Timers interrupts after vector mask. Vector number 0x4b. yes: Active interrupts no: No active interrupts	yes=1 no=0
25	eth1	Set if there are active Ethernet port 1 interrupts after vector mask. Vector number 0x4a. yes: Active interrupts no: No active interrupts	yes=1 no=0
24	eth0	Set if there are active Ethernet port 0 interrupts after vector mask. Vector number 0x49. yes: Active interrupts no: No active interrupts	yes=1 no=0
22	ser3	Set if there are active Serial port 3 interrupts after vector mask. Vector number 0x47. yes: Active interrupts no: No active interrupts	yes=1 no=0
21	ser2	Set if there are active Serial port 2 interrupts after vector mask. Vector number 0x46. yes: Active interrupts no: No active interrupts	yes=1 no=0

20	ser1	Set if there are active Serial port 1 interrupts after vector mask. Vector number 0x45. yes: Active interrupts no: No active interrupts	yes=1 no=0
19	ser0	Set if there are active Serial port 0 interrupts after vector mask. Vector number 0x44. yes: Active interrupts no: No active interrupts	yes=1 no=0
18	sser1	Set if there are active Synchronous serial port 1 interrupts after vector mask. Vector number 0x43. yes: Active interrupts no: No active interrupts	yes=1 no=0
17	sser0	Set if there are active Synchronous serial port 0 interrupts after vector mask. Vector number 0x42. yes: Active interrupts no: No active interrupts	yes=1 no=0
16	ata	Set if there are active ATA interface interrupts after vector mask. Vector number 0x41. yes: Active interrupts no: No active interrupts	yes=1 no=0
15	dma9	Set if there are active dma channel 9 interrupts after vector mask. Vector number 0x40. yes: Active interrupts no: No active interrupts	yes=1 no=0
14	dma8	Set if there are active dma channel 8 interrupts after vector mask. Vector number 0x3f. yes: Active interrupts no: No active interrupts	yes=1 no=0
13	dma7	Set if there are active dma channel 7 interrupts after vector mask. Vector number 0x3e. yes: Active interrupts no: No active interrupts	yes=1 no=0
12	dma6	Set if there are active dma channel 6 interrupts after vector mask. Vector number 0x3d. yes: Active interrupts no: No active interrupts	yes=1 no=0
11	dma5	Set if there are active dma channel 5 interrupts after vector mask. Vector number 0x3c. yes: Active interrupts no: No active interrupts	yes=1 no=0
10	dma4	Set if there are active dma channel 4 interrupts after vector mask. Vector number 0x3b. yes: Active interrupts no: No active interrupts	yes=1 no=0

9	dma3	Set if there are active dma channel 3 interrupts after vector mask. Vector number 0x3a. yes: Active interrupts no: No active interrupts	yes=1 no=0
8	dma2	Set if there are active dma channel 2 interrupts after vector mask. Vector number 0x39. yes: Active interrupts no: No active interrupts	yes=1 no=0
7	dma1	Set if there are active dma channel 1 interrupts after vector mask. Vector number 0x38. yes: Active interrupts no: No active interrupts	yes=1 no=0
6	dma0	Set if there are active dma channel 0 interrupts after vector mask. Vector number 0x37. yes: Active interrupts no: No active interrupts	yes=1 no=0
5	iop3	Set if there are active I/O processor port 3 interrupts after vector mask. Vector number 0x36. yes: Active interrupts no: No active interrupts	yes=1 no=0
4	iop2	Set if there are active I/O processor port 2 interrupts after vector mask. Vector number 0x35. yes: Active interrupts no: No active interrupts	yes=1 no=0
3	iop1	Set if there are active I/O processor port 1 interrupts after vector mask. Vector number 0x34. yes: Active interrupts no: No active interrupts	yes=1 no=0
2	iop0	Set if there are active I/O processor port 0 interrupts after vector mask. Vector number 0x33. yes: Active interrupts no: No active interrupts	yes=1 no=0
1	gen_io	Set if there are active general I/O interrupts after vector mask. Vector number 0x32. yes: Active interrupts no: No active interrupts	yes=1 no=0
0	memarb	Set if there are active memory arbiter breakpoints interrupts after vector mask. Vector number 0x31. yes: Active interrupts no: No active interrupts	yes=1 no=0

25.14.4 r_nmi

Address	0xc
Default	
Type	Read
Description	Active non maskable interrupt (NMI) sources.

Bit(s)	Name	Description	Value
1	watchdog	Set if NMI from Watchdog timer is active. yes: active no: not active	yes=1 no=0
0	ext	Set if NMI from External NMI pin is active. yes: active no: not active	yes=1 no=0

25.14.5 r_guru

Address	0x10
Default	
Type	Read
Description	Active guru mode exception sources.

Bit(s)	Name	Description	Value
0	jtag	Set if guru mode exception signal from JTAG debug interface is active. yes: active no: not active	yes=1 no=0

25.15 iop_crc_par

Instance	Base Address
iop_crc_par0	0xb0020380
iop_crc_par1	0xb0020400

25.15.1 rw_cfg

Address	0x0
Default	0x00000000
Type	Read/Write
Description	Configures the parallel CRC.

Bit(s)	Name	Description	Value
8-6:3	poly	Select the CRC generator polynomial. Polynomials: **crc32** $p(x)=x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ **crc16** $p(x)=x^{16} + x^{15} + x^2 + 1$ **ccitt** $p(x)=x^{16} + x^{12} + x^5 + 1$ **crc5** $p(x)=x^5 + x^2 + 1$ crc5_11 differs from the others. It can only be used for calculating CRC when data is received from rw_wr2byte_last. Only the 11 least significant bits of rw_wr2byte_last are then used, all other bits of the register are ignored. CRC-5 is used by the Universal Serial Bus (USB) protocol. The data rate of USB 2.0 is too high for the I/O Processor to be handled serially. The serial data must therefore be transformed to 8-bit or 16-bit wide data words by an external circuit if the I/O Processor shall handle USB 2.0. crc32: CRC-32 crc16: CRC-16 ccitt: CRC-CCITT crc5: CRC-5 crc5_11: CRC-5 (11-bit data width)	crc32=0 crc16=1 ccitt=2 crc5=3 crc5_11=4
5-4:2	trig	Select if the strobe for incoming data on the receiving data interface, should be triggered when it is high or on positive or negative edge. hi: High strobe pos: Positive edge of strobe neg: Negative edge of strobe pos_neg: Both positive and negative edge of strobe	hi=0 pos=1 neg=2 pos_neg=3

3	inv_out	Select if the calculated CRC bits shall be inverted or not. no: CRC bits are not inverted yes: CRC bits are inverted	no=0 yes=1
2	rev_out	Select if the bit order of the CRC output shall be reversed or not. no: The bits are not reversed yes: The bits are reversed	no=0 yes=1
1	crc_out	Select if the calculated CRC should be sent after the transmitted data. crc_out is only used when mode equals calc. no: The CRC is never sent yes: The CRC is sent after the last data word	no=0 yes=1
0	mode	Configure the parallel CRC to either check the received CRC or calculate CRC for the transmitted data. check: Check received CRC calc: Calculate CRC	check=0 calc=1

25.15.2 rw_init_crc

Address	0x4
Default	
Type	Read/Write
Description	Initial shift register value of the parallel CRC.

25.15.3 rw_correct_crc

Address	0x8
Default	
Type	Read/Write
Description	Set the correct CRC value. If the CRC is correct, r_sh_reg shall have the value of rw_correct_crc, after the data stream (including the CRC) has been received, otherwise it is a CRC error.

25.15.4 rw_ctrl

Address	0xc
Default	0x00000000
Type	Read/Write
Description	Control register for the parallel CRC.

Bit(s)	Name	Description	Value
0	en	Enable or disable the CRC module. When disabled, the data will pass the parallel CRC module unchanged. no: Parallel CRC disabled yes: Parallel CRC enabled	no=0 yes=1

25.15.5 rw_set_last

Address	0x10
Default	
Type	Read/Write
Description	Write a zero-sized word to the parallel CRC. The word is marked as the last word of the packet.

Bit(s)	Name	Description	Value
0	tr_dif	Set last on the transmitting data interface. no: No action yes: Send zero-byte data, marked with last	no=0 yes=1

25.15.6 rw_wr1byte

Address	0x14
Default	
Type	Read/Write
Description	Write one byte of data to the CRC.

Bit(s)	Name	Description	Value
7-0:8	data	8-bit data to write to the CRC.	

25.15.7 rw_wr2byte

Address	0x18
Default	
Type	Read/Write
Description	Write two bytes of data to the CRC.

Bit(s)	Name	Description	Value
15-0:16	data	16-bit data to write to the CRC.	

25.15.8 rw_wr3byte

Address	0x1c
Default	
Type	Read/Write
Description	Write three bytes of data to the CRC.

Bit(s)	Name	Description	Value
23-0:24	data	24-bit data to write to the CRC.	

25.15.9 rw_wr4byte

Address	0x20
Default	
Type	Read/Write
Description	Write four bytes of data to the CRC.

Bit(s)	Name	Description	Value
31-0:32	data	32-bit data to write to the CRC.	

25.15.10 rw_wr1byte_last

Address	0x24
Default	
Type	Read/Write
Description	Write one byte of data to the crc and mark the data as the last of the data stream.

Bit(s)	Name	Description	Value
7-0:8	data	8-bit data to write to the parallel to serial converter.	

25.15.11 rw_wr2byte_last

Address	0x28
Default	
Type	Read/Write
Description	Write two bytes of data to the crc and mark the data as the last of the data stream.

Bit(s)	Name	Description	Value
15-0:16	data	16-bit data to write to the parallel to serial converter.	

25.15.12 rw_wr3byte_last

Address	0x2c
Default	
Type	Read/Write
Description	Write three bytes of data to the crc and mark the data as the last of the data stream.

Bit(s)	Name	Description	Value
23-0:24	data	24-bit data to write to the parallel to serial converter.	

25.15.13 rw_wr4byte_last

Address	0x30
Default	
Type	Read/Write
Description	Write four bytes of data to the crc and mark the data as the last of the data stream.

Bit(s)	Name	Description	Value
31-0:32	data	32-bit data to write to the parallel to serial converter.	

25.15.14 r_stat

Address	0x34
Default	
Type	Read
Description	Status register for the parallel CRC.

Bit(s)	Name	Description	Value
1	busy	Shows if it is possible to send more data to the parallel CRC block. no: Data can be received yes: Data can not be received	no=0 yes=1
0	err	Check if r_sh_reg equals rw_correct_crc or not. no: CRC is correct yes: CRC error	no=0 yes=1

25.15.15 r_sh_reg

Address	0x38
Default	
Type	Read
Description	The current value of the CRC shift register.

25.15.16 r_crc

Address	0x3c
Default	
Type	Read
Description	The current value of the CRC. I.e. the value of r_sh_reg where the bits are inverted or the bit order reversed depending on the value of inv_out and rev_out.

25.15.17 rw_strb_rec_dif_in

Address	0x40
Default	
Type	Read/Write
Description	The current incoming data on the received data interface will be read by the parallel CRC when writing to rw_strb_rec_dif_in.

Bit(s)	Name	Description	Value
1-0:2	last	The data can be marked as the last word of the data stream depending on the value of last and the value of signal last on the data interface. no: No last is set yes: Last is set dif_in: Last is sampled from dif_in last signal	no=0 yes=1 dif_in=2

25.16 iop_dmc_in

Instance	Base Address
iop_dmc_in0	0xb0020480
iop_dmc_in1	0xb0020500

25.16.1 rw_cfg

Address	0x0
Default	0x00000000
Type	Read/Write
Description	Configuration register for the DMA Communicator.

Bit(s)	Name	Description	Value
3	last_dis_dif	Select if the DMA Communicator shall not request more data bytes from the data interface when data, which is the last of the data stream, is received. r_stat.dif_en will then equal no. no: Continue receiving data after last yes: No request for data after last	no=0 yes=1
2-0:3	sth_intr	Selects the number of bytes which must be free in the DMA FIFO if the threshold interrupt should be generated. lim1: At least 1 byte of available buffer space lim2: At least 2 bytes of available buffer space lim4: At least 4 bytes of available buffer space lim8: At least 8 bytes of available buffer space lim16: At least 16 bytes of available buffer space lim32: At least 32 bytes of available buffer space lim64: At least 64 bytes of available buffer space	lim1=0 lim2=1 lim4=2 lim8=3 lim16=4 lim32=5 lim64=6

25.16.2 rw_ctrl

Address	0x4
Default	
Type	Read/Write
Description	Control register for the DMA Communicator.

Bit(s)	Name	Description	Value
2	stream_clr	Select if all data in the current data stream written to the DMA since the last saved context descriptor shall be discarded or not. yes must not be written if a context descriptor is not declared. no: Keep data yes: Clear data	no=0 yes=1
1	dif_dis	Disable the data interface (DIF) for input data. no: The state of DIF will not change yes: Disable DIF	no=0 yes=1
0	dif_en	Enable the data interface (DIF) for input data. This field will not have any effect if dif_dis is at the same time set to yes. no: The state of DIF will not change yes: Enable DIF	no=0 yes=1

25.16.3 r_stat

Address	0x8
Default	
Type	Read
Description	DMC status register.

Bit(s)	Name	Description	Value
0	dif_en	Shows if the data interface is currently enabled or disabled. no: Disabled yes: Enabled	no=0 yes=1

25.16.4 rw_stream_cmd

Address	0xc
Default	
Type	Read/Write
Description	Command register for the DMA in-channel. Before writing to this register you should check if it is ready to use by reading r_stream_stat.cmd_rdy. The command will not be executed until r_stream_stat.stream_busy equals no.

Bit(s)	Name	Description	Value
23-16:8	n	Offset parameter used by the command load_c_n in cmd.	

9-0:10	cmd	Write a command to the DMA. The constants with a description beginning with (Cmd) are commands and those beginning with (Opt) are options. For more information about the commands and their options read section 5.5.5.2. store_descr: (Cmd) Store descriptor (nop) store_md: (Opt) Store meta data store_c: (Opt) Store context descriptor store_g: (Opt) Store group descriptor array: (Opt) When next_en, stop at eol next_en: (Opt) Find the next enabled descriptor copy_next: (Opt) Make the down pointer of the upper group, point to the current next group. next_pkt: (Opt) Go to the first data descriptor of the next packet dis_c: (Opt) Disable the current context descriptor dis_g: (Opt) Disable the current group descriptor copy_up: (Opt) Copy current up pointer when loading the next group descriptor save_down: (Opt) Make the down pointer of the upper group, point to current group save_up: (Opt) Make the up pointer of the lower group, point to current group update_down: (Opt) Make the down pointer of the group point to the loaded context restore: (Opt) Restore context descriptor burst: (Opt) Start bursting set_reg: (Cmd) Set reg ack_pkt: (Cmd) Acknowledge packet(continue burst) load_d: (Cmd) Load data descriptor set_w_size1: (Cmd) Set word size to 1 byte set_w_size2: (Cmd) Set word size to 2 bytes set_w_size4: (Cmd) Set word size to 4 bytes load_c: (Cmd) Load context descriptor load_c_next: (Cmd) Load next context descriptor load_c_n: (Cmd) Load n:th context descriptor load_g: (Cmd) Load group descriptor load_g_next: (Cmd) Load next group descriptor load_g_up: (Cmd) Load upper group descriptor load_g_down: (Cmd) Load lower group descriptor	store_descr=0 store_md=1 store_c=2 store_g=4 array=8 next_en=16 copy_next=16 next_pkt=16 dis_c=16 dis_g=32 copy_up=32 save_down=32 save_up=32 update_down=32 restore=32 burst=32 set_reg=80 ack_pkt=256 load_d=320 set_w_size1=400 set_w_size2=416 set_w_size4=448 load_c=512 load_c_next=576 load_c_n=640 load_g=768 load_g_next=832 load_g_up=896 load_g_down=960

25.16.5 rw_stream_wr_data

Address	0x10
Default	
Type	Read/Write
Description	Data to write to the DMA in-channel. Before writing to this register, you should make sure r_stat.dif_en is set to no and check if the DMA FIFO can receive more bytes by reading r_stream_stat.full. The register should contain 1-4 valid bytes, depending on how large the transfer size was set by the DMA command field, rw_stream_cmd.cmd.

25.16.6 rw_stream_wr_data_last

Address	0x14
Default	
Type	Read/Write
Description	Data to write to the DMA in-channel which contains the last byte of the current data stream. Before writing to this register, you should make sure r_stat.dif_en is set to no and check if the DMA FIFO can receive more bytes by reading r_stream_stat.full. The register should contain 1-4 valid bytes, depending on how large the transfer size was set in rw_stream_ctrl.size.

25.16.7 rw_stream_ctrl

Address	0x18
Default	0x00000000
Type	Read/Write
Description	rw_stream_ctrl is valid only when the data word, which is about to be sent on the streaming interface, contains the last byte of the data stream.

Bit(s)	Name	Description	Value
5-3:3	size	The size of the last word of the current data stream. Will only be used when transferring data through rw_stream_wr_data_last.	
2	keep_md	Select if the source of meta data written to the data descriptor should be rw_data_descr.md or r_data_descr.md. no: Write new data from rw_data_descr.md yes: Keep the last read meta data (r_data_descr.md)	no=0 yes=1
1	wait	Select if you want the DMA channel to stop and wait for a DMA command. no: The DMA will not stop yes: The DMA will stop and wait for a DMA command	no=0 yes=1
0	eop	Select if the last byte of the packet is about to be sent to the DMA. no: Not end of packet yes: End of packet	no=0 yes=1

25.16.8 r_stream_stat

Address	0x1c
Default	
Type	Read
Description	Status register for the streaming interface.

Bit(s)	Name	Description	Value
22	cmd_rdy	Shows if rw_stream_cmd contains a command which has not yet been executed. A command written to rw_stream_cmd will be executed when stream_busy equals no. no: rw_stream_cmd already contains a command yes: A command can be written to rw_stream_cmd	no=0 yes=1
21	stream_busy	Shows if the DMA can or can not receive commands at the moment. no: DMA is ready to execute commands yes: DMA can not handle commands at the moment	no=0 yes=1
20	group_md_valid	Set when meta data from the group descriptor is valid. no: Meta data from the group descriptor is not valid yes: Meta data from the group descriptor is valid	no=0 yes=1
19	ctxt_md_valid	Set when meta data from the context descriptor is valid. no: Meta data from the context descriptor is not valid yes: Meta data from the context descriptor is valid	no=0 yes=1
18	data_md_valid	Set when meta data from the data descriptor is valid. no: Meta data from the data descriptor is not valid yes: Meta data from the data descriptor is valid	no=0 yes=1
17	last_pkt	The current packet is the last packet the DMA buffer can receive. The next data written which is marked with last will make the DMA buffer full. In other words, the field full will then be set to yes. no: There is zero or one packet in the DMA FIFO yes: There are two packets in the DMA FIFO	no=0 yes=1
16	full	Set when the DMA can not receive more data bytes. no: There is available buffer space in the DMA yes: DMA FIFO is full	no=0 yes=1
6-0:7	sth	Space threshold. When bit n is set there are 2^n or more bytes of available buffer space in the DMA for immediate writing.	

25.16.9 r_data_descr

Address	0x20
Default	
Type	Read
Description	Data descriptor. Before reading this register, you must check if the descriptor is valid by reading r_stream_stat.data_md_valid. r_data_descr is updated whenever r_stream_stat.data_md_valid equals yes

Bit(s)	Name	Description	Value
31-16:16	md	Meta data from the data descriptor.	
15-8:8	stat	Status from the data descriptor.	
7-0:8	ctrl	Control from the data descriptor.	

25.16.10 r_ctxt_descr

Address	0x24
Default	
Type	Read
Description	Context descriptor. Before reading this register, you must check if the descriptor is valid by reading r_stream_stat.ctxt_md_valid. r_ctxt_descr is updated whenever r_stream_stat.ctxt_md_valid equals yes.

Bit(s)	Name	Description	Value
31-16:16	md0	Meta data bit 15-0 from the context descriptor.	
15-8:8	stat	Status from the context descriptor.	
7-0:8	ctrl	Control from the context descriptor.	

25.16.11 r_ctxt_descr_md1

Address	0x28
Default	
Type	Read
Description	Meta data bit 47-16 from the context descriptor. r_ctxt_descr_md1 is updated whenever r_stream_stat.ctxt_md_valid equals yes.

25.16.12 r_ctxt_descr_md2

Address	0x2c
Default	
Type	Read
Description	Meta data bit 79-48 from the context descriptor. r_ctxt_descr_md2 is updated whenever r_stream_stat.ctxt_md_valid equals yes.

25.16.13 r_group_descr

Address	0x38
Default	
Type	Read
Description	Group descriptor. Before reading this register, you must check if the descriptor is valid by reading r_stream_stat.group_md_valid. r_group_descr is updated whenever r_stream_stat.group_md_valid equals yes.

Bit(s)	Name	Description	Value
31-16:16	md	Meta data from the group descriptor.	
15-8:8	stat	Status from the group descriptor.	
7-0:8	ctrl	Control from the group descriptor.	

25.16.14 rw_data_descr

Address	0x3c
Default	0x00000000
Type	Read/Write
Description	Data to be written to the data descriptor. rw_data_descr is used when a command, rw_stream_cmd.cmd, contains the save data descriptor flag.

Bit(s)	Name	Description	Value
31-16:16	md	Meta data to be written to the data descriptor.	

25.16.15 rw_ctxt_descr

Address	0x40
Default	0x00000000
Type	Read/Write
Description	Data to be written to the context descriptor. rw_ctxt_descr is used when a command, rw_stream_cmd.cmd, contains the save context descriptor flag.

Bit(s)	Name	Description	Value
31-16:16	md0	Meta data bit 15-0 to be written to the context descriptor.	

25.16.16 rw_ctxt_descr_md1

Address	0x44
Default	0x00000000
Type	Read/Write
Description	Meta data bit 47-16 to be written to the context descriptor. rw_ctxt_descr_md1 is used when a command, rw_stream_cmd.cmd, contains the save context descriptor flag.

25.16.17 rw_ctxt_descr_md2

Address	0x48
Default	0x00000000
Type	Read/Write
Description	Meta data bit 79-48 to be written to the context descriptor. rw_ctxt_descr_md2 is used when a command, rw_stream_cmd.cmd, contains the save context descriptor flag.

25.16.18 rw_group_descr

Address	0x54
Default	0x00000000
Type	Read/Write
Description	Data to be written to the group descriptor. rw_group_descr is used when a command, rw_stream_cmd.cmd, contains the save group descriptor flag.

Bit(s)	Name	Description	Value
31-16:16	md	Meta data to be written to the group descriptor.	

25.16.19 rw_intr_mask

Address	0x58
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts from DMA Communicator in-channel. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
5	full	Enable/disable full interrupt. Generate interrupt when r_stream_stat.full changes its value from no to yes. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	sth	Enable/disable sth interrupt. Generate interrupt if the number of free bytes in the DMA is equal or larger than the specified number in rw_cfg.sth_intr. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	cmd_rdy	Enable/disable cmd_rdy interrupt. Generate interrupt when r_stream_stat.cmd_rdy changes its value from no to yes. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	group_md	Enable/disable group_md interrupt. Generate interrupt when r_stream_stat.group_md_valid changes its value from no to yes. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	ctxt_md	Enable/disable ctxt_md interrupt. Generate interrupt when r_stream_stat.ctxt_md_valid changes its value from no to yes. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	data_md	Enable/disable data_md interrupt. Generate interrupt when r_stream_stat.data_md_valid changes its value from no to yes. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.16.20 rw_ack_intr

Address	0x5c
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts from DMA Communicator in-channel.

Bit(s)	Name	Description	Value
5	full	Acknowledge full interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
4	sth	Acknowledge sth interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
3	cmd_rdy	Acknowledge cmd_rdy interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	group_md	Acknowledge group_md interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	ctxt_md	Acknowledge ctxt_md interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	data_md	Acknowledge data_md interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.16.21 r_intr

Address	0x60
Default	
Type	Read
Description	Interrupts before the mask. Interrupts from DMA Communicator in-channel. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
5	full	Interrupt full active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
4	sth	Interrupt sth active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	cmd_rdy	Interrupt cmd_rdy active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	group_md	Interrupt group_md active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	ctxt_md	Interrupt ctxt_md active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	data_md	Interrupt data_md active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.16.22 r_masked_intr

Address	0x64
Default	
Type	Read
Description	Interrupts after the mask. Interrupts from DMA Communicator in-channel. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
5	full	Interrupt full active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
4	sth	Interrupt sth active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	cmd_rdy	Interrupt cmd_rdy active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	group_md	Interrupt group_md active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	ctxt_md	Interrupt ctxt_md active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	data_md	Interrupt data_md active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.17 iop_dmc_out

Instance	Base Address
iop_dmc_out0	0xb0020580
iop_dmc_out1	0xb0020600

25.17.1 rw_cfg

Address	0x0
Default	0x00000000
Type	Read/Write
Description	Configuration register for the DMA Communicator.

Bit(s)	Name	Description	Value
19-17:3	dth_intr	Selects the number of bytes which must be buffered by the DMA if the threshold interrupt should be generated. lim1: At least 1 byte of buffered data lim2: At least 2 bytes of buffered data lim4: At least 4 bytes of buffered data lim8: At least 8 bytes of buffered data lim16: At least 16 bytes of buffered data lim32: At least 32 bytes of buffered data lim64: At least 64 bytes of buffered data	lim1=0 lim2=1 lim4=2 lim8=3 lim16=4 lim32=5 lim64=6
16	last_at_trf_lim	When set, last is signaled on the data interface when the last word is read. no: Last is not signaled on trf_limit yes: Last is signaled	no=0 yes=1
15-0:16	trf_lim	The DMA Communicator will disable the data interface when it has transmitted a number of transfers, specified by trf_lim. An interrupt will then be generated and rw_ctrl.dif_en will change its value to no. This functionality will be disabled if the value is set to zero.	

25.17.2 rw_ctrl

Address	0x4
Default	
Type	Read/Write
Description	Control register for the DMA Communicator.

Bit(s)	Name	Description	Value
1	dif_dis	Disable the data interface (DIF) for output data. no: The state of DIF will not change yes: Disable	no=0 yes=1
0	dif_en	Enable the data interface (DIF) for output data. This field will not have any effect if dif_dis is at the same time set to yes. no: The state of DIF will not change yes: Enable	no=0 yes=1

25.17.3 r_stat

Address	0x8
Default	
Type	Read
Description	DMC status register.

Bit(s)	Name	Description	Value
0	dif_en	Shows if the data interface is enabled or disabled. No data will be sent on the interface when disabled. The interface can be enabled or disabled by rw_ctrl.dif_en and rw_ctrl.dif_dis. no: Disabled yes: Enabled	no=0 yes=1

25.17.4 rw_stream_cmd

Address	0xc
Default	
Type	Read/Write
Description	Command register for the DMA out-channel. Before writing to this register you should check if rw_stream_cmd is ready to use by reading r_stream_stat.cmd_rdy. The command will not be executed until r_stream_stat.stream_busy equals no.

Bit(s)	Name	Description	Value
23-16:8	n	Offset parameter used by the command load_c_n in cmd.	

9-0:10	cmd	Write a command to the DMA. The constants with a description beginning with (Cmd) are commands and those beginning with (Opt) are options. For more information about the commands and their options read section 5.5.5.2. store_descr: (Cmd) Store descriptor (nop) store_md: (Opt) Store meta data store_c: (Opt) Store context descriptor store_g: (Opt) Store group descriptor array: (Opt) When next_en, stop at eol next_en: (Opt) Find the next enabled descriptor copy_next: (Opt) Make the down pointer of the upper group, point to the current next group next_pkt: (Opt) Go to the first data descriptor of the next packet dis_c: (Opt) Disable the current context descriptor dis_g: (Opt) Disable the current group descriptor copy_up: (Opt) Copy current up pointer when loading the next group descriptor save_down: (Opt) Make the down pointer of the upper group, point to current group save_up: (Opt) Make the up pointer of the lower group, point to current group update_down: (Opt) Make the down pointer of the group point to the loaded context restore: (Opt) Restore context descriptor burst: (Opt) Start bursting set_reg: (Cmd) Set reg ack_pkt: (Cmd) Acknowledge packet(continue burst) load_d: (Cmd) Load data descriptor set_w_size1: (Cmd) Set word size to 1 byte set_w_size2: (Cmd) Set word size to 2 bytes set_w_size4: (Cmd) Set word size to 4 bytes load_c: (Cmd) Load context descriptor load_c_next: (Cmd) Load next context descriptor load_c_n: (Cmd) Load n:th context descriptor load_g: (Cmd) Load group descriptor load_g_next: (Cmd) Load next group descriptor load_g_up: (Cmd) Load upper group descriptor load_g_down: (Cmd) Load lower group descriptor	store_descr=0 store_md=1 store_c=2 store_g=4 array=8 next_en=16 copy_next=16 next_pkt=16 dis_c=16 dis_g=32 copy_up=32 save_down=32 save_up=32 update_down=32 restore=32 burst=32 set_reg=80 ack_pkt=256 load_d=320 set_w_size1=400 set_w_size2=416 set_w_size4=448 load_c=512 load_c_next=576 load_c_n=640 load_g=768 load_g_next=832 load_g_up=896 load_g_down=960

25.17.5 rs_stream_data/r_stream_data

Address	0x10/0x14
Default	
Type	Read with side effects/Read
Description	Data from the DMA out-channel. Before reading this register, you should check if data is valid by reading r_stream_stat.dv. Read r_stream_stat.size and r_stream_stat.last to find out the size of the read data and if it is marked as the last data. When rs_stream_data is read, the DMA Communicator will acknowledge the read to the DMA and new data from the DMA will be on the streaming interface. If there is no more data in the DMA FIFO, r_stream_stat.dv will be set to no.

25.17.6 r_stream_stat

Address	0x18
Default	
Type	Read
Description	Status register for the streaming interface.

Bit(s)	Name	Description	Value
27	cmd_rq	Command request. DMA is waiting for a command. Set after the last word is received by the DMA, if the wait bit in r_data_descr.ctrl is set. no: DMA is not waiting for a command yes: DMA has requested and is waiting for a command	no=0 yes=1
26	cmd_rdy	Shows if rw_stream_cmd contains a command which has not yet been executed. A command written to rw_stream_cmd will be executed when stream_busy equals no. no: rw_stream_cmd already contains a command yes: A command can be written to rw_stream_cmd	no=0 yes=1
25	stream_busy	Shows if the DMA can or can not receive commands at the moment. no: DMA is ready to execute commands yes: DMA can not handle commands at the moment	no=0 yes=1
24	group_md_valid	Set when meta data from the group descriptor is valid. no: Meta data from the group descriptor is not valid yes: Meta data from the group descriptor is valid	no=0 yes=1
23	ctxt_md_valid	Set when meta data from the context descriptor is valid. no: Meta data from the context descriptor is not valid yes: Meta data from the context descriptor is valid	no=0 yes=1
22	data_md_valid	Set when meta data from the data descriptor is valid. no: Meta data from the data descriptor is not valid yes: Meta data from the data descriptor is valid	no=0 yes=1
21-19:3	size	Size of the current data on the DMA out-channel (number of bytes).	
18	last	Set when the current data on the DMA out-channel is the last of the packet. no: Data word does not contain the last byte of data stream yes: Data word does contain the last byte of data stream	no=0 yes=1

17	all_avail	Is set when all data in the packet has been read from memory and it is available for immediate reading on the DMA out-channel. In other words there is a complete packet or the data left in the current packet is smaller than the data threshold. no: The whole packet is not available in the DMA FIFO yes: The whole packet is available in the DMA FIFO	no=0 yes=1
16	dv	Set when there is valid data on the DMA out-channel. no: Data from the stream interface is not valid yes: Data from the stream interface is valid	no=0 yes=1
6-0:7	dth	When bit n is set there are 2^n or more bytes of buffered data available for immediate reading on the DMA out-channel.	

25.17.7 r_data_descr

Address	0x1c
Default	
Type	Read
Description	Data descriptor. Before reading this register, you must check if descriptor is valid by reading r_stream_stat.data_md_valid. r_data_descr is updated whenever r_stream_stat.data_md_valid equals yes

Bit(s)	Name	Description	Value
31-16:16	md	Meta data from the data descriptor.	
15-8:8	stat	Status from the data descriptor.	
7-0:8	ctrl	Control from the data descriptor.	

25.17.8 r_ctxt_descr

Address	0x20
Default	
Type	Read
Description	Context descriptor. Before reading this register, you must check if the descriptor is valid by reading r_stream_stat.ctxt_md_valid. r_ctxt_descr is updated whenever r_stream_stat.ctxt_md_valid equals yes.

Bit(s)	Name	Description	Value
31-16:16	md0	Meta data bit 15-0 from the context descriptor.	
15-8:8	stat	Status from the context descriptor.	
7-0:8	ctrl	Control from the context descriptor.	

25.17.9 r_ctxt_descr_md1

Address	0x24
Default	
Type	Read
Description	Meta data bit 47-16 from the context descriptor. r_ctxt_descr_md1 is updated whenever r_stream_stat.ctxt_md_valid equals yes.

25.17.10 r_ctxt_descr_md2

Address	0x28
Default	
Type	Read
Description	Meta data bit 79-48 from the context descriptor. r_ctxt_descr_md2 is updated whenever r_stream_stat.ctxt_md_valid equals yes.

25.17.11 r_group_descr

Address	0x34
Default	
Type	Read
Description	Group descriptor. Before reading this register, you must check if the descriptor is valid by reading r_stream_stat.group_md_valid. r_group_descr is updated whenever r_stream_stat.group_md_valid equals yes.

Bit(s)	Name	Description	Value
31-16:16	md	Meta data from the group descriptor.	
15-8:8	stat	Status from the group descriptor.	
7-0:8	ctrl	Control from the group descriptor.	

25.17.12 rw_data_descr

Address	0x38
Default	0x00000000
Type	Read/Write
Description	Data to be written to the data descriptor. rw_data_descr is used when a command, rw_stream_cmd.cmd, uses the save data descriptor option store_md.

Bit(s)	Name	Description	Value
31-16:16	md	Meta data to be written to the data descriptor.	

25.17.13 rw_ctxt_descr

Address	0x3c
Default	0x00000000
Type	Read/Write
Description	Data to be written to the context descriptor. rw_ctxt_descr is used when a command, rw_stream_cmd.cmd, uses the save data descriptor option store_c.

Bit(s)	Name	Description	Value
31-16:16	md0	Meta data bit 15-0 to be written to the context descriptor.	

25.17.14 rw_ctxt_descr_md1

Address	0x40
Default	0x00000000
Type	Read/Write
Description	Meta data bit 47-16 to be written to the context descriptor. rw_ctxt_descr_md1 is used when a command, rw_stream_cmd.cmd, uses the save data descriptor option store_c.

25.17.15 rw_ctxt_descr_md2

Address	0x44
Default	0x00000000
Type	Read/Write
Description	Meta data bit 79-48 to be written to the context descriptor. rw_ctxt_descr_md2 is used when a command, rw_stream_cmd.cmd, uses the save data descriptor option store_c.

25.17.16 rw_group_descr

Address	0x50
Default	0x00000000
Type	Read/Write
Description	Data to be written to the group descriptor. rw_group_descr is used when a command, rw_stream_cmd.cmd, uses the save data descriptor option store_g.

Bit(s)	Name	Description	Value
31-16:16	md	Meta data to be written to the group descriptor.	

25.17.17 rw_intr_mask

Address	0x54
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts from DMA Communicator out-channel. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
8	cmd_rq	Enable/disable cmd_rq interrupt. Generate interrupt when r_stream_stat.cmd_rq changes its value from no to yes. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
7	trf_lim	Enable/disable trf_lim interrupt. Generate interrupt when the DMA Communicator automatically disables the data interface after counted a specified number of transfers, configured in rw_cfg.trf_lim. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
6	last_data	Enable/disable last_data interrupt. Generate interrupt if data from the streaming interface is the last of the packet. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
5	dv	Enable/disable dv interrupt. Generate interrupt if data from the streaming interface is valid. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	dth	Enable/disable dth interrupt. Generate interrupt if the number of bytes buffered in the DMA is equal or larger than the specified number in rw_cfg.dth_intr. The interrupt is also generated if all data in the packet has been read from memory and is available for immediate reading on the streaming interface. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	cmd_rdy	Enable/disable cmd_rdy interrupt. Generate interrupt when r_stream_stat.cmd_rdy changes its value from no to yes. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

2	group_md	Enable/disable group_md interrupt. Generate interrupt when r_stream_stat.group_md_valid changes its value from no to yes. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	ctxt_md	Enable/disable ctxt_md interrupt. Generate interrupt when r_stream_stat.ctxt_md_valid changes its value from no to yes. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	data_md	Enable/disable data_md interrupt. Generate interrupt when r_stream_stat.data_md_valid changes its value from no to yes. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.17.18 rw_ack_intr

Address	0x58
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts from DMA Communicator out-channel.

Bit(s)	Name	Description	Value
8	cmd_rq	Acknowledge cmd_rq interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
7	trf_lim	Acknowledge trf_lim interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
6	last_data	Acknowledge last_data interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
5	dv	Acknowledge dv interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
4	dth	Acknowledge dth interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
3	cmd_rdy	Acknowledge cmd_rdy interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	group_md	Acknowledge group_md interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	ctxt_md	Acknowledge ctxt_md interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	data_md	Acknowledge data_md interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.17.19 r_intr

Address	0x5c
Default	
Type	Read
Description	Interrupts before the mask. Interrupts from DMA Communicator out-channel. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
8	cmd_rq	Interrupt cmd_rq active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	trf_lim	Interrupt trf_lim active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	last_data	Interrupt last_data active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	dv	Interrupt dv active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
4	dth	Interrupt dth active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	cmd_rdy	Interrupt cmd_rdy active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	group_md	Interrupt group_md active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	ctxt_md	Interrupt ctxt_md active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	data_md	Interrupt data_md active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.17.20 r_masked_intr

Address	0x60
Default	
Type	Read
Description	Interrupts after the mask. Interrupts from DMA Communicator out-channel. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
8	cmd_rq	Interrupt cmd_rq active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	trf_lim	Interrupt trf_lim active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	last_data	Interrupt last_data active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	dv	Interrupt dv active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
4	dth	Interrupt dth active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	cmd_rdy	Interrupt cmd_rdy active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	group_md	Interrupt group_md active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	ctxt_md	Interrupt ctxt_md active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	data_md	Interrupt data_md active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.18 iop_fifo_out

Instance	Base Address
iop_fifo_out0	0xb0020780
iop_fifo_out1	0xb0020800

25.18.1 rw_cfg

Address	0x0
Default	0x00000024
Type	Read/Write
Description	Configuration register for FIFO. The FIFO will be reset when writing to this register, i.e. the FIFO will be emptied.

Bit(s)	Name	Description	Value
11	last_dis_dif_out	If set the dif_out port of the FIFO will be disabled after last word is sent. no: The FIFO will not be disabled yes: The FIFO will be disabled	no=0 yes=1
10	delay_out_last	Selects if the dif_out last signal will be delayed 5ns or not, relative to data. no: Do not delay last yes: Delay last 5 ns	no=0 yes=1
9-8:2	mode	Select mode of operation for FIFO as well as the width of the DIF-buses. size32: Configure FIFO as 32-bit FIFO. Output is 32-bits wide size24: Configure FIFO as 24-bit FIFO. Output is 24-bits wide size16: Configure FIFO as 16-bit FIFO. Output is 16-bits wide size8: Configure FIFO as 8-bit FIFO. Output is 8-bits wide	size32=0 size24=1 size16=2 size8=3
7	last_dis_dif_in	If set the dif_in port of the FIFO will be disabled after the last word is received. no: The FIFO will not be disabled yes: The FIFO will be disabled	no=0 yes=1

6-5:2	trig	Select if the strobe for FIFO output should be triggered when it is high or on positive or negative edge. hi: High strobe pos: Positive edge of strobe neg: Negative edge of strobe pos_neg: Both positive and negative edge of strobe	hi=0 pos=1 neg=2 pos_neg=3
4-3:2	byte_order	Controls the byte swapping mechanism. order8: 8-bit mode (3 2 1 0) order16: 16-bit mode (2 3 0 1) order24: 24-bit mode (3 0 1 2) order32: 32-bit mode (0 1 2 3)	order8=0 order16=1 order24=2 order32=3
2-0:3	free_lim	Set the number of bytes which must be free in the FIFO before the FIFO will generate an interrupt or signal the parallel in-data interface that it can receive data.	

25.18.2 rw_ctrl

Address	0x4
Default	0x00000000
Type	Read/Write
Description	FIFO command register.

Bit(s)	Name	Description	Value
1	dif_out_en	Enable/disable the output strobe of the FIFO no: Disabled yes: Enabled	no=0 yes=1
0	dif_in_en	Enable/disable the input strobe of the FIFO no: Disabled yes: Enabled	no=0 yes=1

25.18.3 r_stat

Address	0x8
Default	
Type	Read
Description	Status of FIFO.

Bit(s)	Name	Description	Value
14	zero_data_last	Indicates if a last-mark is, or has been, present in an empty FIFO no: Last mark is not present yes: Last mark is present	no=0 yes=1
13	dif_out_en	Status of the FIFOs output strobe enable bit. See rw_ctrl.dif_out_en. no: FIFO is disabled yes: FIFO is enabled	no=0 yes=1
12	dif_in_en	Status of the FIFOs input strobe enable bit. See rw_ctrl.dif_in_en. no: FIFO is disabled yes: FIFO is enabled	no=0 yes=1
11-4:8	last	Vector with last marks for the bytes in FIFO. The least significant bit (lsb) holds the last mark for first byte to be shifted out.	
3-0:4	avail_bytes	Number of bytes currently available in the FIFO.	

25.18.4 rw_wr1byte

Address	0xc
Default	0x00000000
Type	Read/Write
Description	Write one byte of data to FIFO. Only the 8 least significant bits of the register are used. NOTE: To avoid overrun, field r_stat.avail_bytes should be checked.

Bit(s)	Name	Description	Value
7-0:8	data	Payload data.	

25.18.5 rw_wr2byte

Address	0x10
Default	0x00000000
Type	Read/Write
Description	Write two bytes of data to FIFO. Only the 16 least significant bits of the register are used. NOTE: To avoid overrun, field r_stat.avail_bytes should be checked.

Bit(s)	Name	Description	Value
15-0:16	data	Payload data.	

25.18.6 rw_wr3byte

Address	0x14
Default	0x00000000
Type	Read/Write
Description	Write three bytes of data to FIFO. Only the 24 least significant bits of the register are used. NOTE: To avoid overrun, field r_stat.avail_bytes should be checked.

Bit(s)	Name	Description	Value
23-0:24	data	Payload data.	

25.18.7 rw_wr4byte

Address	0x18
Default	0x00000000
Type	Read/Write
Description	Write four bytes of data to the FIFO. NOTE: To avoid overrun, field r_stat.avail_bytes should be checked.

Bit(s)	Name	Description	Value
31-0:32	data	Payload data.	

25.18.8 rw_wr1byte_last

Address	0x1c
Default	0x00000000
Type	Read/Write
Description	Write one byte of data to FIFO and set last flag for that byte. Only the 8 least significant bits of the register are used. NOTE: To avoid overrun, field r_stat.avail_bytes should be checked.

Bit(s)	Name	Description	Value
7-0:8	data	Payload data.	

25.18.9 rw_wr2byte_last

Address	0x20
Default	0x00000000
Type	Read/Write
Description	Write two bytes of data to FIFO and set the last flag for that byte. Only the 16 least significant bits of the register are used. NOTE: To avoid overrun, field r_stat.avail_bytes should be checked.

Bit(s)	Name	Description	Value
15-0:16	data	Payload data.	

25.18.10 rw_wr3byte_last

Address	0x24
Default	0x00000000
Type	Read/Write
Description	Write three bytes of data to FIFO and set the last flag. Only the 24 least significant bits of the register are used. NOTE: To avoid overrun, field r_stat.avail_bytes should be checked.

Bit(s)	Name	Description	Value
23-0:24	data	Payload data.	

25.18.11 rw_wr4byte_last

Address	0x28
Default	0x00000000
Type	Read/Write
Description	Write four bytes of data to the FIFO and set the last flag. NOTE: To avoid overrun, field r_stat.avail_bytes should be checked.

Bit(s)	Name	Description	Value
31-0:32	data	Payload data.	

25.18.12 rw_set_last

Address	0x2c
Default	0x00000000
Type	Read/Write
Description	Mark the last byte in the FIFO with last.

25.18.13 rs_rd_data/r_rd_data

Address	0x30/0x34
Default	
Type	Read with side effects/Read
Description	Reads a word from the FIFO. The FIFO will place one to four bytes in the register, so the owner must be aware of the current FIFO configuration. Reading this register will advance the fifo read-pointer by a value corresponding to the FIFOs current configuration i.e. one to four bytes.

25.18.14 rw_strb_dif_out

Address	0x38
Default	0x00000000
Type	Read/Write
Description	Generate a strobe at the input-channel of the FIFO. Forces the fifo to read data. The number of bytes read corresponds to the width of the FIFO.

25.18.15 rw_intr_mask

Address	0x3c
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts generated by generic FIFO. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	orun	Enable/disable orun interrupt. FIFO overrun interrupt. Generated when an attempt to write data beyond the end of the FIFO is made. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	free	Enable/disable free interrupt. Free interrupt. Generated when the amount of free space available exceeds the amount specified in rw_cfg.free_lim. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	dav	Enable/disable dav interrupt. Data available interrupt. Generated when data the amount of data available exceeds the number specified in rw_cfg.mode. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	last_data	Enable/disable last_data interrupt. Last interrupt. Generated if a byte with last-flag is received. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	urun	Enable/disable urun interrupt. FIFO underrun interrupt. Generated if the FIFO is read through the register interfaces while empty. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.18.16 rw_ack_intr

Address	0x40
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts generated by generic FIFO.

Bit(s)	Name	Description	Value
4	orun	Acknowledge orun interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
3	free	Acknowledge free interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	dav	Acknowledge dav interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	last_data	Acknowledge last_data interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	urun	Acknowledge urun interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.18.17 r_intr

Address	0x44
Default	
Type	Read
Description	Interrupts before the mask. Interrupts generated by generic FIFO. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	orun	Interrupt orun active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	free	Interrupt free active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	dav	Interrupt dav active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	last_data	Interrupt last_data active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	urun	Interrupt urun active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.18.18 r_masked_intr

Address	0x48
Default	
Type	Read
Description	Interrupts after the mask. Interrupts generated by generic FIFO. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	orun	Interrupt orun active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	free	Interrupt free active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	dav	Interrupt dav active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	last_data	Interrupt last_data active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	urun	Interrupt urun active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.19 iop_fifo_in

Instance	Base Address
iop_fifo_in0	0xb0020680
iop_fifo_in1	0xb0020700

25.19.1 rw_cfg

Address	0x0
Default	0x00000024
Type	Read/Write
Description	Configuration register for FIFO. The FIFO will be reset when writing to this register, i.e. the FIFO will be emptied.

Bit(s)	Name	Description	Value
9-8:2	mode	Select mode of operation for FIFO as well as the width of the DIF-buses. size32: Configure FIFO as 32-bit FIFO. Output is 32-bits wide size24: Configure FIFO as 24-bit FIFO. Output is 24-bits wide size16: Configure FIFO as 16-bit FIFO. Output is 16-bits wide size8: Configure FIFO as 8-bit FIFO. Output is 8-bits wide	size32=0 size24=1 size16=2 size8=3
7	last_dis_dif_in	If set the dif_in port of the FIFO will be disabled after the last word is received. no: The FIFO will not be disabled yes: The FIFO will be disabled	no=0 yes=1
6-5:2	trig	Select if the strobe for FIFO output should be triggered when it is high or on positive or negative edge. hi: High strobe pos: Positive edge of strobe neg: Negative edge of strobe pos_neg: Both positive and negative edge of strobe	hi=0 pos=1 neg=2 pos_neg=3
4-3:2	byte_order	Controls the byte swapping mechanism. order8: 8-bit mode (3 2 1 0) order16: 16-bit mode (2 3 0 1) order24: 24-bit mode (3 0 1 2) order32: 32-bit mode (0 1 2 3)	order8=0 order16=1 order24=2 order32=3
2-0:3	avail_lim	Set the number of bytes which must be available in the FIFO before the FIFO will signal the DMC-in that it has data to send.	

25.19.2 rw_ctrl

Address	0x4
Default	0x00000000
Type	Read/Write
Description	FIFO command register.

Bit(s)	Name	Description	Value
1	dif_out_en	Enable/disable the output strobe of the FIFO no: Disabled yes: Enabled	no=0 yes=1
0	dif_in_en	Enable/disable the input strobe of the FIFO no: Disabled yes: Enabled	no=0 yes=1

25.19.3 r_stat

Address	0x8
Default	
Type	Read
Description	Status of FIFO.

Bit(s)	Name	Description	Value
13	dif_out_en	Status of the FIFOs output strobe enable bit. See rw_ctrl.dif_out_en. no: FIFO is disabled yes: FIFO is enabled	no=0 yes=1
12	dif_in_en	Status of the FIFOs input strobe enable bit. See rw_ctrl.dif_in_en. no: FIFO is disabled yes: FIFO is enabled	no=0 yes=1
11-4:8	last	Vector with last marks for the bytes in FIFO. The least significant bit (lsb) holds the last mark for first byte to be shifted out.	
3-0:4	avail_bytes	Number of bytes currently available in the FIFO.	

25.19.4 rs_rd1byte/r_rd1byte

Address	0xc/0x10
Default	
Type	Read with side effects/Read
Description	Read one byte of data from the FIFO. Number of available bytes must be checked in r_stat.avail_bytes. The side-effect of reading this register is that the read-pointer is advanced by one byte.

Bit(s)	Name	Description	Value
7-0:8	data	Payload data.	

25.19.5 rs_rd2byte/r_rd2byte

Address	0x14/0x18
Default	
Type	Read with side effects/Read
Description	Read two bytes of data from the FIFO. Number of available bytes must be checked in r_stat.avail_bytes. The side-effect of reading this register is that the read-pointer is advanced by two bytes.

Bit(s)	Name	Description	Value
15-0:16	data	Payload data.	

25.19.6 rs_rd3byte/r_rd3byte

Address	0x1c/0x20
Default	
Type	Read with side effects/Read
Description	Read three bytes of data from the FIFO. Number of available bytes must be checked in r_stat.avail_bytes. The side-effect of reading this register is that the read-pointer is advanced by three bytes.

Bit(s)	Name	Description	Value
23-0:24	data	Payload data.	

25.19.7 rs_rd4byte/r_rd4byte

Address	0x24/0x28
Default	
Type	Read with side effects/Read
Description	Read four bytes of data from the FIFO. Number of available bytes must be checked in r_stat.avail_bytes. The side-effect of reading this register is that the read-pointer is advanced by four bytes.

Bit(s)	Name	Description	Value
31-0:32	data	Payload data.	

25.19.8 rw_set_last

Address	0x2c
Default	0x00000000
Type	Read/Write
Description	Mark the last byte in the FIFO with last.

25.19.9 rw_strb_dif_in

Address	0x30
Default	0x00000000
Type	Read/Write
Description	Generate a strobe at the input-channel of the FIFO. Forces the fifo to read data. The number of bytes read corresponds to the width of the FIFO.

Bit(s)	Name	Description	Value
1-0:2	last	Select source used to define value of the last-bit. no: Do not set the last flag yes: Set last flag dif_in: Assign value from DIF	no=0 yes=1 dif_in=2

25.19.10 rw_intr_mask

Address	0x34
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts generated by generic FIFO. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	orun	Enable/disable orun interrupt. FIFO overrun interrupt. Generated when an attempt to write data beyond the end of the FIFO is made. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	avail	Enable/disable avail interrupt. Avail interrupt. Generated when the number of available bytes exceed the amount specified in rw_cfg.avail_lim. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	dav	Enable/disable dav interrupt. Data available interrupt. Generated when the amount of data available exceeds the number specified in rw_cfg.mode. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	last_data	Enable/disable last_data interrupt. Last interrupt. Generated if a byte with last-flag is received. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	urun	Enable/disable urun interrupt. FIFO underrun interrupt. Generated if the FIFO is read through the register interfaces while empty. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.19.11 rw_ack_intr

Address	0x38
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts generated by generic FIFO.

Bit(s)	Name	Description	Value
4	orun	Acknowledge orun interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
3	avail	Acknowledge avail interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	dav	Acknowledge dav interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	last_data	Acknowledge last_data interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	urun	Acknowledge urun interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.19.12 r_intr

Address	0x3c
Default	
Type	Read
Description	Interrupts before the mask. Interrupts generated by generic FIFO. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	orun	Interrupt orun active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	avail	Interrupt avail active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	dav	Interrupt dav active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	last_data	Interrupt last_data active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	urun	Interrupt urun active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.19.13 r_masked_intr

Address	0x40
Default	
Type	Read
Description	Interrupts after the mask. Interrupts generated by generic FIFO. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	orun	Interrupt orun active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	avail	Interrupt avail active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	dav	Interrupt dav active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	last_data	Interrupt last_data active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	urun	Interrupt urun active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.20 iop_fifo_out_extra

Instance	Base Address
iop_fifo_out0_extra	0xb00200c0
iop_fifo_out1_extra	0xb0020100

25.20.1 rs_rd_data/r_rd_data

Address	0x0/0x4
Default	
Type	Read with side effects/Read
Description	Reads data from the FIFO. The FIFO will place one to four bytes in the register, so the owner must be aware of the current FIFO configuration. Reading this register will advance the fifo read-pointer by a value corresponding to the FIFOs current configuration i.e. one to four bytes.

25.20.2 r_stat

Address	0x8
Default	
Type	Read
Description	FIFO status register.

Bit(s)	Name	Description	Value
14	zero_data_last	Indicates if a last-mark is, or has been, present in an empty FIFO. no: Last mark is not present yes: Last mark is present	no=0 yes=1
13	dif_out_en	Status of the FIFOs output strobe enable bit. no: FIFO is disabled yes: FIFO is enabled	no=0 yes=1
12	dif_in_en	Status of the FIFOs input strobe enable bit. no: FIFO is disabled yes: FIFO is enabled	no=0 yes=1
11-4:8	last	Vector with last marks for the bytes in FIFO. The least significant bit (lsb) holds the last mark for first byte to be shifted out.	
3-0:4	avail_bytes	Number of bytes currently available in the FIFO.	

25.20.3 rw_strb_dif_out

Address	0xc
Default	
Type	Read/Write
Description	Generate a strobe at the input-channel of the FIFO. Forces the fifo to read data. The number of bytes read corresponds to the width of the FIFO.

25.20.4 rw_intr_mask

Address	0x10
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts generated by generic FIFO. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	orun	Enable/disable orun interrupt. FIFO overrun interrupt. Generated when an attempt to write data beyond the end of the FIFO is made. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	free	Enable/disable free interrupt. Free interrupt. Generated when the amount of free space available exceeds the amount specified in rw_cfg.free_lim. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	dav	Enable/disable dav interrupt. Data available interrupt. Generated when the amount of data available exceeds the number specified in: rw_cfg.mode. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	last_data	Enable/disable last_data interrupt. Last interrupt. Generated if a byte with last-flag is received. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	urun	Enable/disable urun interrupt. FIFO underrun interrupt. Generated if the FIFO is read through the register interfaces while empty. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.20.5 rw_ack_intr

Address	0x14
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts generated by generic FIFO.

Bit(s)	Name	Description	Value
4	orun	Acknowledge orun interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
3	free	Acknowledge free interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	dav	Acknowledge dav interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	last_data	Acknowledge last_data interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	urun	Acknowledge urun interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.20.6 r_intr

Address	0x18
Default	
Type	Read
Description	Interrupts before the mask. Interrupts generated by generic FIFO. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	orun	Interrupt orun active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	free	Interrupt free active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	dav	Interrupt dav active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	last_data	Interrupt last_data active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	urun	Interrupt urun active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.20.7 r_masked_intr

Address	0x1c
Default	
Type	Read
Description	Interrupts after the mask. Interrupts generated by generic FIFO. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	orun	Interrupt orun active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	free	Interrupt free active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	dav	Interrupt dav active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	last_data	Interrupt last_data active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	urun	Interrupt urun active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.21 iop_fifo_in_extra

Instance	Base Address
iop_fifo_in0_extra	0xb0020040
iop_fifo_in1_extra	0xb0020080

25.21.1 rw_wr_data

Address	0x0
Default	
Type	Read/Write
Description	Writes data to the FIFO. The FIFO will read one to four bytes from the register, so the owner must be aware of the current FIFO configuration. Reading this register returns undefined data.

25.21.2 r_stat

Address	0x4
Default	
Type	Read
Description	FIFO status register.

Bit(s)	Name	Description	Value
13	dif_out_en	Status of the FIFOs output strobe enable bit. no: FIFO is disabled yes: FIFO is enabled	no=0 yes=1
12	dif_in_en	Status of the FIFOs input strobe enable bit. no: FIFO is disabled yes: FIFO is enabled	no=0 yes=1
11-4:8	last	Vector with last marks for the bytes in FIFO. The least significant bit (lsb) holds the last mark for first byte to be shifted out.	
3-0:4	avail_bytes	Number of bytes currently available in the FIFO.	

25.21.3 rw_strb_dif_in

Address	0x8
Default	
Type	Read/Write
Description	Generate a strobe at the input-channel of the FIFO. Forces the fifo to read data. The number of bytes read corresponds to the width of the FIFO.

Bit(s)	Name	Description	Value
1-0:2	last	Select source used to define value of the last-bit. no: Do not set the last flag yes: Set last flag fifo_in: Use value stored in FIFO	no=0 yes=1 fifo_in=2

25.21.4 rw_intr_mask

Address	0xc
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts generated by generic FIFO. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	orun	Enable/disable orun interrupt. FIFO overrun interrupt. Generated when an attempt to write data beyond the end of the FIFO is made. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	avail	Enable/disable avail interrupt. Avail interrupt. Generated when the number of available bytes exceeds the amount specified in rw_cfg.avail_lim. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	dav	Enable/disable dav interrupt. Data available interrupt. Generated when the amount of data available exceeds the number specified in: rw_cfg.mode. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	last_data	Enable/disable last_data interrupt. Last interrupt. Generated if a byte with last-flag is received. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	urun	Enable/disable urun interrupt. FIFO underrun interrupt. Generated if the FIFO is read through the register interfaces while empty. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.21.5 rw_ack_intr

Address	0x10
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts generated by generic FIFO.

Bit(s)	Name	Description	Value
4	orun	Acknowledge orun interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
3	avail	Acknowledge avail interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	dav	Acknowledge dav interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	last_data	Acknowledge last_data interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	urun	Acknowledge urun interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.21.6 r_intr

Address	0x14
Default	
Type	Read
Description	Interrupts before the mask. Interrupts generated by generic FIFO. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	orun	Interrupt orun active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	avail	Interrupt avail active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	dav	Interrupt dav active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	last_data	Interrupt last_data active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	urun	Interrupt urun active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.21.7 r_masked_intr

Address	0x18
Default	
Type	Read
Description	Interrupts after the mask. Interrupts generated by generic FIFO. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
4	orun	Interrupt orun active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	avail	Interrupt avail active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	dav	Interrupt dav active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	last_data	Interrupt last_data active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	urun	Interrupt urun active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.22 iop_mpu

Instance	Base Address
iop_mpu	0xb0021600

25.22.1 rw_r

Address	0x0, 0x4, 0x8, 0xc, 0x10, 0x14, 0x18, 0x1c, 0x20, 0x24, 0x28, 0x2c, 0x30, 0x34, 0x38, 0x3c
Default	
Type	Read/Write
Description	MPU general register. If data is written to the register then the corresponding bit in r_wr_stat is set.

25.22.2 rw_ctrl

Address	0x80
Default	0x00000000
Type	Read/Write
Description	The MPU control register.

Bit(s)	Name	Description	Value
0	en	The function of this field is to enable the MPU. The en field must not be used for disabling the MPU. Disabling the MPU is done by writing the HALT instruction to the rw_instr register. When the MPU is disabled, no micro code is executed except for the instruction in rw_instr. no: Disable the MPU yes: Enable the MPU	no=0 yes=1

25.22.3 r_pc

Address	0x84
Default	0x00000000
Type	Read
Description	The MPU program counter.

Bit(s)	Name	Description	Value
11-0:12	addr	The program counter.	

25.22.4 r_stat

Address	0x88
Default	
Type	Read
Description	The MPU status register.

Bit(s)	Name	Description	Value
17-2:16	intr_vect	Interrupt vector. Shows the interrupt requests. intr_vect is only updated when rw_ctrl.en equals yes.	
1	intr_busy	Interrupt status. no: MPU is not running an interrupt routine yes: MPU is running an interrupt routine	no=0 yes=1
0	instr_reg_busy	Shows if rw_instr contains an instruction which hasn't been executed. no: The instruction register is not busy yes: The instruction register is in use	no=0 yes=1

25.22.5 rw_instr

Address	0x8c
Default	
Type	Read/Write
Description	The instruction register. A MPU instruction can be written to this register. The MPU will then be interrupted and the instruction from this register is executed. If the the MPU is running an interrupt routine, the instruction won't be executed until after the interrupt routine has finished. When this register contains an instruction which has not yet been executed, the status field r_stat.instr_reg_busy will be set. If the instruction contains a 32-bit immediate value, the instruction must first be written to rw_instr before writing the immediate value to rw_immediate.

25.22.6 rw_immediate

Address	0x90
Default	
Type	Read/Write
Description	See description in rw_instr.

25.22.7 r_trace

Address	0x94
Default	
Type	Read
Description	Trace register.

Bit(s)	Name	Description	Value
30	intr_busy	Interrupt status (same as r_stat.intr_busy). no: MPU is not running an interrupt routine yes: MPU is running an interrupt routine	no=0 yes=1
29	instr_reg_busy	Status of rw_instr (same as r_stat.instr_reg_busy). no: The instruction register is not busy yes: The instruction register is in use	no=0 yes=1
28	en	MPU is enabled or disabled. no: The MPU is disabled yes: The MPU is enabled	no=0 yes=1
27-16:12	pc	The program counter (same as r_pc.addr).	
15-0:16	intr_vect	Interrupt vector. Shows the interrupt requests. intr_vect is only updated when rw_ctrl.en equals yes.	

25.22.8 r_wr_stat

Address	0x98
Default	
Type	Read
Description	When data is written to one of the general registers, the corresponding status bit in this register is set. The MPU can acknowledge the change of a register by clearing the status bit by executing an arithmetic instruction on the special register WSTS.

Bit(s)	Name	Description	Value
15	r15	Write status for R15. no: No value has been written to R15 yes: A value has been written to R15	no=0 yes=1
14	r14	Write status for R14. no: No value has been written to R14 yes: A value has been written to R14	no=0 yes=1
13	r13	Write status for R13. no: No value has been written to R13 yes: A value has been written to R13	no=0 yes=1
12	r12	Write status for R12. no: No value has been written to R12 yes: A value has been written to R12	no=0 yes=1
11	r11	Write status for R11. no: No value has been written to R11 yes: A value has been written to R11	no=0 yes=1
10	r10	Write status for R10. no: No value has been written to R10 yes: A value has been written to R10	no=0 yes=1
9	r9	Write status for R9. no: No value has been written to R9 yes: A value has been written to R9	no=0 yes=1
8	r8	Write status for R8. no: No value has been written to R8 yes: A value has been written to R8	no=0 yes=1
7	r7	Write status for R7. no: No value has been written to R7 yes: A value has been written to R7	no=0 yes=1
6	r6	Write status for R6. no: No value has been written to R6 yes: A value has been written to R6	no=0 yes=1
5	r5	Write status for R5. no: No value has been written to R5 yes: A value has been written to R5	no=0 yes=1

4	r4	Write status for R4. no: No value has been written to R4 yes: A value has been written to R4	no=0 yes=1
3	r3	Write status for R3. no: No value has been written to R3 yes: A value has been written to R3	no=0 yes=1
2	r2	Write status for R2. no: No value has been written to R2 yes: A value has been written to R2	no=0 yes=1
1	r1	Write status for R1. no: No value has been written to R1 yes: A value has been written to R1	no=0 yes=1
0	r0	Write status for R0. no: No value has been written to R0 yes: A value has been written to R0	no=0 yes=1

25.22.9 rw_thread

Address	0x9c, 0xa0, 0xa4, 0xa8
Default	0x00000000, 0x00000000, 0x00000000, 0x00000000
Type	Read/Write
Description	Thread address.

Bit(s)	Name	Description	Value
11-0:12	addr	Contains a thread address which points to an address in the MPU memory. Threads are explained in 13.4.1.8.	

25.22.10 rw_intr

Address	0xc4, 0xc8, 0xcc, 0xd0, 0xd4, 0xd8, 0xdc, 0xe0, 0xe4, 0xe8, 0xec, 0xf0, 0xf4, 0xf8, 0xfc, 0x100
Default	
Type	Read/Write
Description	Interrupt vector.

Bit(s)	Name	Description	Value
11-0:12	addr	Interrupt address in the MPU memory.	

25.23 iop_sap_in

Instance	Base Address
iop_sap_in	0xb0020d00

25.23.1 rw_bus0_sync

Address	0x0
Default	0x02020202
Type	Read/Write
Description	This register controls the synchronization of input BUS0. There are separate fields for each byte of the bus.

Bit(s)	Name	Description	Value
31	byte3_delay	Delay BUS0[31:24] with one 200 MHz flip-flop, after synchronization. no: No extra 200 MHz flip-flop yes: Add one extra 200 MHz flip-flop	no=0 yes=1
30-29:2	byte3_edge	Selects edge of external source for synchronization, only used if byte3_sel equals ext_clk200 or no_del_ext_clk200. pos: Data is synchronized on positive edge neg: Data is synchronized on negative edge pos_neg: Data is synchronized on both positive and negative edge	pos=1 neg=2 pos_neg=3
28-26:3	byte3_ext_src	Select source for external synchronization (byte3_sel equals ext_clk200 or no_del_ext_clk200) or Timer source (byte3_sel equals tmr_clk200). gio1: Use `in_gio[1]` gio6: Use `in_gio[6]` gio7: Use `in_gio[7]` gio18: Use `in_gio[18]` gio19: Use `in_gio[19]` gio23: Use `in_gio[23]` timer_grp0_tmr3: Use Timer group 0, Timer 3 timer_grp3_tmr3: Use Timer group 3, Timer 3	gio1=0 gio6=1 gio7=2 gio18=3 gio19=4 gio23=5 timer_grp0_tmr3=6 timer_grp3_tmr3=7

25-24:2	byte3_sel	Select synchronization path for byte3, BUS0[31:24]. tmr_clk200: Use timer + 200 MHz flip-flops for synchronization two_clk200: Use two 200 MHz flip-flops for synchronization no_del_ext_clk200: Use external signal (selected by byte3_ext_src) and two 200 MHz flip-flops ext_clk200: Use external signal (selected by byte3_ext_src) and two 200 MHz flip-flops, data can handle zero hold time	tmr_clk200=0 two_clk200=2 no_del_ext_clk200=1 ext_clk200=3
23	byte2_delay	Delay BUS0[23:16] with one 200 MHz flip-flop, after synchronization. no: No extra 200 MHz flip-flop yes: Add one extra 200 MHz flip-flop	no=0 yes=1
22-21:2	byte2_edge	Selects edge of external source for synchronization, only used if byte2_sel equals ext_clk200 or no_del_ext_clk200. pos: Data is synchronized on positive edge neg: Data is synchronized on negative edge pos_neg: Data is synchronized on both positive and negative edge	pos=1 neg=2 pos_neg=3
20-18:3	byte2_ext_src	Select source for external synchronization (byte2_sel equals ext_clk200 or no_del_ext_clk200) or Timer source (byte2_sel equals tmr_clk200). gio1: Use `in_gio[1]` gio6: Use `in_gio[6]` gio7: Use `in_gio[7]` gio18: Use `in_gio[18]` gio19: Use `in_gio[19]` gio23: Use `in_gio[23]` timer_grp0_tmr3: Use Timer group 0, Timer 3 timer_grp3_tmr3: Use Timer group 3, Timer 3	gio1=0 gio6=1 gio7=2 gio18=3 gio19=4 gio23=5 timer_grp0_tmr3=6 timer_grp3_tmr3=7

17-16:2	byte2_sel	Select synchronization path for byte2, BUS0[23:16]. tmr_clk200: Use timer + 200 MHz flip-flops for synchronization two_clk200: Use two 200 MHz flip-flops for synchronization no_del_ext_clk200: Use external signal (selected by byte2_ext_src) and two 200 MHz flip-flops ext_clk200: Use external signal (selected by byte2_ext_src) and two 200 MHz flip-flops, data can handle zero hold time	tmr_clk200=0 two_clk200=2 no_del_ext_clk200=1 ext_clk200=3
15	byte1_delay	Delay BUS0[15:8] with one 200 MHz flip-flop, after synchronization. no: No extra 200 MHz flip-flop yes: Add one extra 200 MHz flip-flop	no=0 yes=1
14-13:2	byte1_edge	Selects edge of external source for synchronization, only used if byte1_sel equals ext_clk200 or no_del_ext_clk200. pos: Data is synchronized on positive edge neg: Data is synchronized on negative edge pos_neg: Data is synchronized on both positive and negative edge	pos=1 neg=2 pos_neg=3
12-10:3	byte1_ext_src	Select source for external synchronization (byte1_sel equals ext_clk200 or no_del_ext_clk200) or Timer source (byte1_sel equals tmr_clk200). gio1: Use `in_gio[1]` gio6: Use `in_gio[6]` gio7: Use `in_gio[7]` gio18: Use `in_gio[18]` gio19: Use `in_gio[19]` gio23: Use `in_gio[23]` timer_grp0_tmr3: Use Timer group 0, Timer 3 timer_grp3_tmr3: Use Timer group 3, Timer 3	gio1=0 gio6=1 gio7=2 gio18=3 gio19=4 gio23=5 timer_grp0_tmr3=6 timer_grp3_tmr3=7

9-8:2	byte1_sel	Select synchronization path for byte1, BUS0[15:8]. tmr_clk200: Use timer + 200 MHz flip-flops for synchronization two_clk200: Use two 200 MHz flip-flops for synchronization no_del_ext_clk200: Use external signal (selected by byte1_ext_src) and two 200 MHz flip-flops ext_clk200: Use external signal (selected by byte1_ext_src) and two 200 MHz flip-flops, data can handle zero hold time	tmr_clk200=0 two_clk200=2 no_del_ext_clk200=1 ext_clk200=3
7	byte0_delay	Delay BUS0[7:0] with one 200 MHz flip-flop, after synchronization. no: No extra 200 MHz flip-flop yes: Add one extra 200 MHz flip-flop	no=0 yes=1
6-5:2	byte0_edge	Selects edge of external source for synchronization, only used if byte0_sel equals ext_clk200 or no_del_ext_clk200. pos: Data is synchronized on positive edge neg: Data is synchronized on negative edge pos_neg: Data is synchronized on both positive and negative edge	pos=1 neg=2 pos_neg=3
4-2:3	byte0_ext_src	Select source for external synchronization (byte0_sel equals ext_clk200 or no_del_ext_clk200) or Timer source (byte0_sel equals tmr_clk200). gio1: Use `in_gio[1]` gio6: Use `in_gio[6]` gio7: Use `in_gio[7]` gio18: Use `in_gio[18]` gio19: Use `in_gio[19]` gio23: Use `in_gio[23]` timer_grp0_tmr3: Use Timer group 0, Timer 3 timer_grp3_tmr3: Use Timer group 3, Timer 3	gio1=0 gio6=1 gio7=2 gio18=3 gio19=4 gio23=5 timer_grp0_tmr3=6 timer_grp3_tmr3=7

1-0:2	byte0_sel	Select synchronization path for byte0, BUS0[7:0]. tmr_clk200: Use timer + 200 MHz flip-flops for synchronization two_clk200: Use two 200 MHz flip-flops for synchronization no_del_ext_clk200: Use external signal (selected by byte0_ext_src) and two 200 MHz flip-flops ext_clk200: Use external signal (selected by byte0_ext_src) and two 200 MHz flip-flops, data can handle zero hold time	tmr_clk200=0 two_clk200=2 no_del_ext_clk200=1 ext_clk200=3

25.23.2 rw_bus1_sync

Address	0x4
Default	0x02020202
Type	Read/Write
Description	This register controls the synchronization of input BUS1. There are separate fields for each byte of the bus.

Bit(s)	Name	Description	Value
31	byte3_delay	Delay BUS1[31:24] with one 200 MHz flip-flop, after synchronization. no: No extra 200 MHz flip-flop yes: Add one extra 200 MHz flip-flop	no=0 yes=1
30-29:2	byte3_edge	Selects edge of external source for synchronization, only used if byte3_sel equals ext_clk200 or no_del_ext_clk200. pos: Data is synchronized on positive edge neg: Data is synchronized on negative edge pos_neg: Data is synchronized on both positive and negative edge	pos=1 neg=2 pos_neg=3
28-26:3	byte3_ext_src	Select source for external synchronization (byte3_sel equals ext_clk200 or no_del_ext_clk200) or Timer source (byte3_sel equals tmr_clk200). gio1: Use `in_gio[1]` gio6: Use `in_gio[6]` gio7: Use `in_gio[7]` gio18: Use `in_gio[18]` gio19: Use `in_gio[19]` gio23: Use `in_gio[23]` timer_grp0_tmr3: Use Timer group 0, Timer 3 timer_grp3_tmr3: Use Timer group 3, Timer 3	gio1=0 gio6=1 gio7=2 gio18=3 gio19=4 gio23=5 timer_grp0_tmr3=6 timer_grp3_tmr3=7

25-24:2	byte3_sel	Select synchronization path for byte3, BUS1[31:24]. tmr_clk200: Use timer + 200 MHz flip-flops for synchronization two_clk200: Use two 200 MHz flip-flops for synchronization no_del_ext_clk200: Use external signal (selected by byte3_ext_src) and two 200 MHz flip-flops ext_clk200: Use external signal (selected by byte3_ext_src) and two 200 MHz flip-flops, data can handle zero hold time	tmr_clk200=0 two_clk200=2 no_del_ext_clk200=1 ext_clk200=3
23	byte2_delay	Delay BUS1[23:16] with one 200 MHz flip-flop, after synchronization. no: No extra 200 MHz flip-flop yes: Add one extra 200 MHz flip-flop	no=0 yes=1
22-21:2	byte2_edge	Selects edge of external source for synchronization, only used if byte2_sel equals ext_clk200 or no_del_ext_clk200. pos: Data is synchronized on positive edge neg: Data is synchronized on negative edge pos_neg: Data is synchronized on both positive and negative edge	pos=1 neg=2 pos_neg=3
20-18:3	byte2_ext_src	Select source for external synchronization (byte2_sel equals ext_clk200 or no_del_ext_clk200) or Timer source (byte2_sel equals tmr_clk200). gio1: Use `in_gio[1]` gio6: Use `in_gio[6]` gio7: Use `in_gio[7]` gio18: Use `in_gio[18]` gio19: Use `in_gio[19]` gio23: Use `in_gio[23]` timer_grp0_tmr3: Use Timer group 0, Timer 3 timer_grp3_tmr3: Use Timer group 3, Timer 3	gio1=0 gio6=1 gio7=2 gio18=3 gio19=4 gio23=5 timer_grp0_tmr3=6 timer_grp3_tmr3=7

17-16:2	byte2_sel	Select synchronization path for byte2, BUS1[23:16]. tmr_clk200: Use timer + 200 MHz flip-flops for synchronization two_clk200: Use two 200 MHz flip-flops for synchronization no_del_ext_clk200: Use external signal (selected by byte2_ext_src) and two 200 MHz flip-flops ext_clk200: Use external signal (selected by byte2_ext_src) and two 200 MHz flip-flops, data can handle zero hold time	tmr_clk200=0 two_clk200=2 no_del_ext_clk200=1 ext_clk200=3
15	byte1_delay	Delay BUS1[15:8] with one 200 MHz flip-flop, after synchronization. no: No extra 200 MHz flip-flop yes: Add one extra 200 MHz flip-flop	no=0 yes=1
14-13:2	byte1_edge	Selects edge of external source for synchronization, only used if byte1_sel equals ext_clk200 or no_del_ext_clk200. pos: Data is synchronized on positive edge neg: Data is synchronized on negative edge pos_neg: Data is synchronized on both positive and negative edge	pos=1 neg=2 pos_neg=3
12-10:3	byte1_ext_src	Select source for external synchronization (byte1_sel equals ext_clk200 or no_del_ext_clk200) or Timer source (byte1_sel equals tmr_clk200). gio1: Use `in_gio[1]` gio6: Use `in_gio[6]` gio7: Use `in_gio[7]` gio18: Use `in_gio[18]` gio19: Use `in_gio[19]` gio23: Use `in_gio[23]` timer_grp0_tmr3: Use Timer group 0, Timer 3 timer_grp3_tmr3: Use Timer group 3, Timer 3	gio1=0 gio6=1 gio7=2 gio18=3 gio19=4 gio23=5 timer_grp0_tmr3=6 timer_grp3_tmr3=7

9-8:2	byte1_sel	Select synchronization path for byte1, BUS1[15:8]. tmr_clk200: Use timer + 200 MHz flip-flops for synchronization two_clk200: Use two 200 MHz flip-flops for synchronization no_del_ext_clk200: Use external signal (selected by byte1_ext_src) and two 200 MHz flip-flops ext_clk200: Use external signal (selected by byte1_ext_src) and two 200 MHz flip-flops, data can handle zero hold time	tmr_clk200=0 two_clk200=2 no_del_ext_clk200=1 ext_clk200=3
7	byte0_delay	Delay BUS1[7:0] with one 200 MHz flip-flop, after synchronization. no: No extra 200 MHz flip-flop yes: Add one extra 200 MHz flip-flop	no=0 yes=1
6-5:2	byte0_edge	Selects edge of external source for synchronization, only used if byte0_sel equals ext_clk200 or no_del_ext_clk200. pos: Data is synchronized on positive edge neg: Data is synchronized on negative edge pos_neg: Data is synchronized on both positive and negative edge	pos=1 neg=2 pos_neg=3
4-2:3	byte0_ext_src	Select source for external synchronization (byte0_sel equals ext_clk200 or no_del_ext_clk200) or Timer source (byte0_sel equals tmr_clk200). gio1: Use `in_gio[1]` gio6: Use `in_gio[6]` gio7: Use `in_gio[7]` gio18: Use `in_gio[18]` gio19: Use `in_gio[19]` gio23: Use `in_gio[23]` timer_grp0_tmr3: Use Timer group 0, Timer 3 timer_grp3_tmr3: Use Timer group 3, Timer 3	gio1=0 gio6=1 gio7=2 gio18=3 gio19=4 gio23=5 timer_grp0_tmr3=6 timer_grp3_tmr3=7

1-0:2	byte0_sel	Select synchronization path for byte0, BUS1[7:0]. tmr_clk200: Use timer + 200 MHz flip-flops for synchronization two_clk200: Use two 200 MHz flip-flops for synchronization no_del_ext_clk200: Use external signal (selected by byte0_ext_src) and two 200 MHz flip-flops ext_clk200: Use external signal (selected by byte0_ext_src) and two 200 MHz flip-flops, data can handle zero hold time	tmr_clk200=0 two_clk200=2 no_del_ext_clk200=1 ext_clk200=3

25.23.3 rw_gio

Address	0x8, 0xc, 0x10, 0x14, 0x18, 0x1c, 0x20, 0x24, 0x28, 0x2c, 0x30, 0x34, 0x38, 0x3c, 0x40, 0x44, 0x48, 0x4c, 0x50, 0x54, 0x58, 0x5c, 0x60, 0x64, 0x68, 0x6c, 0x70, 0x74, 0x78, 0x7c, 0x80, 0x84
Default	0x00000000, 0x00000000
Type	Read/Write
Description	This vector register controls the synchronization and NOT/AND/OR logic of each input GIO.

Bit(s)	Name	Description	Value
9-8:2	logic	NOT/AND/OR logic for GIO. Each synchronized GIO can be and:ed or or:ed with another synchronized GIO input. e.g., GIO[0] can be and:ed with GIO[1]. Note: GIO[31] uses GIO[0] as operand. none: No extra logic inv: The GIO[x] signal is inverted and: The GIO[x] signal is and:ed with GIO[x+1] or: The GIO[x] signal is or:ed with GIO[x+1]	none=0 inv=1 and=2 or=3
7	delay	Delay GIO with one 200 MHz flip-flop, after synchronization. no: No extra 200 MHz flip-flop yes: Add one extra 200 MHz flip-flop	no=0 yes=1
6-5:2	sync_edge	Selects edge of external source for synchronization, only used if sync_sel equals ext_clk200 or no_del_ext_clk200. pos: Data is synchronized on positive edge neg: Data is synchronized on negative edge pos_neg: Data is synchronized on both positive and negative edge	pos=1 neg=2 pos_neg=3

4-2:3	sync_ext_src	Select source for external synchronization (sync_sel equals ext_clk200 or no_del_ext_clk200) or Timer source (sync_sel equals tmr_clk200). timer_grp1_tmr3: Use Timer group 1, Timer 3 timer_grp2_tmr3: Use Timer group 2, Timer 3 timer_grp0_tmr3: Use Timer group 0, Timer 3 timer_grp3_tmr3: Use Timer group 3, Timer 3 gio1: Use `in_gio[1]` gio6: Use `in_gio[6]` gio7: Use `in_gio[7]` gio18: Use `in_gio[18]` gio5: Use `in_gio[5]` gio13: Use `in_gio[13]` gio21: Use `in_gio[21]` gio29: Use `in_gio[29]`	timer_grp1_tmr3=4 timer_grp2_tmr3=5 timer_grp0_tmr3=6 timer_grp3_tmr3=7 gio1=0 gio6=1 gio7=2 gio18=3 gio5=4 gio13=5 gio21=6 gio29=7
1-0:2	sync_sel	Select synchronization path for GIO. tmr_clk200: Use Timer, selected by sync_ext_src, followed by two 200 MHz flip-flops for synchronization two_clk200: Use two 200 MHz flip-flops for synchronization no_del_ext_clk200: Use external signal (selected by sync_ext_src) and two 200 MHz flip-flops ext_clk200: Use external signal (selected by sync_ext_src) and two 200 MHz flip-flops, data can handle zero hold time	tmr_clk200=0 two_clk200=2 no_del_ext_clk200=1 ext_clk200=3

25.24 iop_sap_out

Instance	Base Address
iop_sap_out	0xb0020e00

25.24.1 rw_gen_gated

Address	0x0
Default	0x00000000
Type	Read/Write
Description	This register is used to configure the SAP_OUT gated clocks. There are four gated clocks which can be used to clock out signals.

Bit(s)	Name	Description	Value
27-25:3	clk3_force_src	The clk3_force_src selects a signal to OR with the signal selected by clk3_gate_src, before latch. none: Use no signal spu0_gio6: Use spu0 `out_gio[6]` spu0_gio7: Use spu0 `out_gio[7]` spu0_gio15: Use spu0 `out_gio[15]` spu1_gio6: Use spu1 `out_gio[6]` spu1_gio7: Use spu1 `out_gio[7]` spu1_gio15: Use spu1 `out_gio[15]`	none=0 spu0_gio6=2 spu0_gio7=3 spu0_gio15=4 spu1_gio6=5 spu1_gio7=6 spu1_gio15=7
24-23:2	clk3_gate_src	Gate source for gated clock 3. gio7: Use `in_gio[7]` gio15: Use `in_gio[15]` gio23: Use `in_gio[23]` gio31: Use `in_gio[31]`	gio7=0 gio15=1 gio23=2 gio31=3
22-21:2	clk3_src	Clock source for gated clock 3. gio1: Use `in_gio[1]` gio5: Use `in_gio[5]` gio13: Use `in_gio[13]` gio18: Use `in_gio[18]`	gio1=0 gio5=1 gio13=2 gio18=3
20-18:3	clk2_force_src	The clk2_force_src selects a signal to OR with the signal selected by clk2_gate_src, before latch. none: Use no signal spu0_gio4: Use spu0 `out_gio[4]` spu0_gio5: Use spu0 `out_gio[5]` spu0_gio14: Use spu0 `out_gio[14]` spu1_gio4: Use spu1 `out_gio[4]` spu1_gio5: Use spu1 `out_gio[5]` spu1_gio14: Use spu1 `out_gio[14]`	none=0 spu0_gio4=2 spu0_gio5=3 spu0_gio14=4 spu1_gio4=5 spu1_gio5=6 spu1_gio14=7

17-16:2	clk2_gate_src	Gate source for gated clock 2. gio7: Use in_gio[7] gio15: Use in_gio[15] gio23: Use in_gio[23] gio31: Use in_gio[31]	gio7=0 gio15=1 gio23=2 gio31=3
15-14:2	clk2_src	Clock source for gated clock 2. gio1: Use in_gio[1] gio5: Use in_gio[5] gio13: Use in_gio[13] gio18: Use in_gio[18]	gio1=0 gio5=1 gio13=2 gio18=3
13-11:3	clk1_force_src	The clk1_force_src selects a signal to OR with the signal selected by clk1_gate_src, before latch. none: Use no signal spu0_gio2: Use spu0 out_gio[2] spu0_gio3: Use spu0 out_gio[3] spu0_gio13: Use spu0 out_gio[13] spu1_gio2: Use spu1 out_gio[2] spu1_gio3: Use spu1 out_gio[3] spu1_gio13: Use spu1 out_gio[13]	none=0 spu0_gio2=2 spu0_gio3=3 spu0_gio13=4 spu1_gio2=5 spu1_gio3=6 spu1_gio13=7
10-9:2	clk1_gate_src	Gate source for gated clock 1. gio7: Use in_gio[7] gio15: Use in_gio[15] gio23: Use in_gio[23] gio31: Use in_gio[31]	gio7=0 gio15=1 gio23=2 gio31=3
8-7:2	clk1_src	Clock source for gated clock 1. gio1: Use in_gio[1] gio5: Use in_gio[5] gio13: Use in_gio[13] gio18: Use in_gio[18]	gio1=0 gio5=1 gio13=2 gio18=3
6-4:3	clk0_force_src	The clk0_force_src selects a signal to OR with the signal selected by clk0_gate_src, before latch. none: Use no signal spu0_gio0: Use spu0 out_gio[0] spu0_gio1: Use spu0 out_gio[1] spu0_gio12: Use spu0 out_gio[12] spu1_gio0: Use spu1 out_gio[0] spu1_gio1: Use spu1 out_gio[1] spu1_gio12: Use spu1 out_gio[12]	none=0 spu0_gio0=2 spu0_gio1=3 spu0_gio12=4 spu1_gio0=5 spu1_gio1=6 spu1_gio12=7
3-2:2	clk0_gate_src	Gate source for gated clock 0. gio7: Use in_gio[7] gio15: Use in_gio[15] gio23: Use in_gio[23] gio31: Use in_gio[31]	gio7=0 gio15=1 gio23=2 gio31=3

1-0:2	clk0_src	Clock source for gated clock 0. gio1: Use `in_gio[1]` gio5: Use `in_gio[5]` gio13: Use `in_gio[13]` gio18: Use `in_gio[18]`	gio1=0 gio5=1 gio13=2 gio18=3

25.24.2 rw_bus0

Address	0x4
Default	0x00000000
Type	Read/Write
Description	This register controls the out-clocking of BUS0. Each byte of the bus has separate fields to select out-clocking behavior.

Bit(s)	Name	Description	Value
23	byte3_clk_inv	Invert gated clock before out clocking. This field is only used if byte3_clk_sel equals gated. no: Do not invert gated clock (posedge) yes: Inverted gated clock (negedge)	no=0 yes=1
22-21:2	byte3_gated_clk	Select which gated clock to use. This field is only used if byte3_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
20-18:3	byte3_clk_sel	Out-clocking of byte3, bus0[31:24]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop gated: Use one flip-flop, clocked by gated clock	none=0 clk200=1 gated=4
17	byte2_clk_inv	Invert gated clock before out clocking. This field is only used if byte2_clk_sel equals gated. no: Do not invert gated clock (posedge) yes: Inverted gated clock (negedge)	no=0 yes=1
16-15:2	byte2_gated_clk	Select which gated clock to use. This field is only used if byte2_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
14-12:3	byte2_clk_sel	Out-clocking of byte2, bus0[23:16]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop gated: Use one flip-flop, clocked by gated clock	none=0 clk200=1 gated=4
11	byte1_clk_inv	Invert gated clock before out clocking. This field is only used if byte1_clk_sel equals gated. no: Do not invert gated clock (posedge) yes: Inverted gated clock (negedge)	no=0 yes=1

10-9:2	byte1_gated_clk	Select which gated clock to use. This field is only used if byte1_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
8-6:3	byte1_clk_sel	Out-clocking of byte1, bus0[15:8]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop gated: Use one flip-flop, clocked by gated clock	none=0 clk200=1 gated=4
5	byte0_clk_inv	Invert gated clock before out clocking. This field is only used if byte0_clk_sel equals gated. no: Do not invert gated clock (posedge) yes: Inverted gated clock (negedge)	no=0 yes=1
4-3:2	byte0_gated_clk	Select which gated clock to use. This field is only used if byte0_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
2-0:3	byte0_clk_sel	Out-clocking of byte0, bus0[7:0]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop gated: Use one flip-flop, clocked by gated clock	none=0 clk200=1 gated=4

25.24.3 rw_bus1

Address	0x8
Default	0x00000000
Type	Read/Write
Description	This register controls the out-clocking of BUS1. Each byte of the bus has separate fields to select out-clocking behavior.

Bit(s)	Name	Description	Value
23	byte3_clk_inv	Invert gated clock before out clocking. This field is only used if byte3_clk_sel equals gated. no: Do not invert gated clock (posedge) yes: Inverted gated clock (negedge)	no=0 yes=1
22-21:2	byte3_gated_clk	Select which gated clock to use. This field is only used if byte3_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
20-18:3	byte3_clk_sel	Out-clocking of byte3, bus1[31:24]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop gated: Use one flip-flop, clocked by gated clock	none=0 clk200=1 gated=4
17	byte2_clk_inv	Invert gated clock before out clocking. This field is only used if byte2_clk_sel equals gated. no: Do not invert gated clock (posedge) yes: Inverted gated clock (negedge)	no=0 yes=1
16-15:2	byte2_gated_clk	Select which gated clock to use. This field is only used if byte2_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
14-12:3	byte2_clk_sel	Out-clocking of byte2, bus1[23:16]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop gated: Use one flip-flop, clocked by gated clock	none=0 clk200=1 gated=4
11	byte1_clk_inv	Invert gated clock before out clocking. This field is only used if byte1_clk_sel equals gated. no: Do not invert gated clock (posedge) yes: Inverted gated clock (negedge)	no=0 yes=1

10-9:2	byte1_gated_clk	Select which gated clock to use. This field is only used if byte1_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
8-6:3	byte1_clk_sel	Out-clocking of byte1, bus1[15:8]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop gated: Use one flip-flop, clocked by gated clock	none=0 clk200=1 gated=4
5	byte0_clk_inv	Invert gated clock before out clocking. This field is only used if byte0_clk_sel equals gated. no: Do not invert gated clock (posedge) yes: Inverted gated clock (negedge)	no=0 yes=1
4-3:2	byte0_gated_clk	Select which gated clock to use. This field is only used if byte0_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
2-0:3	byte0_clk_sel	Out-clocking of byte0, bus1[7:0]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop gated: Use one flip-flop, clocked by gated clock	none=0 clk200=1 gated=4

25.24.4 rw_bus0_lo_oe

Address	0xc
Default	0x00000000
Type	Read/Write
Description	This register controls the out-clocking of the output enable (OE) signal used for byte0 and byte1 of BUS0. OE for each byte can be clocked out by selected source.

Bit(s)	Name	Description	Value
21-20:2	byte1_logic	NOT/AND/NAND logic for byte1 OE signal. none: No extra logic inv: The OE signal is inverted and: The OE signal is and:ed with `out_gio[4]` nand: The OE signal is nand:ed with `out_gio[4]`	none=0 inv=1 and=2 nand=3
19	byte1_clk_inv	Invert gated, external clock or 12 MHz clock before out clocking. This field is only used if byte1_clk_sel equals gated, ext or clk12. no: Do not invert clock (posedge) yes: Inverted clock (negedge)	no=0 yes=1
18-17:2	byte1_gated_clk	Select which gated clock to use. This field is only used if byte1_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
16-14:3	byte1_clk_ext	External signal used for clocking out OE for byte1, BUS0[15:8]. This field is only used if byte1_clk_sel equals ext. in_gio5: Use `in_gio[5]` in_gio13: Use `in_gio[13]` in_gio21: Use `in_gio[21]` in_gio29: Use `in_gio[29]`	in_gio5=0 in_gio13=1 in_gio21=2 in_gio29=3
13-11:3	byte1_clk_sel	Out-clocking source of OE for byte1, BUS0[15:8]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop clk12: Use one 12 MHz flip-flop ext: Use one flip-flop, clocked by an external source (selected by byte1_clk_ext) gated: Use one flip-flop, clocked by gated clock (selected by byte1_gated_clk)	none=0 clk200=1 clk12=2 ext=3 gated=4
10-9:2	byte0_logic	NOT/AND/NAND logic for byte0 OE signal. none: No extra logic inv: The OE signal is inverted and: The OE signal is and:ed with `out_gio[4]` nand: The OE signal is nand:ed with `out_gio[4]`	none=0 inv=1 and=2 nand=3

8	byte0_clk_inv	Invert gated, external clock or 12 MHz clock before out clocking. This field is only used if byte0_clk_sel equals gated, ext or clk12. no: Do not invert clock (posedge) yes: Inverted clock (negedge)	no=0 yes=1
7-6:2	byte0_gated_clk	Select which gated clock to use. This field is only used if byte0_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
5-3:3	byte0_clk_ext	External signal used for clocking out OE for byte0, BUS0[7:0]. This field is only used if byte0_clk_sel equals ext. in_gio5: Use `in_gio[5]` in_gio13: Use `in_gio[13]` in_gio21: Use `in_gio[21]` in_gio29: Use `in_gio[29]`	in_gio5=0 in_gio13=1 in_gio21=2 in_gio29=3
2-0:3	byte0_clk_sel	Out-clocking source of OE for byte0, BUS0[7:0]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop clk12: Use one 12 MHz flip-flop ext: Use one flip-flop, clocked by an external source (selected by byte0_clk_ext) gated: Use one flip-flop, clocked by gated clock (selected by byte0_gated_clk)	none=0 clk200=1 clk12=2 ext=3 gated=4

25.24.5 rw_bus0_hi_oe

Address	0x10
Default	0x00000000
Type	Read/Write
Description	This register controls the out-clocking of the output enable (OE) signal used for byte2 and byte3 of BUS0. OE for each byte can be clocked out by selected source.

Bit(s)	Name	Description	Value
21-20:2	byte3_logic	NOT/AND/NAND logic for byte3 OE signal. none: No extra logic inv: The OE signal is inverted and: The OE signal is and:ed with `out_gio[4]` nand: The OE signal is nand:ed with `out_gio[4]`	none=0 inv=1 and=2 nand=3
19	byte3_clk_inv	Invert gated, external clock or 12 MHz clock before out clocking. This field is only used if byte3_clk_sel equals gated, ext or clk12. no: Do not invert clock (posedge) yes: Inverted clock (negedge)	no=0 yes=1
18-17:2	byte3_gated_clk	Select which gated clock to use. This field is only used if byte3_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
16-14:3	byte3_clk_ext	External signal used for clocking out OE for byte3, BUS0[31:24]. This field is only used if byte3_clk_sel equals ext. in_gio5: Use `in_gio[5]` in_gio13: Use `in_gio[13]` in_gio21: Use `in_gio[21]` in_gio29: Use `in_gio[29]`	in_gio5=0 in_gio13=1 in_gio21=2 in_gio29=3
13-11:3	byte3_clk_sel	Out-clocking source of OE for byte3, BUS0[31:24]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop clk12: Use one 12 MHz flip-flop ext: Use one flip-flop, clocked by an external source (selected by byte3_clk_ext) gated: Use one flip-flop, clocked by gated clock (selected by byte3_gated_clk)	none=0 clk200=1 clk12=2 ext=3 gated=4
10-9:2	byte2_logic	NOT/AND/NAND logic for byte2 OE signal. none: No extra logic inv: The OE signal is inverted and: The OE signal is and:ed with `out_gio[4]` nand: The OE signal is nand:ed with `out_gio[4]`	none=0 inv=1 and=2 nand=3

8	byte2_clk_inv	Invert gated, external clock or 12 MHz clock before out clocking. This field is only used if byte2_clk_sel equals gated, ext or clk12. no: Do not invert clock (posedge) yes: Inverted clock (negedge)	no=0 yes=1
7-6:2	byte2_gated_clk	Select which gated clock to use. This field is only used if byte2_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
5-3:3	byte2_clk_ext	External signal used for clocking out OE for byte2, BUS0[23:16]. This field is only used if byte2_clk_sel equals ext. in_gio5: Use `in_gio[5]` in_gio13: Use `in_gio[13]` in_gio21: Use `in_gio[21]` in_gio29: Use `in_gio[29]`	in_gio5=0 in_gio13=1 in_gio21=2 in_gio29=3
2-0:3	byte2_clk_sel	Out-clocking source of OE for byte2, BUS0[23:16]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop clk12: Use one 12 MHz flip-flop ext: Use one flip-flop, clocked by an external source (selected by byte2_clk_ext) gated: Use one flip-flop, clocked by gated clock (selected by byte2_gated_clk)	none=0 clk200=1 clk12=2 ext=3 gated=4

25.24.6 rw_bus1_lo_oe

Address	0x14
Default	0x00000000
Type	Read/Write
Description	This register controls the out-clocking of the output enable (OE) signal used for byte0 and byte1 of BUS1. OE for each byte can be clocked out by selected source.

Bit(s)	Name	Description	Value
21-20:2	byte1_logic	NOT/AND/NAND logic for byte1 OE signal. none: No extra logic inv: The OE signal is inverted and: The OE signal is and:ed with `out_gio[0]` nand: The OE signal is nand:ed with `out_gio[0]`	none=0 inv=1 and=2 nand=3
19	byte1_clk_inv	Invert gated, external clock or 12 MHz clock before out clocking. This field is only used if byte1_clk_sel equals gated, ext or clk12. no: Do not invert clock (posedge) yes: Inverted clock (negedge)	no=0 yes=1
18-17:2	byte1_gated_clk	Select which gated clock to use. This field is only used if byte1_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
16-14:3	byte1_clk_ext	External signal used for clocking out OE for byte1, BUS1[15:8]. This field is only used if byte1_clk_sel equals ext. in_gio5: Use `in_gio[5]` in_gio13: Use `in_gio[13]` in_gio21: Use `in_gio[21]` in_gio29: Use `in_gio[29]`	in_gio5=0 in_gio13=1 in_gio21=2 in_gio29=3
13-11:3	byte1_clk_sel	Out-clocking source of OE for byte1, BUS1[15:8]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop clk12: Use one 12 MHz flip-flop ext: Use one flip-flop, clocked by an external source (selected by byte1_clk_ext) gated: Use one flip-flop, clocked by gated clock (selected by byte1_gated_clk)	none=0 clk200=1 clk12=2 ext=3 gated=4
10-9:2	byte0_logic	NOT/AND/NAND logic for byte0 OE signal. none: No extra logic inv: The OE signal is inverted and: The OE signal is and:ed with `out_gio[0]` nand: The OE signal is nand:ed with `out_gio[0]`	none=0 inv=1 and=2 nand=3

8	byte0_clk_inv	Invert gated, external clock or 12 MHz clock before out clocking. This field is only used if byte0_clk_sel equals gated, ext or clk12. no: Do not invert clock (posedge) yes: Inverted clock (negedge)	no=0 yes=1
7-6:2	byte0_gated_clk	Select which gated clock to use. This field is only used if byte0_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
5-3:3	byte0_clk_ext	External signal used for clocking out OE for byte0, BUS1[7:0]. This field is only used if byte0_clk_sel equals ext. in_gio5: Use `in_gio[5]` in_gio13: Use `in_gio[13]` in_gio21: Use `in_gio[21]` in_gio29: Use `in_gio[29]`	in_gio5=0 in_gio13=1 in_gio21=2 in_gio29=3
2-0:3	byte0_clk_sel	Out-clocking source of OE for byte0, BUS1[7:0]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop clk12: Use one 12 MHz flip-flop ext: Use one flip-flop, clocked by an external source (selected by byte0_clk_ext) gated: Use one flip-flop, clocked by gated clock (selected by byte0_gated_clk)	none=0 clk200=1 clk12=2 ext=3 gated=4

25.24.7 rw_bus1_hi_oe

Address	0x18
Default	0x00000000
Type	Read/Write
Description	This register controls the out-clocking of the output enable (OE) signal used for byte2 and byte3 of BUS1. OE for each byte can be clocked out by selected source.

Bit(s)	Name	Description	Value
21-20:2	byte3_logic	NOT/AND/NAND logic for byte3 OE signal. none: No extra logic inv: The OE signal is inverted and: The OE signal is and:ed with `out_gio[0]` nand: The OE signal is nand:ed with `out_gio[0]`	none=0 inv=1 and=2 nand=3
19	byte3_clk_inv	Invert gated, external clock or 12 MHz clock before out clocking. This field is only used if byte3_clk_sel equals gated, ext or clk12. no: Do not invert clock (posedge) yes: Inverted clock (negedge)	no=0 yes=1
18-17:2	byte3_gated_clk	Select which gated clock to use. This field is only used if byte3_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
16-14:3	byte3_clk_ext	External signal used for clocking out OE for byte3, BUS1[31:24]. This field is only used if byte3_clk_sel equals ext. in_gio5: Use `in_gio[5]` in_gio13: Use `in_gio[13]` in_gio21: Use `in_gio[21]` in_gio29: Use `in_gio[29]`	in_gio5=0 in_gio13=1 in_gio21=2 in_gio29=3
13-11:3	byte3_clk_sel	Out-clocking source of OE for byte3, BUS1[31:24]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop clk12: Use one 12 MHz flip-flop ext: Use one flip-flop, clocked by an external source (selected by byte3_clk_ext) gated: Use one flip-flop, clocked by gated clock (selected by byte3_gated_clk)	none=0 clk200=1 clk12=2 ext=3 gated=4
10-9:2	byte2_logic	NOT/AND/NAND logic for byte2 OE signal. none: No extra logic inv: The OE signal is inverted and: The OE signal is and:ed with `out_gio[0]` nand: The OE signal is nand:ed with `out_gio[0]`	none=0 inv=1 and=2 nand=3

8	byte2_clk_inv	Invert gated, external clock or 12 MHz clock before out clocking. This field is only used if byte2_clk_sel equals gated, ext or clk12. no: Do not invert clock (posedge) yes: Inverted clock (negedge)	no=0 yes=1
7-6:2	byte2_gated_clk	Select which gated clock to use. This field is only used if byte2_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
5-3:3	byte2_clk_ext	External signal used for clocking out OE for byte2, BUS1[23:16]. This field is only used if byte2_clk_sel equals ext. in_gio5: Use `in_gio[5]` in_gio13: Use `in_gio[13]` in_gio21: Use `in_gio[21]` in_gio29: Use `in_gio[29]`	in_gio5=0 in_gio13=1 in_gio21=2 in_gio29=3
2-0:3	byte2_clk_sel	Out-clocking source of OE for byte2, BUS1[23:16]. none: No flip-flops at all clk200: Use one 200 MHz flip-flop clk12: Use one 12 MHz flip-flop ext: Use one flip-flop, clocked by an external source (selected by byte2_clk_ext) gated: Use one flip-flop, clocked by gated clock (selected by byte2_gated_clk)	none=0 clk200=1 clk12=2 ext=3 gated=4

25.24.8 rw_gio

Address	0x1c, 0x20, 0x24, 0x28, 0x2c, 0x30, 0x34, 0x38, 0x3c, 0x40, 0x44, 0x48, 0x4c, 0x50, 0x54, 0x58, 0x5c, 0x60, 0x64, 0x68, 0x6c, 0x70, 0x74, 0x78, 0x7c, 0x80, 0x84, 0x88, 0x8c, 0x90, 0x94, 0x98
Default	0x00000000, 0x00000000
Type	Read/Write
Description	This register controls the out-clocking of the GIO and output enable (OE). The gio and its oe signal can be clocked out by selected source.

Bit(s)	Name	Description	Value
21-20:2	oe_logic	NOT/AND/NAND logic for GIO OE signal. The value of a is either 0 or 4 depending on the index number of rw_gio. For index 0, 5-7, 16-23 and 28-31, `out_gio[4]` is used and for index 1-4, 8-15 and 24-27, `out_gio[0]`. none: No extra logic inv: The OE signal is inverted and: The OE signal is and:ed with `out_gio[a]` nand: The OE signal is nand:ed with `out_gio[a]`	none=0 inv=1 and=2 nand=3
19	oe_clk_inv	Invert gated, external clock or 12 MHz clock before OE out clocking. This field is only used if oe_clk_sel equals gated, ext or clk12. no: Do not invert clock (posedge) yes: Inverted clock (negedge)	no=0 yes=1
18-17:2	oe_gated_clk	Select which gated clock to use. This field is only used if oe_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
16-14:3	oe_clk_ext	External signal used for clocking out OE, this field is only used if oe_clk_sel equals ext. in_gio5: Use `in_gio[5]` in_gio13: Use `in_gio[13]` in_gio21: Use `in_gio[21]` in_gio29: Use `in_gio[29]`	in_gio5=0 in_gio13=1 in_gio21=2 in_gio29=3

13-11:3	oe_clk_sel	Out-clocking source of OE for GIO out. none: No flip-flops at all clk200: Use one 200 MHz flip-flop clk12: Use one 12 MHz flip-flop ext: Use one flip-flop, clocked by an external source (selected by oe_clk_ext) gated: Use one flip-flop, clocked by gated clock (selected by oe_gated_clk)	none=0 clk200=1 clk12=2 ext=3 gated=4
10	out_logic	Out logic for GIO out signal. none: No extra logic inv: The out_gio signal is inverted	none=0 inv=1
9	out_clk_inv	Invert gated, external clock or 12 MHz clock before out clocking. This field is only used if out_clk_sel equals gated, ext or clk12. no: Do not invert clock (posedge) yes: Inverted clock (negedge)	no=0 yes=1
8-7:2	out_gated_clk	Select which gated clock to use. This field is only used if out_clk_sel equals gated. clk0: Use gated clock 0 clk1: Use gated clock 1 clk2: Use gated clock 2 clk3: Use gated clock 3	clk0=0 clk1=1 clk2=2 clk3=3
6-3:4	out_clk_ext	Select source for clocking out GIO (out_clk_sel equals ext) or Timer source (out_clk_sel equals tmr). timer_grp0_tmr2: Use timer group 0 timer_grp1_tmr2: Use timer group 1 timer_grp2_tmr2: Use timer group 2 timer_grp3_tmr2: Use timer group 3 gio1_clk: Use `in_gio[1]` gio5_clk: Use `in_gio[5]` gio6_clk: Use `in_gio[6]` gio7_clk: Use `in_gio[7]` gio13_clk: Use `in_gio[13]` gio18_clk: Use `in_gio[18]` gio21_clk: Use `in_gio[21]` gio29_clk: Use `in_gio[29]`	timer_grp0_tmr2=4 timer_grp1_tmr2=5 timer_grp2_tmr2=6 timer_grp3_tmr2=7 gio1_clk=8 gio5_clk=9 gio6_clk=10 gio7_clk=11 gio13_clk=12 gio18_clk=13 gio21_clk=14 gio29_clk=15

2-0:3	out_clk_sel	Out-clocking source of GIO out. none: No flip-flops at all clk200: Use one 200 MHz flip-flop clk12: Use one 12 MHz flip-flop ext: Use one flip-flop, clocked by an external source (selected by out_clk_ext) gated: Use one flip-flop, clocked by gated clock (selected by out_gated_clk) tmr: Use one flip-flop, clocked by 200 MHz + timer signal (selected by out_clk_ext)	none=0 clk200=1 clk12=2 ext=3 gated=4 tmr=5

25.25 iop_spu

Instance	Base Address
iop_spu0	0xb0020f00
iop_spu1	0xb0021000

25.25.1 rw_r

Address	0x0, 0x4, 0x8, 0xc, 0x10, 0x14, 0x18, 0x1c, 0x20, 0x24, 0x28, 0x2c, 0x30, 0x34, 0x38, 0x3c
Default	
Type	Read/Write
Description	SPU general register. If data is written to the register the corresponding bit in rs_wr_stat is set.

25.25.2 rw_seq_pc

Address	0x40
Default	
Type	Read/Write
Description	The program counter (PC) for SPU in sequential mode, 32 bit instruction alignment.

Bit(s)	Name	Description	Value
11-0:12	addr	The sequential mode program counter.	

25.25.3 rw_fsm_pc

Address	0x44
Default	
Type	Read/Write
Description	The program counter (PC) for SPU in FSM mode, 64 bit instruction alignment.

Bit(s)	Name	Description	Value
11-0:12	addr	The FSM mode program counter.	

25.25.4 rw_ctrl

Address	0x48
Default	0x00000000
Type	Read/Write
Description	The SPU control register.

Bit(s)	Name	Description	Value
1	en	Used to enable or disable the SPU. no: Disable the SPU yes: Enable the SPU	no=0 yes=1
0	fsm	The fsm field controls which mode (sequential or FSM) the SPU will use. no: Disable SPU FSM mode yes: Enable SPU FSM mode	no=0 yes=1

25.25.5 rw_fsm_inputs3_0

Address	0x4c
Default	
Type	Read/Write
Description	rw_fsm_inputs3_0 and rw_fsm_inputs7_4 are used to select eight inputs. These eight inputs are delayed one clock cycle and will thus be 5 ns delayed. Each FSM state then selects four of these eight inputs.

Bit(s)	Name	Description	Value
31-29:3	src3	Select source for `fsm_input[3]`. val3 is used to select which bit to use. gio_in: Input is selected from `gio_in[31:0]` flag: Input is selected from the four ALU flags reg_hi: Input is selected from bit[0] in general reg r15..r8 xor: Input is selected from the four xor signals statin_hi: Input is selected from `stat_in[31:16]` (hi) gio_out: Input is selected from `spu_gio_out[7:0]` attn_hi: Input is selected from `attn[15:8]` (hi) trigger: Input is selected from 32 Trigger strobes	gio_in=0 flag=2 reg_hi=2 xor=3 statin_hi=4 gio_out=5 attn_hi=5 trigger=6

28-24:5	val3	Select bit from source (src3) to use as `fsm_input[3]`. xor_bus0_r2_0: xor on bus0 and R2 bit 0 xor_bus1_r3_0: xor on bus1 and R3 bit 0 xor_bus0m_r2_0: xor on bus0 (masked with R0) and R2 bit 0 xor_bus1m_r3_0: xor on bus1 (masked with R1) and R3 bit 0 c: ALU c-flag v: ALU v-flag z: ALU z-flag n: ALU n-flag r8: bit[0] of r8 r9: bit[0] of r9 r10: bit[0] of r10 r11: bit[0] of r11 r12: bit[0] of r12 r13: bit[0] of r13 r14: bit[0] of r14 r15: bit[0] of r15 gio_out0: SPU `gio_out[0]` gio_out1: SPU `gio_out[1]` gio_out2: SPU `gio_out[2]` gio_out3: SPU `gio_out[3]` gio_out4: SPU `gio_out[4]` gio_out5: SPU `gio_out[5]` gio_out6: SPU `gio_out[6]` gio_out7: SPU `gio_out[7]` attn_r8: Attention of register R8 attn_r9: Attention of register R9 attn_r10: Attention of register R10 attn_r11: Attention of register R11 attn_r12: Attention of register R12 attn_r13: Attention of register R13 attn_r14: Attention of register R14 attn_r15: Attention of register R15	xor_bus0_r2_0=0 xor_bus1_r3_0=1 xor_bus0m_r2_0=2 xor_bus1m_r3_0=3 c=0 v=1 z=2 n=3 r8=8 r9=9 r10=10 r11=11 r12=12 r13=13 r14=14 r15=15 gio_out0=8 gio_out1=9 gio_out2=10 gio_out3=11 gio_out4=12 gio_out5=13 gio_out6=14 gio_out7=15 attn_r8=0 attn_r9=1 attn_r10=2 attn_r11=3 attn_r12=4 attn_r13=5 attn_r14=6 attn_r15=7
23-21:3	src2	Select source for `fsm_input[2]`. val2 is used to select which bit to use. gio_in: Input is selected from `gio_in[31:0]` flag: Input is selected from the four ALU flags reg_hi: Input is selected from bit[0] in general reg r15..r8 xor: Input is selected from the four xor signals statin_hi: Input is selected from `stat_in[31:16]` (hi) gio_out: Input is selected from `spu_gio_out[7:0]` attn_hi: Input is selected from `attn[15:8]` (hi) trigger: Input is selected from 32 Trigger strobes	gio_in=0 flag=2 reg_hi=2 xor=3 statin_hi=4 gio_out=5 attn_hi=5 trigger=6

20-16:5	val2	Select bit from source (src2) to use as `fsm_input[2]`. xor_bus0_r2_0: xor on bus0 and R2 bit 0 xor_bus1_r3_0: xor on bus1 and R3 bit 0 xor_bus0m_r2_0: xor on bus0 (masked with R0) and R2 bit 0 xor_bus1m_r3_0: xor on bus1 (masked with R1) and R3 bit 0 c: ALU c-flag v: ALU v-flag z: ALU z-flag n: ALU n-flag r8: bit[0] of r8 r9: bit[0] of r9 r10: bit[0] of r10 r11: bit[0] of r11 r12: bit[0] of r12 r13: bit[0] of r13 r14: bit[0] of r14 r15: bit[0] of r15 gio_out0: SPU `gio_out[0]` gio_out1: SPU `gio_out[1]` gio_out2: SPU `gio_out[2]` gio_out3: SPU `gio_out[3]` gio_out4: SPU `gio_out[4]` gio_out5: SPU `gio_out[5]` gio_out6: SPU `gio_out[6]` gio_out7: SPU `gio_out[7]` attn_r8: Attention of register R8 attn_r9: Attention of register R9 attn_r10: Attention of register R10 attn_r11: Attention of register R11 attn_r12: Attention of register R12 attn_r13: Attention of register R13 attn_r14: Attention of register R14 attn_r15: Attention of register R15	xor_bus0_r2_0=0 xor_bus1_r3_0=1 xor_bus0m_r2_0=2 xor_bus1m_r3_0=3 c=0 v=1 z=2 n=3 r8=8 r9=9 r10=10 r11=11 r12=12 r13=13 r14=14 r15=15 gio_out0=8 gio_out1=9 gio_out2=10 gio_out3=11 gio_out4=12 gio_out5=13 gio_out6=14 gio_out7=15 attn_r8=0 attn_r9=1 attn_r10=2 attn_r11=3 attn_r12=4 attn_r13=5 attn_r14=6 attn_r15=7
15-13:3	src1	Select source for `fsm_input[1]`. val1 is used to select which bit to use. gio_in: Input is selected from `gio_in[31:0]` flag: Input is selected from the four ALU flags reg_lo: Input is selected from bit[0] in general reg r7..r0 xor: Input is selected from the four xor signals statin_lo: Input is selected from `stat_in[15:0]` (lo) gio_out: Input is selected from `spu_gio_out[7:0]` attn_lo: Input is selected from `attn[7:0]` (lo) trigger: Input is selected from 32 Trigger strobes	gio_in=0 flag=2 reg_lo=2 xor=3 statin_lo=4 gio_out=5 attn_lo=5 trigger=6

12-8:5	val1	Select bit from source (src1) to use as `fsm_input[1]`. xor_bus0_r2_0: xor on bus0 and R2 bit 0 xor_bus1_r3_0: xor on bus1 and R3 bit 0 xor_bus0m_r2_0: xor on bus0 (masked with R0) and R2 bit 0 xor_bus1m_r3_0: xor on bus1 (masked with R1) and R3 bit 0 c: ALU c-flag v: ALU v-flag z: ALU z-flag n: ALU n-flag r0: bit[0] of r0 r1: bit[0] of r1 r2: bit[0] of r2 r3: bit[0] of r3 r4: bit[0] of r4 r5: bit[0] of r5 r6: bit[0] of r6 r7: bit[0] of r7 gio_out0: SPU `gio_out[0]` gio_out1: SPU `gio_out[1]` gio_out2: SPU `gio_out[2]` gio_out3: SPU `gio_out[3]` gio_out4: SPU `gio_out[4]` gio_out5: SPU `gio_out[5]` gio_out6: SPU `gio_out[6]` gio_out7: SPU `gio_out[7]` attn_r0: Attention of register R0 attn_r1: Attention of register R1 attn_r2: Attention of register R2 attn_r3: Attention of register R3 attn_r4: Attention of register R4 attn_r5: Attention of register R5 attn_r6: Attention of register R6 attn_r7: Attention of register R7	xor_bus0_r2_0=0 xor_bus1_r3_0=1 xor_bus0m_r2_0=2 xor_bus1m_r3_0=3 c=0 v=1 z=2 n=3 r0=8 r1=9 r2=10 r3=11 r4=12 r5=13 r6=14 r7=15 gio_out0=8 gio_out1=9 gio_out2=10 gio_out3=11 gio_out4=12 gio_out5=13 gio_out6=14 gio_out7=15 attn_r0=0 attn_r1=1 attn_r2=2 attn_r3=3 attn_r4=4 attn_r5=5 attn_r6=6 attn_r7=7
7-5:3	src0	Select source for `fsm_input[0]`. val0 is used to select which bit to use. gio_in: Input is selected from `gio_in[31:0]` flag: Input is selected from the four ALU flags reg_lo: Input is selected from bit[0] in general reg r7..r0 xor: Input is selected from the four xor signals statin_lo: Input is selected from `stat_in[15:0]` (lo) gio_out: Input is selected from `spu_gio_out[7:0]` attn_lo: Input is selected from `attn[7:0]` (lo) trigger: Input is selected from 32 Trigger strobes	gio_in=0 flag=2 reg_lo=2 xor=3 statin_lo=4 gio_out=5 attn_lo=5 trigger=6

4-0:5	val0	Select bit from source (src0) to use as `fsm_input[0]`. xor_bus0_r2_0: xor on bus0 and R2 bit 0 xor_bus1_r3_0: xor on bus1 and R3 bit 0 xor_bus0m_r2_0: xor on bus0 (masked with R0) and R2 bit 0 xor_bus1m_r3_0: xor on bus1 (masked with R1) and R3 bit 0 c: ALU c-flag v: ALU v-flag z: ALU z-flag n: ALU n-flag r0: bit[0] of r0 r1: bit[0] of r1 r2: bit[0] of r2 r3: bit[0] of r3 r4: bit[0] of r4 r5: bit[0] of r5 r6: bit[0] of r6 r7: bit[0] of r7 gio_out0: SPU `gio_out[0]` gio_out1: SPU `gio_out[1]` gio_out2: SPU `gio_out[2]` gio_out3: SPU `gio_out[3]` gio_out4: SPU `gio_out[4]` gio_out5: SPU `gio_out[5]` gio_out6: SPU `gio_out[6]` gio_out7: SPU `gio_out[7]` attn_r0: Attention of register R0 attn_r1: Attention of register R1 attn_r2: Attention of register R2 attn_r3: Attention of register R3 attn_r4: Attention of register R4 attn_r5: Attention of register R5 attn_r6: Attention of register R6 attn_r7: Attention of register R7	xor_bus0_r2_0=0 xor_bus1_r3_0=1 xor_bus0m_r2_0=2 xor_bus1m_r3_0=3 c=0 v=1 z=2 n=3 r0=8 r1=9 r2=10 r3=11 r4=12 r5=13 r6=14 r7=15 gio_out0=8 gio_out1=9 gio_out2=10 gio_out3=11 gio_out4=12 gio_out5=13 gio_out6=14 gio_out7=15 attn_r0=0 attn_r1=1 attn_r2=2 attn_r3=3 attn_r4=4 attn_r5=5 attn_r6=6 attn_r7=7

25.25.6 rw_fsm_inputs7_4

Address	0x50
Default	
Type	Read/Write
Description	rw_fsm_inputs3_0 and rw_fsm_inputs7_4 are used to select eight inputs. These eight inputs are delayed one clock cycle and will thus be 5 ns delayed. Each FSM state then selects four of these eight inputs.

Bit(s)	Name	Description	Value
31-29:3	src7	Select source for `fsm_input[7]`. val7 is used to select which bit to use. gio_in: Input is selected from `gio_in[31:0]` flag: Input is selected from the four ALU flags reg_hi: Input is selected from bit[0] in general reg r15..r8 xor: Input is selected from the four xor signals statin_hi: Input is selected from `stat_in[31:16]` (hi) gio_out: Input is selected from `spu_gio_out[7:0]` attn_hi: Input is selected from `attn[15:8]` (hi) trigger: Input is selected from 32 Trigger strobes	gio_in=0 flag=2 reg_hi=2 xor=3 statin_hi=4 gio_out=5 attn_hi=5 trigger=6

28-24:5	val7	Select bit from source (src7) to use as `fsm_input[7]`. xor_bus0_r2_0: xor on bus0 and R2 bit 0 xor_bus1_r3_0: xor on bus1 and R3 bit 0 xor_bus0m_r2_0: xor on bus0 (masked with R0) and R2 bit 0 xor_bus1m_r3_0: xor on bus1 (masked with R1) and R3 bit 0 c: ALU c-flag v: ALU v-flag z: ALU z-flag n: ALU n-flag r8: bit[0] of r8 r9: bit[0] of r9 r10: bit[0] of r10 r11: bit[0] of r11 r12: bit[0] of r12 r13: bit[0] of r13 r14: bit[0] of r14 r15: bit[0] of r15 gio_out0: SPU `gio_out[0]` gio_out1: SPU `gio_out[1]` gio_out2: SPU `gio_out[2]` gio_out3: SPU `gio_out[3]` gio_out4: SPU `gio_out[4]` gio_out5: SPU `gio_out[5]` gio_out6: SPU `gio_out[6]` gio_out7: SPU `gio_out[7]` attn_r8: Attention of register R8 attn_r9: Attention of register R9 attn_r10: Attention of register R10 attn_r11: Attention of register R11 attn_r12: Attention of register R12 attn_r13: Attention of register R13 attn_r14: Attention of register R14 attn_r15: Attention of register R15	xor_bus0_r2_0=0 xor_bus1_r3_0=1 xor_bus0m_r2_0=2 xor_bus1m_r3_0=3 c=0 v=1 z=2 n=3 r8=8 r9=9 r10=10 r11=11 r12=12 r13=13 r14=14 r15=15 gio_out0=8 gio_out1=9 gio_out2=10 gio_out3=11 gio_out4=12 gio_out5=13 gio_out6=14 gio_out7=15 attn_r8=0 attn_r9=1 attn_r10=2 attn_r11=3 attn_r12=4 attn_r13=5 attn_r14=6 attn_r15=7
23-21:3	src6	Select source for `fsm_input[6]`. val6 is used to select which bit to use. gio_in: Input is selected from `gio_in[31:0]` flag: Input is selected from the four ALU flags reg_hi: Input is selected from bit[0] in general reg r15..r8 xor: Input is selected from the four xor signals statin_hi: Input is selected from `stat_in[31:16]` (hi) gio_out: Input is selected from `spu_gio_out[7:0]` attn_hi: Input is selected from `attn[15:8]` (hi) trigger: Input is selected from 32 Trigger strobes	gio_in=0 flag=2 reg_hi=2 xor=3 statin_hi=4 gio_out=5 attn_hi=5 trigger=6

20-16:5	val6	Select bit from source (src6) to use as `fsm_input[6]`. xor_bus0_r2_0: xor on bus0 and R2 bit 0 xor_bus1_r3_0: xor on bus1 and R3 bit 0 xor_bus0m_r2_0: xor on bus0 (masked with R0) and R2 bit 0 xor_bus1m_r3_0: xor on bus1 (masked with R1) and R3 bit 0 c: ALU c-flag v: ALU v-flag z: ALU z-flag n: ALU n-flag r8: bit[0] of r8 r9: bit[0] of r9 r10: bit[0] of r10 r11: bit[0] of r11 r12: bit[0] of r12 r13: bit[0] of r13 r14: bit[0] of r14 r15: bit[0] of r15 gio_out0: SPU `gio_out[0]` gio_out1: SPU `gio_out[1]` gio_out2: SPU `gio_out[2]` gio_out3: SPU `gio_out[3]` gio_out4: SPU `gio_out[4]` gio_out5: SPU `gio_out[5]` gio_out6: SPU `gio_out[6]` gio_out7: SPU `gio_out[7]` attn_r8: Attention of register R8 attn_r9: Attention of register R9 attn_r10: Attention of register R10 attn_r11: Attention of register R11 attn_r12: Attention of register R12 attn_r13: Attention of register R13 attn_r14: Attention of register R14 attn_r15: Attention of register R15	xor_bus0_r2_0=0 xor_bus1_r3_0=1 xor_bus0m_r2_0=2 xor_bus1m_r3_0=3 c=0 v=1 z=2 n=3 r8=8 r9=9 r10=10 r11=11 r12=12 r13=13 r14=14 r15=15 gio_out0=8 gio_out1=9 gio_out2=10 gio_out3=11 gio_out4=12 gio_out5=13 gio_out6=14 gio_out7=15 attn_r8=0 attn_r9=1 attn_r10=2 attn_r11=3 attn_r12=4 attn_r13=5 attn_r14=6 attn_r15=7
15-13:3	src5	Select source for `fsm_input[5]`. val5 is used to select which bit to use. gio_in: Input is selected from `gio_in[31:0]` flag: Input is selected from the four ALU flags reg_lo: Input is selected from bit[0] in general reg r7..r0 xor: Input is selected from the four xor signals statin_lo: Input is selected from `stat_in[15:0]` (lo) gio_out: Input is selected from `spu_gio_out[7:0]` attn_lo: Input is selected from `attn[7:0]` (lo) trigger: Input is selected from 32 Trigger strobes	gio_in=0 flag=2 reg_lo=2 xor=3 statin_lo=4 gio_out=5 attn_lo=5 trigger=6

12-8:5	val5	Select bit from source (src5) to use as `fsm_input[5]`. xor_bus0_r2_0: xor on bus0 and R2 bit 0 xor_bus1_r3_0: xor on bus1 and R3 bit 0 xor_bus0m_r2_0: xor on bus0 (masked with R0) and R2 bit 0 xor_bus1m_r3_0: xor on bus1 (masked with R1) and R3 bit 0 c: ALU c-flag v: ALU v-flag z: ALU z-flag n: ALU n-flag r0: bit[0] of r0 r1: bit[0] of r1 r2: bit[0] of r2 r3: bit[0] of r3 r4: bit[0] of r4 r5: bit[0] of r5 r6: bit[0] of r6 r7: bit[0] of r7 gio_out0: SPU `gio_out[0]` gio_out1: SPU `gio_out[1]` gio_out2: SPU `gio_out[2]` gio_out3: SPU `gio_out[3]` gio_out4: SPU `gio_out[4]` gio_out5: SPU `gio_out[5]` gio_out6: SPU `gio_out[6]` gio_out7: SPU `gio_out[7]` attn_r0: Attention of register R0 attn_r1: Attention of register R1 attn_r2: Attention of register R2 attn_r3: Attention of register R3 attn_r4: Attention of register R4 attn_r5: Attention of register R5 attn_r6: Attention of register R6 attn_r7: Attention of register R7	xor_bus0_r2_0=0 xor_bus1_r3_0=1 xor_bus0m_r2_0=2 xor_bus1m_r3_0=3 c=0 v=1 z=2 n=3 r0=8 r1=9 r2=10 r3=11 r4=12 r5=13 r6=14 r7=15 gio_out0=8 gio_out1=9 gio_out2=10 gio_out3=11 gio_out4=12 gio_out5=13 gio_out6=14 gio_out7=15 attn_r0=0 attn_r1=1 attn_r2=2 attn_r3=3 attn_r4=4 attn_r5=5 attn_r6=6 attn_r7=7
7-5:3	src4	Select source for `fsm_input[4]`. val4 is used to select which bit to use. gio_in: Input is selected from `gio_in[31:0]` flag: Input is selected from the four ALU flags reg_lo: Input is selected from bit[0] in general reg r7..r0 xor: Input is selected from the four xor signals statin_lo: Input is selected from `stat_in[15:0]` (lo) gio_out: Input is selected from `spu_gio_out[7:0]` attn_lo: Input is selected from `attn[7:0]` (lo) trigger: Input is selected from 32 Trigger strobes	gio_in=0 flag=2 reg_lo=2 xor=3 statin_lo=4 gio_out=5 attn_lo=5 trigger=6

4-0:5	val4	Select bit from source (src4) to use as `fsm_input[4]`. xor_bus0_r2_0: xor on bus0 and R2 bit 0 xor_bus1_r3_0: xor on bus1 and R3 bit 0 xor_bus0m_r2_0: xor on bus0 (masked with R0) and R2 bit 0 xor_bus1m_r3_0: xor on bus1 (masked with R1) and R3 bit 0 c: ALU c-flag v: ALU v-flag z: ALU z-flag n: ALU n-flag r0: bit[0] of r0 r1: bit[0] of r1 r2: bit[0] of r2 r3: bit[0] of r3 r4: bit[0] of r4 r5: bit[0] of r5 r6: bit[0] of r6 r7: bit[0] of r7 gio_out0: SPU `gio_out[0]` gio_out1: SPU `gio_out[1]` gio_out2: SPU `gio_out[2]` gio_out3: SPU `gio_out[3]` gio_out4: SPU `gio_out[4]` gio_out5: SPU `gio_out[5]` gio_out6: SPU `gio_out[6]` gio_out7: SPU `gio_out[7]` attn_r0: Attention of register R0 attn_r1: Attention of register R1 attn_r2: Attention of register R2 attn_r3: Attention of register R3 attn_r4: Attention of register R4 attn_r5: Attention of register R5 attn_r6: Attention of register R6 attn_r7: Attention of register R7	xor_bus0_r2_0=0 xor_bus1_r3_0=1 xor_bus0m_r2_0=2 xor_bus1m_r3_0=3 c=0 v=1 z=2 n=3 r0=8 r1=9 r2=10 r3=11 r4=12 r5=13 r6=14 r7=15 gio_out0=8 gio_out1=9 gio_out2=10 gio_out3=11 gio_out4=12 gio_out5=13 gio_out6=14 gio_out7=15 attn_r0=0 attn_r1=1 attn_r2=2 attn_r3=3 attn_r4=4 attn_r5=5 attn_r6=6 attn_r7=7

25.25.7 rw_gio_out

Address	0x54
Default	0x00000000
Type	Read/Write
Description	This is the output register for the SPU. SPU uses GOUT (P5).

25.25.8 rw_bus0_out

Address	0x58
Default	
Type	Read/Write
Description	This is the bus0_out register for the SPU. SPU uses B0OUT (P6).

25.25.9 rw_bus1_out

Address	0x5c
Default	
Type	Read/Write
Description	This is the bus1_out register for the SPU. SPU uses B1OUT (P7).

25.25.10 r_gio_in

Address	0x60
Default	
Type	Read
Description	The gio_in to the SPU, also readable from SPU as GIN (P8).

25.25.11 r_bus0_in

Address	0x64
Default	
Type	Read
Description	The bus0_in to the SPU, also readable from SPU as B0IN (P9).

25.25.12 r_bus1_in

Address	0x68
Default	
Type	Read
Description	The bus1_in to the SPU, also readable from SPU as B1IN (P10).

25.25.13 rw_gio_out_set

Address	0x6c
Default	
Type	Read/Write
Description	Bit mask for setting bits in the SPU gio_out.

25.25.14 rw_gio_out_clr

Address	0x70
Default	
Type	Read/Write
Description	Bit mask for clearing bits in the SPU gio_out.

25.25.15 rs_wr_stat/r_wr_stat

Address	0x74/0x78
Default	
Type	Read with side effects/Read
Description	When SPU owner writes to R0-R15 the corresponding bit is set in this register. The SPU can clear bits with ALU operation on register WSTS (P13). Reading rs_wr_stat clears all bits in it.

Bit(s)	Name	Description	Value
15	r15	Write status for R15. no: No value has been written to R15 yes: A value has been written to R15	no=0 yes=1
14	r14	Write status for R14. no: No value has been written to R14 yes: A value has been written to R14	no=0 yes=1
13	r13	Write status for R13. no: No value has been written to R13 yes: A value has been written to R13	no=0 yes=1
12	r12	Write status for R12. no: No value has been written to R12 yes: A value has been written to R12	no=0 yes=1
11	r11	Write status for R11. no: No value has been written to R11 yes: A value has been written to R11	no=0 yes=1
10	r10	Write status for R10. no: No value has been written to R10 yes: A value has been written to R10	no=0 yes=1
9	r9	Write status for R9. no: No value has been written to R9 yes: A value has been written to R9	no=0 yes=1
8	r8	Write status for R8. no: No value has been written to R8 yes: A value has been written to R8	no=0 yes=1
7	r7	Write status for R7. no: No value has been written to R7 yes: A value has been written to R7	no=0 yes=1
6	r6	Write status for R6. no: No value has been written to R6 yes: A value has been written to R6	no=0 yes=1
5	r5	Write status for R5. no: No value has been written to R5 yes: A value has been written to R5	no=0 yes=1

4	r4	Write status for R4. no: No value has been written to R4 yes: A value has been written to R4	no=0 yes=1
3	r3	Write status for R3. no: No value has been written to R3 yes: A value has been written to R3	no=0 yes=1
2	r2	Write status for R2. no: No value has been written to R2 yes: A value has been written to R2	no=0 yes=1
1	r1	Write status for R1. no: No value has been written to R1 yes: A value has been written to R1	no=0 yes=1
0	r0	Write status for R0. no: No value has been written to R0 yes: A value has been written to R0	no=0 yes=1

25.25.16 r_reg_indexed_by_bus0_in

Address	0x7c
Default	
Type	Read
Description	This register holds the value of general register R0 to R15 depending on the 4 lowest bits of bus0_in. The SPU can read this register using INDEX (P15).

25.25.17 r_stat_in

Address	0x80
Default	
Type	Read
Description	This register holds the value of status_in port connected to the SPU. The SPU can read this register using STATIN (P11). spu0_stat_in[31:30] = mc_owned_spu0, mc_busy_spu0 ## MC spu0_stat_in[29] = scrc_in0_crc_err ## SCRC_IN0 spu0_stat_in[28] = scrc_out0_dif_out.data ## SCRC_OUT0 spu0_stat_in[27] = sync_clk_ext ## Sync clk 12 spu0_stat_in[26:23] = spu1_gio_out[3:0] ## SPU1 spu0_stat_in[22:20] = dmc_in0_cmd_rdy, dmc_in0_full, dmc_in0_sth ## DMC_IN0 spu0_stat_in[19:16] = timer_grp2_strb ## TG2 spu0_stat_in[15] = crc_par0_correct ## PCRC0 spu0_stat_in[14:13] = dmc_out0_cmd_rdy, dmc_out0_cmd_rq ## DMC_OUT0 spu0_stat_in[12:8] = dmc_out0_dif.last, dmc_out0_dv, dmc_out0_eop, dmc_out0_dth, dmc_out0_all_avail ## DMC_OUT0 spu0_stat_in[7] = fifo_in0_dif_in.rdy ## FIFO_IN0 spu0_stat_in[6:4] = fifo_out0_all, fifo_out0_dif_out.rdy, fifo_out0_dif_out.last ## FIFO_OUT0 spu0_stat_in[3:0] = timer_grp0_strb ## TG0 spu1_stat_in[31:30] = mc_owned_spu1, mc_busy_spu1 ## MC spu1_stat_in[29] = scrc_in1_crc_err ## SCRC_IN1 spu1_stat_in[28] = scrc_out1_dif_out.data ## SCRC_OUT1 spu1_stat_in[27] = sync_clk_ext ## Sync clk 12 spu1_stat_in[26:23] = spu0_gio_out[3:0] ## SPU0 spu1_stat_in[22:20] = dmc_in1_cmd_rdy, dmc_in1_full, dmc_in1_sth ## DMC_IN1 spu1_stat_in[19:16] = timer_grp3_strb ## TG3 spu1_stat_in[15] = crc_par1_correct ## PCRC1 spu1_stat_in[14:13] = dmc_out1_cmd_rdy, dmc_out1_cmd_rq ## DMC_OUT1 spu1_stat_in[12:8] = dmc_out1_dif.last, dmc_out1_dv, dmc_out1_eop, dmc_out1_dth, dmc_out1_all_avail ## DMC_OUT1 spu1_stat_in[7] = fifo_in1_dif_in.rdy ## FIFO_IN1 spu1_stat_in[6:4] = fifo_out1_all, fifo_out1_dif_out.rdy, fifo_out1_dif_out.last ## FIFO_OUT1 spu1_stat_in[3:0] = timer_grp1_strb ## TG1

Bit(s)	Name	Description	Value
31	mc_owned	Indicates if the MC is owned by this SPU.	
30	mc_busy	Indicates if the MC is busy performing an operation issued by this SPU.	
29	scrc_in_err	CRC error signal from scrc_in (scrc0 to SPU0, scrc1 to SPU1).	
28	scrc_out_data	Serial data from scrc_out (scrc0 to SPU0, scrc1 to SPU1).	

27	sync_clk12	Synchronized 12 MHz clock.	
26-23:4	spu_gio_out	`gio_out` signals [3:0] from other SPU.	
22	dmc_in_cmd_rdy	`cmd_rdy` signal from dmc_in (dmc_in0 to SPU0, dmc_in1 to SPU1).	
21	dmc_in_full	`full` signal from dmc_in (dmc_in0 to SPU0, dmc_in1 to SPU1).	
20	dmc_in_sth	Space threshold (`sth`) signal from dmc_in (dmc_in0 to SPU0, dmc_in1 to SPU1).	
19-16:4	timer_grp_hi	Timer strobe signals from timer_grp[3:2] (SPU0 timer_grp2, SPU1 timer_grp3).	
15	pcrc_correct	`crc_correct` signal from pcrc_in (pcrc_in0 to SPU0, pcrc_in1 to SPU1).	
14	dmc_out_cmd_rdy	`cmd_rdy` signal from dmc_out (dmc_out0 to SPU0, dmc_out1 to SPU1).	
13	dmc_out_cmd_rq	`cmd_rq` signal from dmc_out (dmc_out0 to SPU0, dmc_out1 to SPU1).	
12	dmc_out_last	`dif.last` signal from dmc_out (dmc_out0 to SPU0, dmc_out1 to SPU1).	
11	dmc_out_dv	`dv` signal from dmc_out (dmc_out0 to SPU0, dmc_out1 to SPU1).	
10	dmc_out_eop	`eop` signal from dmc_out (dmc_out0 to SPU0, dmc_out1 to SPU1).	
9	dmc_out_dth	Data threshold (`dth`) signal from dmc_out (dmc_out0 to SPU0, dmc_out1 to SPU1).	
8	dmc_out_all	`all_avail` signal from dmc_out (dmc_out0 to SPU0, dmc_out1 to SPU1).	
7	fifo_in_rdy	`rdy` signal from fifo_in (fifo_in0 to SPU0, fifo_in1 to SPU1).	
6	fifo_out_all	`all` signal from fifo_out (fifo_out0 to SPU0, fifo_out1 to SPU1).	
5	fifo_out_rdy	`rdy` signal from fifo_out (fifo_out0 to SPU0, fifo_out1 to SPU1).	
4	fifo_out_last	`last` signal from fifo_out (fifo_out0 to SPU0, fifo_out1 to SPU1).	
3-0:4	timer_grp_lo	Timer strobe signals from timer_grp[1:0] (SPU0 timer_grp0, SPU1 timer_grp1).	

25.25.18 r_trigger_in

Address	0x84
Default	
Type	Read
Description	This register holds the value of trigger_in port connected to the SPU. The SPU can read this register using TRIGGER (P12).

25.25.19 r_special_stat

Address	0x88
Default	
Type	Read
Description	This register holds the value of flags, xor signals and selected insignals for FSM mode. The SPU can read this register using SPECS (P14).

Bit(s)	Name	Description	Value
19	event3	Event 3 signal.	
18	event2	Event 2 signal.	
17	event1	Event 1 signal.	
16	event0	Event 0 signal.	
15	fsm_in7	Value of selected input for fsm_in7 (selected by rw_fsm_inputs7_4.src7 and rw_fsm_inputs7_4.val7.	
14	fsm_in6	Value of selected input for fsm_in6 (selected by rw_fsm_inputs7_4.src6 and rw_fsm_inputs7_4.val6.	
13	fsm_in5	Value of selected input for fsm_in5 (selected by rw_fsm_inputs7_4.src5 and rw_fsm_inputs7_4.val5.	
12	fsm_in4	Value of selected input for fsm_in4 (selected by rw_fsm_inputs7_4.src4 and rw_fsm_inputs7_4.val4.	
11	fsm_in3	Value of selected input for fsm_in3 (selected by rw_fsm_inputs3_0.src3 and rw_fsm_inputs3_0.val3.	
10	fsm_in2	Value of selected input for fsm_in2 (selected by rw_fsm_inputs3_0.src2 and rw_fsm_inputs3_0.val2.	
9	fsm_in1	Value of selected input for fsm_in1 (selected by rw_fsm_inputs3_0.src1 and rw_fsm_inputs3_0.val1.	
8	fsm_in0	Value of selected input for fsm_in0 (selected by rw_fsm_inputs3_0.src0 and rw_fsm_inputs3_0.val0.	
7	xor_bus1m_r3_0	Value of xor calculation on spu_bus1_in (masked with R1) and R3 bit 0.	
6	xor_bus0m_r2_0	Value of xor calculation on spu_bus0_in (masked with R0) and R2 bit 0.	
5	xor_bus1_r3_0	Value of xor calculation on spu_bus1_in and R3 bit 0.	
4	xor_bus0_r2_0	Value of xor calculation on spu_bus0_in and R2 bit 0.	
3	n_flag	ALU n-flag.	
2	z_flag	ALU z-flag.	
1	v_flag	ALU v-flag.	
0	c_flag	ALU c-flag.	

25.25.20 rw_reg_access

Address	0x8c
Default	0x00000000
Type	Read/Write
Description	This is REGA (P2) special register. Used as address register and also has imm_hi when writing with RWQ.

Bit(s)	Name	Description	Value
31-16:16	imm_hi	This value is used as imm_hi value when a RWQ instruction is performed.	
12-0:13	addr	Register address.	

25.25.21 rw_event_cfg

Address	0x90, 0x94, 0x98, 0x9c
Default	
Type	Read/Write
Description	SPU event configuration register.

Bit(s)	Name	Description	Value
17	gt_inv	Invert result from greater than operator.	
16	gt_en	Use greater than operator.	
15	eq_inv	Invert result from equal operator.	
14	eq_en	Use equal operator.	
13-12:2	src	Event source. gio_in: SPU gio_in signals wsts_gioout_spec: wsts[15:8] or wsts[7:0], gio_out[7:0], fsm_inp[7:0], xor[3:0], event[3:0] stat_in: SPU `status_in` signals trig: Trigger signals	gio_in=0 wsts_gioout_spec=1 stat_in=2 trig=3
11-0:12	addr	Address to which the event will jump if event is taken.	

25.25.22 rw_event_mask

Address	0xa0, 0xa4, 0xa8, 0xac
Default	
Type	Read/Write
Description	SPU event mask register.

25.25.23 rw_event_val

Address	0xb0, 0xb4, 0xb8, 0xbc
Default	
Type	Read/Write
Description	SPU event value register.

25.25.24 rw_event_ret

Address	0xc0
Default	
Type	Read/Write
Description	SPU event return address register. The address to the state where the event is taken is stored in addr. It is possible for the SPU to use E12 to jump back to this state, creating a kind of event subroutine.

Bit(s)	Name	Description	Value
11-0:12	addr	Address.	

25.25.25 r_trace

Address	0xc4
Default	
Type	Read
Description	Trace register for SPU. This register is only updated when rw_ctrl.en equals yes.

Bit(s)	Name	Description	Value
31-20:12	fsm_addr	The FSM program counter.	
17-6:12	seq_addr	The sequential program counter.	
5	n_flag	ALU n-flag.	
4	z_flag	ALU z-flag.	
3	v_flag	ALU v-flag.	
2	c_flag	ALU c-flag.	
1	en	The SPU enable status.	
0	fsm	The SPU mode.	

25.25.26 r_fsm_trace

Address	0xc8
Default	
Type	Read
Description	Trace register for SPU, FSM mode. This register is only updated when rw_ctrl.en equals yes.

Bit(s)	Name	Description	Value
31-20:12	fsm_addr	The FSM program counter.	
18-11:8	gio_out	`gio_out[7:0]` signals from SPU.	
10	event3	Event 3 signal.	
9	event2	Event 2 signal.	
8	event1	Event 1 signal.	
7	event0	Event 0 signal.	
6	inp3	The selected fsm `inp[3]` signal.	
5	inp2	The selected fsm `inp[2]` signal.	
4	inp1	The selected fsm `inp[1]` signal.	
3	inp0	The selected fsm `inp[0]` signal.	
2	tmr_done	The FSM timer done signal.	
1	en	The SPU enable status.	
0	fsm	The SPU mode.	

25.25.27 rw_brp

Address	0xcc, 0xd0, 0xd4, 0xd8
Default	0x00000000, 0x00000000, 0x00000000, 0x00000000
Type	Read/Write
Description	SPU breakpoint register, the SPU will halt if the breakpoint is enabled (en) and when address (selected by fsm) matches addr.

Bit(s)	Name	Description	Value
13	en	Enable this breakpoint. no: The breakpoint is not enabled yes: The breakpoint is enabled	no=0 yes=1
12	fsm	If set, the breakpoint address is a FSM address. no: The breakpoint is for sequential mode yes: The breakpoint is for FSM mode	no=0 yes=1
11-0:12	addr	The breakpoint address.	

25.26 iop_sw_cfg

Instance	Base Address
iop_sw_cfg	0xb0021100

25.26.1 rw_crc_par0_owner

Address	0x0
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of crc_par0.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of crc_par0. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.2 rw_crc_par1_owner

Address	0x4
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of crc_par1.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of crc_par1. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.3 rw_dmc_in0_owner

Address	0x8
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of dmc_in0.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of dmc_in0. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.4 rw_dmc_in1_owner

Address	0xc
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of dmc_in1.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of dmc_in1. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.5 rw_dmc_out0_owner

Address	0x10
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of dmc_out0.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of dmc_out0. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.6 rw_dmc_out1_owner

Address	0x14
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of dmc_out1.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of dmc_out1. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.7 rw_fifo_in0_owner

Address	0x18
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of fifo_in0.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of fifo_in0. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.8 rw_fifo_in0_extra_owner

Address	0x1c
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of fifo_in0_extra.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of fifo_in0_extra. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.9 rw_fifo_in1_owner

Address	0x20
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of fifo_in1.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of fifo_in1. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.10 rw_fifo_in1_extra_owner

Address	0x24
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of fifo_in1_extra.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of fifo_in1_extra. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.11 rw_fifo_out0_owner

Address	0x28
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of fifo_out0.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of fifo_out0. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.12 rw_fifo_out0_extra_owner

Address	0x2c
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of fifo_out0_extra.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of fifo_out0_extra. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.13 rw_fifo_out1_owner

Address	0x30
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of fifo_out1.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of fifo_out1. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.14 rw_fifo_out1_extra_owner

Address	0x34
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of fifo_out1_extra.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of fifo_out1_extra. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.15 rw_sap_in_owner

Address	0x38
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of sap_in.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of sap_in. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.16 rw_sap_out_owner

Address	0x3c
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of sap_out.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of sap_out. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.17 rw_scrc_in0_owner

Address	0x40
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of scrc_in0.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of scrc_in0. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.18 rw_scrc_in1_owner

Address	0x44
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of scrc_in1.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of scrc_in1. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.19 rw_scrc_out0_owner

Address	0x48
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of scrc_out0.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of scrc_out0. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.20 rw_scrc_out1_owner

Address	0x4c
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of scrc_out1.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of scrc_out1. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.21 rw_spu0_owner

Address	0x50
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of spu0.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of spu0. cpu: cpu is owner mpu: mpu is owner spu1: spu1 is owner	cpu=0 mpu=1 spu1=3

25.26.22 rw_spu1_owner

Address	0x54
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of spu1.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of spu1. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner	cpu=0 mpu=1 spu0=2

25.26.23 rw_timer_grp0_owner

Address	0x58
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of timer_grp0.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of timer_grp0. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.24 rw_timer_grp1_owner

Address	0x5c
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of timer_grp1.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of timer_grp1. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.25 rw_timer_grp2_owner

Address	0x60
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of timer_grp2.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of timer_grp2. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.26 rw_timer_grp3_owner

Address	0x64
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of timer_grp3.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of timer_grp3. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.27 rw_trigger_grp0_owner

Address	0x68
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of trigger_grp0.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of trigger_grp0. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.28 rw_trigger_grp1_owner

Address	0x6c
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of trigger_grp1.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of trigger_grp1. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.29 rw_trigger_grp2_owner

Address	0x70
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of trigger_grp2.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of trigger_grp2. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.30 rw_trigger_grp3_owner

Address	0x74
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of trigger_grp3.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of trigger_grp3. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.31 rw_trigger_grp4_owner

Address	0x78
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of trigger_grp4.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of trigger_grp4. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.32 rw_trigger_grp5_owner

Address	0x7c
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of trigger_grp5.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of trigger_grp5. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.33 rw_trigger_grp6_owner

Address	0x80
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of trigger_grp6.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of trigger_grp6. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.34 rw_trigger_grp7_owner

Address	0x84
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of trigger_grp7.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of trigger_grp7. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.26.35 rw_bus0_mask

Address	0x88
Default	0x00000000
Type	Read/Write
Description	Mask register for output BUS0.

Bit(s)	Name	Description	Value
31-24:8	byte3	Mask value.	
23-16:8	byte2	Mask value.	
15-8:8	byte1	Mask value.	
7-0:8	byte0	Mask value.	

25.26.36 rw_bus0_oe_mask

Address	0x8c
Default	0x00000000
Type	Read/Write
Description	Mask register for OE, BUS0.

Bit(s)	Name	Description	Value
3	byte3	Mask value.	
2	byte2	Mask value.	
1	byte1	Mask value.	
0	byte0	Mask value.	

25.26.37 rw_bus1_mask

Address	0x90
Default	0x00000000
Type	Read/Write
Description	Mask register for output BUS1.

Bit(s)	Name	Description	Value
31-24:8	byte3	Mask value.	
23-16:8	byte2	Mask value.	
15-8:8	byte1	Mask value.	
7-0:8	byte0	Mask value.	

25.26.38 rw_bus1_oe_mask

Address	0x94
Default	0x00000000
Type	Read/Write
Description	Mask register for OE, BUS1.

Bit(s)	Name	Description	Value
3	byte3	Mask value.	
2	byte2	Mask value.	
1	byte1	Mask value.	
0	byte0	Mask value.	

25.26.39 rw_gio_mask

Address	0x98
Default	0x00000000
Type	Read/Write
Description	Mask register for output GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Mask value.	

25.26.40 rw_gio_oe_mask

Address	0x9c
Default	0x00000000
Type	Read/Write
Description	Mask register for OE, GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Mask value.	

25.26.41 rw_pinmapping

Address	0xa0
Default	0x55555555
Type	Read/Write
Description	This register controls how to map bus0, bus1 and gio to pins.

Bit(s)	Name	Description	Value
31-30:2	gio31_28	Mapping of gio[31:28]. a: Mapping a, port PE[15:12] is selected b: Mapping b, port PD[7:4] is selected	a=1 b=2
29-28:2	gio27_24	Mapping of gio[27:24]. a: Mapping a, port PC[15:12] is selected b: Mapping b, port PE[7:4] is selected	a=1 b=2
27-26:2	gio23_20	Mapping of gio[23:20]. a: Mapping a, port PD[15:12] is selected b: Mapping b, port PC[7:4] is selected	a=1 b=2
25-24:2	gio19_16	Mapping of gio[19:16]. a: Mapping a, port PB[15:12] is selected b: Mapping b, port PC[3:0] is selected	a=1 b=2
23-22:2	gio15_12	Mapping of gio[15:12]. a: Mapping a, port PE[11:8] is selected b: Mapping b, port PD[3:0] is selected	a=1 b=2
21-20:2	gio11_8	Mapping of gio[11:8]. a: Mapping a, port PC[11:8] is selected b: Mapping b, port PE[3:0] is selected	a=1 b=2
19-18:2	gio7_4	Mapping of gio[7:4]. a: Mapping a, port PD[11:8] is selected b: Mapping b, port PB[17:16], PC[17:16] is selected	a=1 b=2
17-16:2	gio3_0	Mapping of gio[3:0]. a: Mapping a, port PB[11:8] is selected b: Mapping b, port PD[17:16], PE[17:16] is selected	a=1 b=2
15-14:2	bus1_byte3	Mapping of bus1[31:24]. a: Mapping a, port PB[7:0] is selected b: Mapping b, port PC[15:8] is selected	a=1 b=2
13-12:2	bus1_byte2	Mapping of bus1[23:16]. a: Mapping a, port PB[15:8] is selected b: Mapping b, port PD[15:8] is selected	a=1 b=2
11-10:2	bus1_byte1	Mapping of bus1[15:8]. a: Mapping a, port PC[15:8] is selected b: Mapping b, port PE[15:8] is selected	a=1 b=2
9-8:2	bus1_byte0	Mapping of bus1[7:0]. a: Mapping a, port PC[7:0] is selected b: Mapping b, port PE[7:0] is selected	a=1 b=2

7-6:2	bus0_byte3	Mapping of bus0[31:24]. a: Mapping a, port PE[7:0] is selected b: Mapping b, port PD[15:8] is selected	a=1 b=2
5-4:2	bus0_byte2	Mapping of bus0[23:16]. a: Mapping a, port PD[7:0] is selected b: Mapping b, port PE[15:8] is selected	a=1 b=2
3-2:2	bus0_byte1	Mapping of bus0[15:8]. a: Mapping a, port PC[7:0] is selected b: Mapping b, port PB[15:8] is selected	a=1 b=2
1-0:2	bus0_byte0	Mapping of bus0[7:0]. a: Mapping a, port PB[7:0] is selected b: Mapping b, port PD[7:0] is selected	a=1 b=2

25.26.42 rw_bus_out_cfg

Address	0xa4
Default	0x00000000
Type	Read/Write
Description	This register select HW sources for BUS0, BUS1, BUS0 OE and BUS1 OE.

Bit(s)	Name	Description	Value
23-21:3	bus1_hi_oe	Output enable register for BUS1[31:16]. spu0_gio2: Use SPU0 GIO[2] as OE spu0_gio1: Use SPU0 GIO[1] as OE spu1_gio2: Use SPU1 GIO[2] as OE spu1_gio1: Use SPU1 GIO[1] as OE timer_grp0_tmr1: Use Timer group 0, Timer 1 as OE timer_grp1_tmr1: Use Timer group 1, Timer 1 as OE timer_grp2_tmr1: Use Timer group 2, Timer 1 as OE timer_grp3_tmr1: Use Timer group 3, Timer 1 as OE	spu0_gio2=0 spu0_gio1=1 spu1_gio2=2 spu1_gio1=3 timer_grp0_tmr1=4 timer_grp1_tmr1=5 timer_grp2_tmr1=6 timer_grp3_tmr1=7
20-18:3	bus1_lo_oe	Output enable register for BUS1[15:0]. spu0_gio2: Use SPU0 GIO[2] as OE spu0_gio1: Use SPU0 GIO[1] as OE spu1_gio2: Use SPU1 GIO[2] as OE spu1_gio1: Use SPU1 GIO[1] as OE timer_grp0_tmr1: Use Timer group 0, Timer 1 as OE timer_grp1_tmr1: Use Timer group 1, Timer 1 as OE timer_grp2_tmr1: Use Timer group 2, Timer 1 as OE timer_grp3_tmr1: Use Timer group 3, Timer 1 as OE	spu0_gio2=0 spu0_gio1=1 spu1_gio2=2 spu1_gio1=3 timer_grp0_tmr1=4 timer_grp1_tmr1=5 timer_grp2_tmr1=6 timer_grp3_tmr1=7
17-15:3	bus1_hi	BUS1[31:16]. pdp_out0_lo: Parallel datapath 0 [15:0] pdp_out0_hi: Parallel datapath 0 [31:16] pdp_out1_lo: Parallel datapath 1 [15:0] pdp_out1_hi: Parallel datapath 1 [31:16] pdp_out0_lo_rot8: Parallel datapath 0 rotated 8bits [7:0], [15:8] pdp_out1_hi_rot8: Parallel datapath 1 rotated 8bits [23:16], [31:24] spu1_bus_out0_hi: SPU1 BUS0 out [31:16] spu0_bus_out1_lo: SPU0 BUS1 out [15:0]	pdp_out0_lo=0 pdp_out0_hi=1 pdp_out1_lo=2 pdp_out1_hi=3 pdp_out0_lo_rot8=4 pdp_out1_hi_rot8=5 spu1_bus_out0_hi=6 spu0_bus_out1_lo=7

14-12:3	bus1_lo	BUS1[15:0]. pdp_out0_lo: Parallel datapath 0 [15:0] pdp_out0_hi: Parallel datapath 0 [31:16] pdp_out1_lo: Parallel datapath 1 [15:0] pdp_out1_hi: Parallel datapath 1 [31:16] pdp_out1_lo_rot8: Parallel datapath 1 rotated 8bits [7:0], [15:8] pdp_out0_hi_rot8: Parallel datapath 0 rotated 8bits [23:16], [31:24] spu1_bus_out0_lo: SPU1 BUS0 out [15:0] spu0_bus_out1_hi: SPU0 BUS1 out [31:16]	pdp_out0_lo=0 pdp_out0_hi=1 pdp_out1_lo=2 pdp_out1_hi=3 pdp_out1_lo_rot8=4 pdp_out0_hi_rot8=5 spu1_bus_out0_lo=6 spu0_bus_out1_hi=7
11-9:3	bus0_hi_oe	Output enable register for BUS0[31:16]. spu0_gio0: Use SPU0 GIO[0] as OE spu0_gio1: Use SPU0 GIO[1] as OE spu1_gio0: Use SPU1 GIO[0] as OE spu1_gio1: Use SPU1 GIO[1] as OE timer_grp0_tmr0: Use Timer group 0, Timer 0 as OE timer_grp1_tmr0: Use Timer group 1, Timer 0 as OE timer_grp2_tmr0: Use Timer group 2, Timer 0 as OE timer_grp3_tmr0: Use Timer group 3, Timer 0 as OE	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3 timer_grp0_tmr0=4 timer_grp1_tmr0=5 timer_grp2_tmr0=6 timer_grp3_tmr0=7
8-6:3	bus0_lo_oe	Output enable register for BUS0[15:0]. spu0_gio0: Use SPU0 GIO[0] as OE spu0_gio1: Use SPU0 GIO[1] as OE spu1_gio0: Use SPU1 GIO[0] as OE spu1_gio1: Use SPU1 GIO[1] as OE timer_grp0_tmr0: Use Timer group 0, Timer 0 as OE timer_grp1_tmr0: Use Timer group 1, Timer 0 as OE timer_grp2_tmr0: Use Timer group 2, Timer 0 as OE timer_grp3_tmr0: Use Timer group 3, Timer 0 as OE	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3 timer_grp0_tmr0=4 timer_grp1_tmr0=5 timer_grp2_tmr0=6 timer_grp3_tmr0=7
5-3:3	bus0_hi	BUS0[31:16]. pdp_out0_lo: Parallel datapath 0 [15:0] pdp_out0_hi: Parallel datapath 0 [31:16] pdp_out1_lo: Parallel datapath 1 [15:0] pdp_out1_hi: Parallel datapath 1 [31:16] pdp_out1_lo_rot8: Parallel datapath 1 rotated 8bits [7:0], [15:8] pdp_out0_hi_rot8: Parallel datapath 0 rotated 8bits [23:16], [31:24] spu0_bus_out0_hi: SPU0 BUS0 out [31:16] spu1_bus_out1_lo: SPU1 BUS1 out [15:0]	pdp_out0_lo=0 pdp_out0_hi=1 pdp_out1_lo=2 pdp_out1_hi=3 pdp_out1_lo_rot8=4 pdp_out0_hi_rot8=5 spu0_bus_out0_hi=6 spu1_bus_out1_lo=7

2-0:3	bus0_lo	BUS0[15:0]. pdp_out0_lo: Parallel datapath 0 [15:0] pdp_out0_hi: Parallel datapath 0 [31:16] pdp_out1_lo: Parallel datapath 1 [15:0] pdp_out1_hi: Parallel datapath 1 [31:16] pdp_out0_lo_rot8: Parallel datapath 0 rotated 8bits [7:0], [15:8] pdp_out1_hi_rot8: Parallel datapath 1 rotated 8bits [23:16], [31:24] spu0_bus_out0_lo: SPU0 BUS0 out [15:0] spu1_bus_out1_hi: SPU1 BUS1 out [31:16]	pdp_out0_lo=0 pdp_out0_hi=1 pdp_out1_lo=2 pdp_out1_hi=3 pdp_out0_lo_rot8=4 pdp_out1_hi_rot8=5 spu0_bus_out0_lo=6 spu1_bus_out1_hi=7

25.26.43 rw_gio_out_grp0_cfg

Address	0xa8
Default	0x00000000
Type	Read/Write
Description	This register select HW sources for GIO[3:0] and GIO[3:0] OE.

Bit(s)	Name	Description	Value
23-22:2	gio3_oe	Output enable for GIO[3]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
21-18:4	gio3	GIO[3]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb3: Timer group 0, Timer 3 is selected timer_grp1_strb3: Timer group 1, Timer 3 is selected timer_grp2_strb3: Timer group 2, Timer 3 is selected timer_grp3_strb3: Timer group 3, Timer 3 is selected spu0_g6: SPU0 GIO[6] is selected spu1_g6: SPU1 GIO[6] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb3=10 timer_grp1_strb3=11 timer_grp2_strb3=12 timer_grp3_strb3=13 spu0_g6=14 spu1_g6=15
17-16:2	gio2_oe	Output enable for GIO[2]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3

15-12:4	gio2	GIO[2]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb2: Timer group 0, Timer 2 is selected timer_grp1_strb2: Timer group 1, Timer 2 is selected timer_grp2_strb2: Timer group 2, Timer 2 is selected timer_grp3_strb2: Timer group 3, Timer 2 is selected spu0_g1: SPU0 GIO[1] is selected spu1_g1: SPU1 GIO[1] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb2=10 timer_grp1_strb2=11 timer_grp2_strb2=12 timer_grp3_strb2=13 spu0_g1=14 spu1_g1=15
11-10:2	gio1_oe	Output enable for GIO[1]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
9-6:4	gio1	GIO[1]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb1: Timer group 0, Timer 1 is selected timer_grp1_strb1: Timer group 1, Timer 1 is selected timer_grp2_strb1: Timer group 2, Timer 1 is selected timer_grp3_strb1: Timer group 3, Timer 1 is selected spu0_g0: SPU0 GIO[0] is selected spu1_g0: SPU1 GIO[0] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb1=10 timer_grp1_strb1=11 timer_grp2_strb1=12 timer_grp3_strb1=13 spu0_g0=14 spu1_g0=15

5-4:2	gio0_oe	Output enable for GIO[0]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
3-0:4	gio0	GIO[0]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb0: Timer group 0, Timer 0 is selected timer_grp1_strb0: Timer group 1, Timer 0 is selected timer_grp2_strb0: Timer group 2, Timer 0 is selected timer_grp3_strb0: Timer group 3, Timer 0 is selected spu0_g7: SPU0 GIO[7] is selected spu1_g7: SPU1 GIO[7] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb0=10 timer_grp1_strb0=11 timer_grp2_strb0=12 timer_grp3_strb0=13 spu0_g7=14 spu1_g7=15

25.26.44 rw_gio_out_grp1_cfg

Address	0xac
Default	0x00000000
Type	Read/Write
Description	This register select HW sources for GIO[7:4] and GIO[7:4] OE.

Bit(s)	Name	Description	Value
23-22:2	gio7_oe	Output enable for GIO[7]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
21-18:4	gio7	GIO[7]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb3: Timer group 0, Timer 3 is selected timer_grp1_strb3: Timer group 1, Timer 3 is selected timer_grp2_strb3: Timer group 2, Timer 3 is selected timer_grp3_strb3: Timer group 3, Timer 3 is selected spu0_g4: SPU0 GIO[4] is selected spu1_g4: SPU1 GIO[4] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb3=10 timer_grp1_strb3=11 timer_grp2_strb3=12 timer_grp3_strb3=13 spu0_g4=14 spu1_g4=15
17-16:2	gio6_oe	Output enable for GIO[6]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3

15-12:4	gio6	GIO[6]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb2: Timer group 0, Timer 2 is selected timer_grp1_strb2: Timer group 1, Timer 2 is selected timer_grp2_strb2: Timer group 2, Timer 2 is selected timer_grp3_strb2: Timer group 3, Timer 2 is selected spu0_g3: SPU0 GIO[3] is selected spu1_g3: SPU1 GIO[3] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb2=10 timer_grp1_strb2=11 timer_grp2_strb2=12 timer_grp3_strb2=13 spu0_g3=14 spu1_g3=15
11-10:2	gio5_oe	Output enable for GIO[5]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
9-6:4	gio5	GIO[5]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb1: Timer group 0, Timer 1 is selected timer_grp1_strb1: Timer group 1, Timer 1 is selected timer_grp2_strb1: Timer group 2, Timer 1 is selected timer_grp3_strb1: Timer group 3, Timer 1 is selected spu0_g2: SPU0 GIO[2] is selected spu1_g2: SPU1 GIO[2] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb1=10 timer_grp1_strb1=11 timer_grp2_strb1=12 timer_grp3_strb1=13 spu0_g2=14 spu1_g2=15

5-4:2	gio4_oe	Output enable for GIO[4]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
3-0:4	gio4	GIO[4]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb0: Timer group 0, Timer 0 is selected timer_grp1_strb0: Timer group 1, Timer 0 is selected timer_grp2_strb0: Timer group 2, Timer 0 is selected timer_grp3_strb0: Timer group 3, Timer 0 is selected spu0_g5: SPU0 GIO[5] is selected spu1_g5: SPU1 GIO[5] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb0=10 timer_grp1_strb0=11 timer_grp2_strb0=12 timer_grp3_strb0=13 spu0_g5=14 spu1_g5=15

25.26.45 rw_gio_out_grp2_cfg

Address	0xb0
Default	0x00000000
Type	Read/Write
Description	This register select HW sources for GIO[11:8] and GIO[11:8] OE.

Bit(s)	Name	Description	Value
23-22:2	gio11_oe	Output enable for GIO[11]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
21-18:4	gio11	GIO[11]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb3: Timer group 0, Timer 3 is selected timer_grp1_strb3: Timer group 1, Timer 3 is selected timer_grp2_strb3: Timer group 2, Timer 3 is selected timer_grp3_strb3: Timer group 3, Timer 3 is selected spu0_gioout11: SPU0 GIO[11] is selected spu1_gioout11: SPU1 GIO[11] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb3=10 timer_grp1_strb3=11 timer_grp2_strb3=12 timer_grp3_strb3=13 spu0_gioout11=14 spu1_gioout11=15
17-16:2	gio10_oe	Output enable for GIO[10]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3

15-12:4	gio10	GIO[10]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb2: Timer group 0, Timer 2 is selected timer_grp1_strb2: Timer group 1, Timer 2 is selected timer_grp2_strb2: Timer group 2, Timer 2 is selected timer_grp3_strb2: Timer group 3, Timer 2 is selected spu0_gioout10: SPU0 GIO[10] is selected spu1_gioout10: SPU1 GIO[10] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb2=10 timer_grp1_strb2=11 timer_grp2_strb2=12 timer_grp3_strb2=13 spu0_gioout10=14 spu1_gioout10=15
11-10:2	gio9_oe	Output enable for GIO[9]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
9-6:4	gio9	GIO[9]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb1: Timer group 0, Timer 1 is selected timer_grp1_strb1: Timer group 1, Timer 1 is selected timer_grp2_strb1: Timer group 2, Timer 1 is selected timer_grp3_strb1: Timer group 3, Timer 1 is selected spu0_gioout9: SPU0 GIO[9] is selected spu1_gioout9: SPU1 GIO[9] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb1=10 timer_grp1_strb1=11 timer_grp2_strb1=12 timer_grp3_strb1=13 spu0_gioout9=14 spu1_gioout9=15

5-4:2	gio8_oe	Output enable for GIO[8]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
3-0:4	gio8	GIO[8]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb0: Timer group 0, Timer 0 is selected timer_grp1_strb0: Timer group 1, Timer 0 is selected timer_grp2_strb0: Timer group 2, Timer 0 is selected timer_grp3_strb0: Timer group 3, Timer 0 is selected spu0_gioout8: SPU0 GIO[8] is selected spu1_gioout8: SPU1 GIO[8] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb0=10 timer_grp1_strb0=11 timer_grp2_strb0=12 timer_grp3_strb0=13 spu0_gioout8=14 spu1_gioout8=15

25.26.46 rw_gio_out_grp3_cfg

Address	0xb4
Default	0x00000000
Type	Read/Write
Description	This register select HW sources for GIO[15:12] and GIO[15:12] OE.

Bit(s)	Name	Description	Value
23-22:2	gio15_oe	Output enable for GIO[15]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
21-18:4	gio15	GIO[15]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb3: Timer group 0, Timer 3 is selected timer_grp1_strb3: Timer group 1, Timer 3 is selected timer_grp2_strb3: Timer group 2, Timer 3 is selected timer_grp3_strb3: Timer group 3, Timer 3 is selected spu0_gioout15: SPU0 GIO[15] is selected spu1_gioout15: SPU1 GIO[15] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb3=10 timer_grp1_strb3=11 timer_grp2_strb3=12 timer_grp3_strb3=13 spu0_gioout15=14 spu1_gioout15=15
17-16:2	gio14_oe	Output enable for GIO[14]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3

15-12:4	gio14	GIO[14]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb2: Timer group 0, Timer 2 is selected timer_grp1_strb2: Timer group 1, Timer 2 is selected timer_grp2_strb2: Timer group 2, Timer 2 is selected timer_grp3_strb2: Timer group 3, Timer 2 is selected spu0_gioout14: SPU0 GIO[14] is selected spu1_gioout14: SPU1 GIO[14] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb2=10 timer_grp1_strb2=11 timer_grp2_strb2=12 timer_grp3_strb2=13 spu0_gioout14=14 spu1_gioout14=15
11-10:2	gio13_oe	Output enable for GIO[13]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
9-6:4	gio13	GIO[13]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb1: Timer group 0, Timer 1 is selected timer_grp1_strb1: Timer group 1, Timer 1 is selected timer_grp2_strb1: Timer group 2, Timer 1 is selected timer_grp3_strb1: Timer group 3, Timer 1 is selected spu0_gioout13: SPU0 GIO[13] is selected spu1_gioout13: SPU1 GIO[13] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb1=10 timer_grp1_strb1=11 timer_grp2_strb1=12 timer_grp3_strb1=13 spu0_gioout13=14 spu1_gioout13=15

5-4:2	gio12_oe	Output enable for GIO[12]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
3-0:4	gio12	GIO[12]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb0: Timer group 0, Timer 0 is selected timer_grp1_strb0: Timer group 1, Timer 0 is selected timer_grp2_strb0: Timer group 2, Timer 0 is selected timer_grp3_strb0: Timer group 3, Timer 0 is selected spu0_gioout12: SPU0 GIO[12] is selected spu1_gioout12: SPU1 GIO[12] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb0=10 timer_grp1_strb0=11 timer_grp2_strb0=12 timer_grp3_strb0=13 spu0_gioout12=14 spu1_gioout12=15

25.26.47 rw_gio_out_grp4_cfg

Address	0xb8
Default	0x00000000
Type	Read/Write
Description	This register select HW sources for GIO[19:16] and GIO[19:16] OE.

Bit(s)	Name	Description	Value
23-22:2	gio19_oe	Output enable for GIO[19]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
21-18:4	gio19	GIO[19]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb3: Timer group 0, Timer 3 is selected timer_grp1_strb3: Timer group 1, Timer 3 is selected timer_grp2_strb3: Timer group 2, Timer 3 is selected timer_grp3_strb3: Timer group 3, Timer 3 is selected spu0_gioout19: SPU0 GIO[19] is selected spu1_gioout19: SPU1 GIO[19] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb3=10 timer_grp1_strb3=11 timer_grp2_strb3=12 timer_grp3_strb3=13 spu0_gioout19=14 spu1_gioout19=15
17-16:2	gio18_oe	Output enable for GIO[18]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3

15-12:4	gio18	GIO[18]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb2: Timer group 0, Timer 2 is selected timer_grp1_strb2: Timer group 1, Timer 2 is selected timer_grp2_strb2: Timer group 2, Timer 2 is selected timer_grp3_strb2: Timer group 3, Timer 2 is selected spu0_gioout18: SPU0 GIO[18] is selected spu1_gioout18: SPU1 GIO[18] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb2=10 timer_grp1_strb2=11 timer_grp2_strb2=12 timer_grp3_strb2=13 spu0_gioout18=14 spu1_gioout18=15
11-10:2	gio17_oe	Output enable for GIO[17]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
9-6:4	gio17	GIO[17]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb1: Timer group 0, Timer 1 is selected timer_grp1_strb1: Timer group 1, Timer 1 is selected timer_grp2_strb1: Timer group 2, Timer 1 is selected timer_grp3_strb1: Timer group 3, Timer 1 is selected spu0_gioout17: SPU0 GIO[17] is selected spu1_gioout17: SPU1 GIO[17] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb1=10 timer_grp1_strb1=11 timer_grp2_strb1=12 timer_grp3_strb1=13 spu0_gioout17=14 spu1_gioout17=15

5-4:2	gio16_oe	Output enable for GIO[16]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
3-0:4	gio16	GIO[16]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb0: Timer group 0, Timer 0 is selected timer_grp1_strb0: Timer group 1, Timer 0 is selected timer_grp2_strb0: Timer group 2, Timer 0 is selected timer_grp3_strb0: Timer group 3, Timer 0 is selected spu0_gioout16: SPU0 GIO[16] is selected spu1_gioout16: SPU1 GIO[16] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb0=10 timer_grp1_strb0=11 timer_grp2_strb0=12 timer_grp3_strb0=13 spu0_gioout16=14 spu1_gioout16=15

25.26.48 rw_gio_out_grp5_cfg

Address	0xbc
Default	0x00000000
Type	Read/Write
Description	This register select HW sources for GIO[23:20] and GIO[23:20] OE.

Bit(s)	Name	Description	Value
23-22:2	gio23_oe	Output enable for GIO[23]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
21-18:4	gio23	GIO[23]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb3: Timer group 0, Timer 3 is selected timer_grp1_strb3: Timer group 1, Timer 3 is selected timer_grp2_strb3: Timer group 2, Timer 3 is selected timer_grp3_strb3: Timer group 3, Timer 3 is selected spu0_gioout23: SPU0 GIO[23] is selected spu1_gioout23: SPU1 GIO[23] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb3=10 timer_grp1_strb3=11 timer_grp2_strb3=12 timer_grp3_strb3=13 spu0_gioout23=14 spu1_gioout23=15
17-16:2	gio22_oe	Output enable for GIO[22]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3

15-12:4	gio22	GIO[22]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb2: Timer group 0, Timer 2 is selected timer_grp1_strb2: Timer group 1, Timer 2 is selected timer_grp2_strb2: Timer group 2, Timer 2 is selected timer_grp3_strb2: Timer group 3, Timer 2 is selected spu0_gioout22: SPU0 GIO[22] is selected spu1_gioout22: SPU1 GIO[22] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb2=10 timer_grp1_strb2=11 timer_grp2_strb2=12 timer_grp3_strb2=13 spu0_gioout22=14 spu1_gioout22=15
11-10:2	gio21_oe	Output enable for GIO[21]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
9-6:4	gio21	GIO[21]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb1: Timer group 0, Timer 1 is selected timer_grp1_strb1: Timer group 1, Timer 1 is selected timer_grp2_strb1: Timer group 2, Timer 1 is selected timer_grp3_strb1: Timer group 3, Timer 1 is selected spu0_gioout21: SPU0 GIO[21] is selected spu1_gioout21: SPU1 GIO[21] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb1=10 timer_grp1_strb1=11 timer_grp2_strb1=12 timer_grp3_strb1=13 spu0_gioout21=14 spu1_gioout21=15

5-4:2	gio20_oe	Output enable for GIO[20]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
3-0:4	gio20	GIO[20]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb0: Timer group 0, Timer 0 is selected timer_grp1_strb0: Timer group 1, Timer 0 is selected timer_grp2_strb0: Timer group 2, Timer 0 is selected timer_grp3_strb0: Timer group 3, Timer 0 is selected spu0_gioout20: SPU0 GIO[20] is selected spu1_gioout20: SPU1 GIO[20] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb0=10 timer_grp1_strb0=11 timer_grp2_strb0=12 timer_grp3_strb0=13 spu0_gioout20=14 spu1_gioout20=15

25.26.49 rw_gio_out_grp6_cfg

Address	0xc0
Default	0x00000000
Type	Read/Write
Description	This register select HW sources for GIO[27:24] and GIO[27:24] OE.

Bit(s)	Name	Description	Value
23-22:2	gio27_oe	Output enable for GIO[27]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
21-18:4	gio27	GIO[27]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb3: Timer group 0, Timer 3 is selected timer_grp1_strb3: Timer group 1, Timer 3 is selected timer_grp2_strb3: Timer group 2, Timer 3 is selected timer_grp3_strb3: Timer group 3, Timer 3 is selected spu0_gioout27: SPU0 GIO[27] is selected spu1_gioout27: SPU1 GIO[27] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb3=10 timer_grp1_strb3=11 timer_grp2_strb3=12 timer_grp3_strb3=13 spu0_gioout27=14 spu1_gioout27=15
17-16:2	gio26_oe	Output enable for GIO[26]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3

15-12:4	gio26	GIO[26]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb2: Timer group 0, Timer 2 is selected timer_grp1_strb2: Timer group 1, Timer 2 is selected timer_grp2_strb2: Timer group 2, Timer 2 is selected timer_grp3_strb2: Timer group 3, Timer 2 is selected spu0_gioout26: SPU0 GIO[26] is selected spu1_gioout26: SPU1 GIO[26] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb2=10 timer_grp1_strb2=11 timer_grp2_strb2=12 timer_grp3_strb2=13 spu0_gioout26=14 spu1_gioout26=15
11-10:2	gio25_oe	Output enable for GIO[25]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
9-6:4	gio25	GIO[25]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb1: Timer group 0, Timer 1 is selected timer_grp1_strb1: Timer group 1, Timer 1 is selected timer_grp2_strb1: Timer group 2, Timer 1 is selected timer_grp3_strb1: Timer group 3, Timer 1 is selected spu0_gioout25: SPU0 GIO[25] is selected spu1_gioout25: SPU1 GIO[25] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb1=10 timer_grp1_strb1=11 timer_grp2_strb1=12 timer_grp3_strb1=13 spu0_gioout25=14 spu1_gioout25=15

5-4:2	gio24_oe	Output enable for GIO[24]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
3-0:4	gio24	GIO[24]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb0: Timer group 0, Timer 0 is selected timer_grp1_strb0: Timer group 1, Timer 0 is selected timer_grp2_strb0: Timer group 2, Timer 0 is selected timer_grp3_strb0: Timer group 3, Timer 0 is selected spu0_gioout24: SPU0 GIO[24] is selected spu1_gioout24: SPU1 GIO[24] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb0=10 timer_grp1_strb0=11 timer_grp2_strb0=12 timer_grp3_strb0=13 spu0_gioout24=14 spu1_gioout24=15

25.26.50 rw_gio_out_grp7_cfg

Address	0xc4
Default	0x00000000
Type	Read/Write
Description	This register select HW sources for GIO[31:28] and GIO[31:28] OE.

Bit(s)	Name	Description	Value
23-22:2	gio31_oe	Output enable for GIO[31]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
21-18:4	gio31	GIO[31]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb3: Timer group 0, Timer 3 is selected timer_grp1_strb3: Timer group 1, Timer 3 is selected timer_grp2_strb3: Timer group 2, Timer 3 is selected timer_grp3_strb3: Timer group 3, Timer 3 is selected spu0_gioout31: SPU0 GIO[31] is selected spu1_gioout31: SPU1 GIO[31] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb3=10 timer_grp1_strb3=11 timer_grp2_strb3=12 timer_grp3_strb3=13 spu0_gioout31=14 spu1_gioout31=15
17-16:2	gio30_oe	Output enable for GIO[30]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3

15-12:4	gio30	GIO[30]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb2: Timer group 0, Timer 2 is selected timer_grp1_strb2: Timer group 1, Timer 2 is selected timer_grp2_strb2: Timer group 2, Timer 2 is selected timer_grp3_strb2: Timer group 3, Timer 2 is selected spu0_gioout30: SPU0 GIO[30] is selected spu1_gioout30: SPU1 GIO[30] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb2=10 timer_grp1_strb2=11 timer_grp2_strb2=12 timer_grp3_strb2=13 spu0_gioout30=14 spu1_gioout30=15
11-10:2	gio29_oe	Output enable for GIO[29]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
9-6:4	gio29	GIO[29]. spu0_gioout1: SPU0 GIO[1] is selected spu1_gioout1: SPU1 GIO[1] is selected spu0_gioout3: SPU0 GIO[3] is selected spu1_gioout3: SPU1 GIO[3] is selected spu0_gioout5: SPU0 GIO[5] is selected spu1_gioout5: SPU1 GIO[5] is selected spu0_gioout7: SPU0 GIO[7] is selected spu1_gioout7: SPU1 GIO[7] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb1: Timer group 0, Timer 1 is selected timer_grp1_strb1: Timer group 1, Timer 1 is selected timer_grp2_strb1: Timer group 2, Timer 1 is selected timer_grp3_strb1: Timer group 3, Timer 1 is selected spu0_gioout29: SPU0 GIO[29] is selected spu1_gioout29: SPU1 GIO[29] is selected	spu0_gioout1=0 spu1_gioout1=1 spu0_gioout3=2 spu1_gioout3=3 spu0_gioout5=4 spu1_gioout5=5 spu0_gioout7=6 spu1_gioout7=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb1=10 timer_grp1_strb1=11 timer_grp2_strb1=12 timer_grp3_strb1=13 spu0_gioout29=14 spu1_gioout29=15

5-4:2	gio28_oe	Output enable for GIO[28]. spu0_gio0: SPU0 GIO[0] is selected spu0_gio1: SPU0 GIO[1] is selected spu1_gio0: SPU1 GIO[0] is selected spu1_gio1: SPU1 GIO[1] is selected	spu0_gio0=0 spu0_gio1=1 spu1_gio0=2 spu1_gio1=3
3-0:4	gio28	GIO[28]. spu0_gioout0: SPU0 GIO[0] is selected spu1_gioout0: SPU1 GIO[0] is selected spu0_gioout2: SPU0 GIO[2] is selected spu1_gioout2: SPU1 GIO[2] is selected spu0_gioout4: SPU0 GIO[4] is selected spu1_gioout4: SPU1 GIO[4] is selected spu0_gioout6: SPU0 GIO[6] is selected spu1_gioout6: SPU1 GIO[6] is selected sdp_out0: SDP0 out is selected sdp_out1: SDP1 out is selected timer_grp0_strb0: Timer group 0, Timer 0 is selected timer_grp1_strb0: Timer group 1, Timer 0 is selected timer_grp2_strb0: Timer group 2, Timer 0 is selected timer_grp3_strb0: Timer group 3, Timer 0 is selected spu0_gioout28: SPU0 GIO[28] is selected spu1_gioout28: SPU1 GIO[28] is selected	spu0_gioout0=0 spu1_gioout0=1 spu0_gioout2=2 spu1_gioout2=3 spu0_gioout4=4 spu1_gioout4=5 spu0_gioout6=6 spu1_gioout6=7 sdp_out0=8 sdp_out1=9 timer_grp0_strb0=10 timer_grp1_strb0=11 timer_grp2_strb0=12 timer_grp3_strb0=13 spu0_gioout28=14 spu1_gioout28=15

25.26.51 rw_spu0_cfg

Address	0xc8
Default	0x00000000
Type	Read/Write
Description	This register controls the source of input buses into SPU0.

Bit(s)	Name	Description	Value
3-2:2	bus1_in	Select source for BUS1. bus0: Data from external BUS0 bus1: Data from external BUS1 pdp_out0: Parallel datapath 0 (Out) pdp_out1: Parallel datapath 1 (Out)	bus0=0 bus1=1 pdp_out0=2 pdp_out1=3
1-0:2	bus0_in	Select source for BUS0. bus0: Data from external BUS0 bus1: Data from external BUS1 pdp_out0: Parallel datapath 0 (Out) pdp_out1: Parallel datapath 1 (Out)	bus0=0 bus1=1 pdp_out0=2 pdp_out1=3

25.26.52 rw_spu1_cfg

Address	0xcc
Default	0x00000000
Type	Read/Write
Description	This register controls the source of input buses into SPU1.

Bit(s)	Name	Description	Value
3-2:2	bus1_in	Select source for BUS1. bus0: Data from external BUS0 bus1: Data from external BUS1 pdp_out0: Parallel datapath 0 (Out) pdp_out1: Parallel datapath 1 (Out)	bus0=0 bus1=1 pdp_out0=2 pdp_out1=3
1-0:2	bus0_in	Select source for BUS0. bus0: Data from external BUS0 bus1: Data from external BUS1 pdp_out0: Parallel datapath 0 (Out) pdp_out1: Parallel datapath 1 (Out)	bus0=0 bus1=1 pdp_out0=2 pdp_out1=3

25.26.53 rw_timer_grp0_cfg

Address	0xd0
Default	0x00000000
Type	Read/Write
Description	This register controls the sources of signals into Timer group 0.

Bit(s)	Name	Description	Value
10	tmr3_dis	Select source for Timer 3 disable. trig0_3: Trigger group 0, Trigger 3, (gio_in[3]) trig4_3: Trigger group 4, Trigger 3, (gio_in[19])	trig0_3=0 trig4_3=1
9	tmr2_dis	Select source for Timer 2 disable. trig0_2: Trigger group 0, Trigger 2, (gio_in[2]) trig4_2: Trigger group 4, Trigger 2, (gio_in[18])	trig0_2=0 trig4_2=1
8	tmr1_dis	Select source for Timer 1 disable. trig0_1: Trigger group 0, Trigger 1, (gio_in[1]) trig4_1: Trigger group 4, Trigger 1, (gio_in[17])	trig0_1=0 trig4_1=1
7	tmr0_dis	Select source for Timer 0 disable. trig0_0: Trigger group 0, Trigger 0, (gio_in[0]) trig4_0: Trigger group 4, Trigger 0, (gio_in[16])	trig0_0=0 trig4_0=1
6	tmr3_en	Select source for Timer 3 enable. trig0_3: Trigger group 0, Trigger 3, (gio_in[3]) trig4_3: Trigger group 4, Trigger 3, (gio_in[19])	trig0_3=0 trig4_3=1
5	tmr2_en	Select source for Timer 2 enable. trig0_2: Trigger group 0, Trigger 2, (gio_in[2]) trig4_2: Trigger group 4, Trigger 2, (gio_in[18])	trig0_2=0 trig4_2=1
4	tmr1_en	Select source for Timer 1 enable. trig0_1: Trigger group 0, Trigger 1, (gio_in[1]) trig4_1: Trigger group 4, Trigger 1, (gio_in[17])	trig0_1=0 trig4_1=1
3	tmr0_en	Select source for Timer 0 enable. trig0_0: Trigger group 0, Trigger 0 (gio_in[0]) trig4_0: Trigger group 4, Trigger 0 (gio_in[16])	trig0_0=0 trig4_0=1
2-0:3	ext_clk	Select source for Timer group 0 ext clock. clk12: Synchronized 12 MHz clock gio1: Synchronized gio_in[1] gio3: Synchronized gio_in[3] gio5: Synchronized gio_in[5] gio7: Synchronized gio_in[7] spu0_gio5: Use SPU0 GIO[5] as ext_clk spu0_gio6: Use SPU0 GIO[6] as ext_clk spu0_gio7: Use SPU0 GIO[7] as ext_clk	clk12=0 gio1=1 gio3=2 gio5=3 gio7=4 spu0_gio5=5 spu0_gio6=6 spu0_gio7=7

25.26.54 rw_timer_grp1_cfg

Address	0xd4
Default	0x00000000
Type	Read/Write
Description	This register controls the sources of signals into Timer group 1.

Bit(s)	Name	Description	Value
10	tmr3_dis	Select source for Timer 3 disable. trig1_3: Trigger group 1, Trigger 3, (gio_in[7]) trig5_3: Trigger group 5, Trigger 3, (gio_in[23])	trig1_3=0 trig5_3=1
9	tmr2_dis	Select source for Timer 2 disable. trig1_2: Trigger group 1, Trigger 2, (gio_in[6]) trig5_2: Trigger group 5, Trigger 2, (gio_in[22])	trig1_2=0 trig5_2=1
8	tmr1_dis	Select source for Timer 1 disable. trig1_1: Trigger group 1, Trigger 1, (gio_in[5]) trig5_1: Trigger group 5, Trigger 1, (gio_in[21])	trig1_1=0 trig5_1=1
7	tmr0_dis	Select source for Timer 0 disable. trig1_0: Trigger group 1, Trigger 0, (gio_in[4]) trig5_0: Trigger group 5, Trigger 0, (gio_in[20])	trig1_0=0 trig5_0=1
6	tmr3_en	Select source for Timer 3 enable. trig1_3: Trigger group 1, Trigger 3, (gio_in[7]) trig5_3: Trigger group 5, Trigger 3, (gio_in[23])	trig1_3=0 trig5_3=1
5	tmr2_en	Select source for Timer 2 enable. trig1_2: Trigger group 1, Trigger 2, (gio_in[6]) trig5_2: Trigger group 5, Trigger 2, (gio_in[22])	trig1_2=0 trig5_2=1
4	tmr1_en	Select source for Timer 1 enable. trig1_1: Trigger group 1, Trigger 1, (gio_in[5]) trig5_1: Trigger group 5, Trigger 1, (gio_in[21])	trig1_1=0 trig5_1=1
3	tmr0_en	Select source for Timer 0 enable. trig1_0: Trigger group 1, Trigger 0, (gio_in[4]) trig5_0: Trigger group 5, Trigger 0, (gio_in[20])	trig1_0=0 trig5_0=1
2-0:3	ext_clk	Select source for Timer group 1 ext clock. clk12: Synchronized 12 MHz clock gio1: Synchronized gio_in[1] gio3: Synchronized gio_in[3] gio5: Synchronized gio_in[5] gio7: Synchronized gio_in[7] spu1_gio5: Use SPU1 GIO[5] as ext_clk spu1_gio6: Use SPU1 GIO[6] as ext_clk spu1_gio7: Use SPU1 GIO[7] as ext_clk	clk12=0 gio1=1 gio3=2 gio5=3 gio7=4 spu1_gio5=5 spu1_gio6=6 spu1_gio7=7

25.26.55 rw_timer_grp2_cfg

Address	0xd8
Default	0x00000000
Type	Read/Write
Description	This register controls the sources of signals into Timer group 2.

Bit(s)	Name	Description	Value
10	tmr3_dis	Select source for Timer 3 disable. trig2_3: Trigger group 2, Trigger 3, (gio_in[11]) trig6_3: Trigger group 6, Trigger 3, (gio_in[27])	trig2_3=0 trig6_3=1
9	tmr2_dis	Select source for Timer 2 disable. trig2_2: Trigger group 2, Trigger 2, (gio_in[10]) trig6_2: Trigger group 6, Trigger 2, (gio_in[26])	trig2_2=0 trig6_2=1
8	tmr1_dis	Select source for Timer 1 disable. trig2_1: Trigger group 2, Trigger 1, (gio_in[9]) trig6_1: Trigger group 6, Trigger 1, (gio_in[25])	trig2_1=0 trig6_1=1
7	tmr0_dis	Select source for Timer 0 disable. trig2_0: Trigger group 2, Trigger 0, (gio_in[8]) trig6_0: Trigger group 6, Trigger 0, (gio_in[24])	trig2_0=0 trig6_0=1
6	tmr3_en	Select source for Timer 3 enable. trig2_3: Trigger group 2, Trigger 3, (gio_in[11]) trig6_3: Trigger group 6, Trigger 3, (gio_in[27])	trig2_3=0 trig6_3=1
5	tmr2_en	Select source for Timer 2 enable. trig2_2: Trigger group 2, Trigger 2, (gio_in[10]) trig6_2: Trigger group 6, Trigger 2, (gio_in[26])	trig2_2=0 trig6_2=1
4	tmr1_en	Select source for Timer 1 enable. trig2_1: Trigger group 2, Trigger 1, (gio_in[9]) trig6_1: Trigger group 6, Trigger 1, (gio_in[25])	trig2_1=0 trig6_1=1
3	tmr0_en	Select source for Timer 0 enable. trig2_0: Trigger group 2, Trigger 0, (gio_in[8]) trig6_0: Trigger group 6, Trigger 0, (gio_in[24])	trig2_0=0 trig6_0=1
2-0:3	ext_clk	Select source for Timer group 2 ext clock. clk12: Synchronized 12 MHz clock gio1: Synchronized gio_in[1] gio3: Synchronized gio_in[3] gio5: Synchronized gio_in[5] gio7: Synchronized gio_in[7] spu0_gio5: Use SPU0 GIO[5] as ext_clk spu0_gio6: Use SPU0 GIO[6] as ext_clk spu0_gio7: Use SPU0 GIO[7] as ext_clk	clk12=0 gio1=1 gio3=2 gio5=3 gio7=4 spu0_gio5=5 spu0_gio6=6 spu0_gio7=7

25.26.56 rw_timer_grp3_cfg

Address	0xdc
Default	0x00000000
Type	Read/Write
Description	This register controls the sources of signals into Timer group 3.

Bit(s)	Name	Description	Value
10	tmr3_dis	Select source for Timer 3 disable. trig3_3: Trigger group 3, Trigger 3, (gio_in[15]) trig7_3: Trigger group 7, Trigger 3, (gio_in[31])	trig3_3=0 trig7_3=1
9	tmr2_dis	Select source for Timer 2 disable. trig3_2: Trigger group 3, Trigger 2, (gio_in[14]) trig7_2: Trigger group 7, Trigger 2, (gio_in[30])	trig3_2=0 trig7_2=1
8	tmr1_dis	Select source for Timer 1 disable. trig3_1: Trigger group 3, Trigger 1, (gio_in[13]) trig7_1: Trigger group 7, Trigger 1, (gio_in[29])	trig3_1=0 trig7_1=1
7	tmr0_dis	Select source for Timer 0 disable. trig3_0: Trigger group 3, Trigger 0, (gio_in[12]) trig7_0: Trigger group 7, Trigger 0, (gio_in[28])	trig3_0=0 trig7_0=1
6	tmr3_en	Select source for Timer 3 enable. trig3_3: Trigger group 3, Trigger 3, (gio_in[15]) trig7_3: Trigger group 7, Trigger 3, (gio_in[31])	trig3_3=0 trig7_3=1
5	tmr2_en	Select source for Timer 2 enable. trig3_2: Trigger group 3, Trigger 2, (gio_in[14]) trig7_2: Trigger group 7, Trigger 2, (gio_in[30])	trig3_2=0 trig7_2=1
4	tmr1_en	Select source for Timer 1 enable. trig3_1: Trigger group 3, Trigger 1, (gio_in[13]) trig7_1: Trigger group 7, Trigger 1, (gio_in[29])	trig3_1=0 trig7_1=1
3	tmr0_en	Select source for Timer 0 enable. trig3_0: Trigger group 3, Trigger 0, (gio_in[12]) trig7_0: Trigger group 7, Trigger 0, (gio_in[28])	trig3_0=0 trig7_0=1
2-0:3	ext_clk	Select source for Timer group 3 ext clock. clk12: Synchronized 12 MHz clock gio1: Synchronized gio_in[1] gio3: Synchronized gio_in[3] gio5: Synchronized gio_in[5] gio7: Synchronized gio_in[7] spu1_gio5: Use SPU1 GIO[5] as ext_clk spu1_gio6: Use SPU1 GIO[6] as ext_clk spu1_gio7: Use SPU1 GIO[7] as ext_clk	clk12=0 gio1=1 gio3=2 gio5=3 gio7=4 spu1_gio5=5 spu1_gio6=6 spu1_gio7=7

25.26.57 rw_trigger_grps_cfg

Address	0xe0
Default	0x00000000
Type	Read/Write
Description	This register configures the Triggers.

Bit(s)	Name	Description	Value
15	grp7_en	Selects source of group 7 enable signals. timer_grp3: Timer group 3, timer 3-0 timer_grp3_rot: Timer group 3, timer 0-1	timer_grp3=0 timer_grp3_rot=1
14	grp7_dis	Selects source of group 7 disable signals. timer_grp3: Timer group 3, timer 3-0 timer_grp3_rot: Timer group 3, timer 0-1	timer_grp3=0 timer_grp3_rot=1
13	grp6_en	Selects source of group 6 enable signals. timer_grp2: Timer group 2, timer 3-0 timer_grp2_rot: Timer group 2, timer 0-1	timer_grp2=0 timer_grp2_rot=1
12	grp6_dis	Selects source of group 6 disable signals. timer_grp2: Timer group 2, timer 3-0 timer_grp2_rot: Timer group 2, timer 0-1	timer_grp2=0 timer_grp2_rot=1
11	grp5_en	Selects source of group 5 enable signals. timer_grp1: Timer group 1, timer 3-0 timer_grp1_rot: Timer group 1, timer 0-1	timer_grp1=0 timer_grp1_rot=1
10	grp5_dis	Selects source of group 5 disable signals. timer_grp1: Timer group 1, timer 3-0 timer_grp1_rot: Timer group 1, timer 0-1	timer_grp1=0 timer_grp1_rot=1
9	grp4_en	Selects source of group 4 enable signals. timer_grp0: Timer group 0, timer 3-0 timer_grp0_rot: Timer group 0, timer 0-1	timer_grp0=0 timer_grp0_rot=1
8	grp4_dis	Selects source of group 4 disable signals. timer_grp0: Timer group 0, timer 3-0 timer_grp0_rot: Timer group 0, timer 0-1	timer_grp0=0 timer_grp0_rot=1
7	grp3_en	Selects source of group 3 enable signals. timer_grp3: Timer group 3, timer 3-0 timer_grp3_rot: Timer group 3, timer 0-1	timer_grp3=0 timer_grp3_rot=1
6	grp3_dis	Selects source of group 3 disable signals. timer_grp3: Timer group 3, timer 3-0 timer_grp3_rot: Timer group 3, timer 0-1	timer_grp3=0 timer_grp3_rot=1
5	grp2_en	Selects source of group 2 enable signals. timer_grp2: Timer group 2, timer 3-0 timer_grp2_rot: Timer group 2, timer 0-1	timer_grp2=0 timer_grp2_rot=1
4	grp2_dis	Selects source of group 2 disable signals. timer_grp2: Timer group 2, timer 3-0 timer_grp2_rot: Timer group 2, timer 0-1	timer_grp2=0 timer_grp2_rot=1

3	grp1_en	Selects source of group 1 enable signals. timer_grp1: Timer group 1, timer 3-0 timer_grp1_rot: Timer group 1, timer 0-1	timer_grp1=0 timer_grp1_rot=1
2	grp1_dis	Selects source of group 1 disable signals. timer_grp1: Timer group 1, timer 3-0 timer_grp1_rot: Timer group 1, timer 0-1	timer_grp1=0 timer_grp1_rot=1
1	grp0_en	Selects source of group 0 enable signals. timer_grp0: Timer group 0, timer 3-0 timer_grp0_rot: Timer group 0, timer 0-1	timer_grp0=0 timer_grp0_rot=1
0	grp0_dis	Selects source of group 0 disable signals. timer_grp0: Timer group 0, timer 3-0 timer_grp0_rot: Timer group 0, timer 0-1	timer_grp0=0 timer_grp0_rot=1

25.26.58 rw_pdp0_cfg

Address	0xe4
Default	0x00000000
Type	Read/Write
Description	This register configures the parallel datapath (DMC, FIFO, CRC).

Bit(s)	Name	Description	Value
18	out_src	Selects source for PDP out 0. dmc0: Data from DMC 0 dmc1: Data from DMC 1	dmc0=0 dmc1=1
17-14:4	in_strb	Select source for PDP in 0 strobe signal. none: No strobe source strb_timer_grp0_tmr0: Timer group 0, timer 0 strb_timer_grp2_tmr1: Timer group 2, timer 1 strb_timer_grp2_tmr0: Timer group 2, timer 0 gio0: Synchronized gio in 0 gio2: Synchronized gio in 2 gio4: Synchronized gio in 4 gio6: Synchronized gio in 6 spu0_gio_out0: Use SPU0 GIO[0] as strobe spu0_gio_out1: Use SPU0 GIO[1] as strobe spu0_gio_out2: Use SPU0 GIO[2] as strobe spu0_gio_out3: Use SPU0 GIO[3] as strobe spu0_gio_out4: Use SPU0 GIO[4] as strobe spu0_gio_out5: Use SPU0 GIO[5] as strobe spu0_gio_out6: Use SPU0 GIO[6] as strobe spu0_gio_out7: Use SPU0 GIO[7] as strobe	none=0 strb_timer_grp0_tmr0=1 strb_timer_grp2_tmr1=2 strb_timer_grp2_tmr0=3 gio0=4 gio2=5 gio4=6 gio6=7 spu0_gio_out0=8 spu0_gio_out1=9 spu0_gio_out2=10 spu0_gio_out3=11 spu0_gio_out4=12 spu0_gio_out5=13 spu0_gio_out6=14 spu0_gio_out7=15
13-12:2	in_last	Select source for PDP in 0 last signal. none: No last source last_timer_grp0_tmr2: Timer group 0, timer 2 last_timer_grp2_tmr2: Timer group 2, timer 2 last_timer_grp2_tmr3: Timer group 2, timer 3	none=0 last_timer_grp0_tmr2=1 last_timer_grp2_tmr2=2 last_timer_grp2_tmr3=3
11-9:3	in_size	Select size for PDP in 0. none: Bus size is 0 size8: Bus size is 8 size16: Bus size is 16 size24: Bus size is 24 size32: Bus size is 32	none=0 size8=1 size16=2 size24=3 size32=4

8-6:3	in_src	Selects source for PDP in 0. bus0: Data from external bus0 bus1: Data from external bus1 bus0_rot8: Data from external bus0 rotated 8 bits ([7:0], [31:8]) bus1_rot8: Data from external bus1 rotated 8 bits ([7:0], [31:8]) bus0_rot16: Data from external bus0 rotated 16 bits ([15:0], [31:16]) bus1_rot16: Data from external bus1 rotated 16 bits ([15:0], [31:16]) bus0_rot24: Data from external bus0 rotated 24 bits ([23:0], [31:24]) bus1_rot24: Data from external bus1 rotated 24 bits ([23:0], [31:24])	bus0=0 bus1=1 bus0_rot8=2 bus1_rot8=3 bus0_rot16=4 bus1_rot16=5 bus0_rot24=6 bus1_rot24=7
5-1:5	out_strb	Selects source for PDP out 0 strobe signal. none: No strobe source strb_timer_grp0_tmr0: Timer group 0, timer 0 strb_timer_grp0_tmr1: Timer group 0, timer 1 strb_timer_grp2_tmr0: Timer group 2, timer 0 gio0: Synchronized gio in 0 gio2: Synchronized gio in 2 gio4: Synchronized gio in 4 gio6: Synchronized gio in 6 spu0_gio_out0: Use SPU0 GIO[0] as strobe spu0_gio_out1: Use SPU0 GIO[1] as strobe spu0_gio_out2: Use SPU0 GIO[2] as strobe spu0_gio_out3: Use SPU0 GIO[3] as strobe spu0_gio_out4: Use SPU0 GIO[4] as strobe spu0_gio_out5: Use SPU0 GIO[5] as strobe spu0_gio_out6: Use SPU0 GIO[6] as strobe spu0_gio_out7: Use SPU0 GIO[7] as strobe gated_clk0: Use gated clock 0 from SAP out as strobe gated_clk1: Use gated clock 1 from SAP out as strobe gated_clk2: Use gated clock 2 from SAP out as strobe gated_clk3: Use gated clock 3 from SAP out as strobe	none=0 strb_timer_grp0_tmr0=1 strb_timer_grp0_tmr1=2 strb_timer_grp2_tmr0=3 gio0=4 gio2=5 gio4=6 gio6=7 spu0_gio_out0=8 spu0_gio_out1=9 spu0_gio_out2=10 spu0_gio_out3=11 spu0_gio_out4=12 spu0_gio_out5=13 spu0_gio_out6=14 spu0_gio_out7=15 gated_clk0=16 gated_clk1=17 gated_clk2=18 gated_clk3=19
0	dmc0_usr	Selects user of DMC OUT 0. par0: Parallel datapath 0 par1: Parallel datapath 1	par0=0 par1=1

25.26.59 rw_pdp1_cfg

Address	0xe8
Default	0x00000000
Type	Read/Write
Description	This register configures the parallel datapath (DMC, FIFO, CRC).

Bit(s)	Name	Description	Value
18	out_src	Selects source for PDP out 1. dmc0: Data from DMC 0 dmc1: Data from DMC 1	dmc0=0 dmc1=1
17-14:4	in_strb	Select source for PDP in 1 strobe signal. none: No strobe source strb_timer_grp1_tmr0: Timer group 1, timer 0 strb_timer_grp3_tmr1: Timer group 3, timer 1 strb_timer_grp3_tmr0: Timer group 3, timer 0 gio0: Synchronized gio in 0 gio2: Synchronized gio in 2 gio4: Synchronized gio in 4 gio6: Synchronized gio in 6 spu1_gio_out0: Use SPU1 GIO[0] as strobe spu1_gio_out1: Use SPU1 GIO[1] as strobe spu1_gio_out2: Use SPU1 GIO[2] as strobe spu1_gio_out3: Use SPU1 GIO[3] as strobe spu1_gio_out4: Use SPU1 GIO[4] as strobe spu1_gio_out5: Use SPU1 GIO[5] as strobe spu1_gio_out6: Use SPU1 GIO[6] as strobe spu1_gio_out7: Use SPU1 GIO[7] as strobe	none=0 strb_timer_grp1_tmr0=1 strb_timer_grp3_tmr1=2 strb_timer_grp3_tmr0=3 gio0=4 gio2=5 gio4=6 gio6=7 spu1_gio_out0=8 spu1_gio_out1=9 spu1_gio_out2=10 spu1_gio_out3=11 spu1_gio_out4=12 spu1_gio_out5=13 spu1_gio_out6=14 spu1_gio_out7=15
13-12:2	in_last	Select source for PDP in 1 last signal. none: No last source last_timer_grp1_tmr2: Timer group 1, timer 2 last_timer_grp3_tmr2: Timer group 3, timer 2 last_timer_grp3_tmr3: Timer group 3, timer 3	none=0 last_timer_grp1_tmr2=1 last_timer_grp3_tmr2=2 last_timer_grp3_tmr3=3
11-9:3	in_size	Select size for PDP in 1. none: Bus size is 0 size8: Bus size is 8 size16: Bus size is 16 size24: Bus size is 24 size32: Bus size is 32	none=0 size8=1 size16=2 size24=3 size32=4

8-6:3	in_src	Selects source for PDP in 1. bus0: Data from external bus0 bus1: Data from external bus1 bus0_rot8: Data from external bus0 rotated 8 bits ([7:0], [31:8]) bus1_rot8: Data from external bus1 rotated 8 bits ([7:0], [31:8]) bus0_rot16: Data from external bus0 rotated 16 bits ([15:0], [31:16]) bus1_rot16: Data from external bus1 rotated 16 bits ([15:0], [31:16]) bus0_rot24: Data from external bus0 rotated 24 bits ([23:0], [31:24]) bus1_rot24: Data from external bus1 rotated 24 bits ([23:0], [31:24])	bus0=0 bus1=1 bus0_rot8=2 bus1_rot8=3 bus0_rot16=4 bus1_rot16=5 bus0_rot24=6 bus1_rot24=7
5-1:5	out_strb	Selects source for PDP out 1 strobe signal. none: No strobe source strb_timer_grp1_tmr0: Timer group 1, timer 0 strb_timer_grp1_tmr1: Timer group 1, timer 1 strb_timer_grp3_tmr0: Timer group 3, timer 0 gio0: Synchronized gio in 0 gio2: Synchronized gio in 2 gio4: Synchronized gio in 4 gio6: Synchronized gio in 6 spu1_gio_out0: Use SPU1 GIO[0] as strobe spu1_gio_out1: Use SPU1 GIO[1] as strobe spu1_gio_out2: Use SPU1 GIO[2] as strobe spu1_gio_out3: Use SPU1 GIO[3] as strobe spu1_gio_out4: Use SPU1 GIO[4] as strobe spu1_gio_out5: Use SPU1 GIO[5] as strobe spu1_gio_out6: Use SPU1 GIO[6] as strobe spu1_gio_out7: Use SPU1 GIO[7] as strobe gated_clk0: Use gated clock 0 from SAP out as strobe gated_clk1: Use gated clock 1 from SAP out as strobe gated_clk2: Use gated clock 2 from SAP out as strobe gated_clk3: Use gated clock 3 from SAP out as strobe	none=0 strb_timer_grp1_tmr0=1 strb_timer_grp1_tmr1=2 strb_timer_grp3_tmr0=3 gio0=4 gio2=5 gio4=6 gio6=7 spu1_gio_out0=8 spu1_gio_out1=9 spu1_gio_out2=10 spu1_gio_out3=11 spu1_gio_out4=12 spu1_gio_out5=13 spu1_gio_out6=14 spu1_gio_out7=15 gated_clk0=16 gated_clk1=17 gated_clk2=18 gated_clk3=19
0	dmc1_usr	Selects user of DMC OUT 1. par0: Parallel datapath 0 par1: Parallel datapath 1	par0=0 par1=1

25.26.60 rw_sdp_cfg

Address	0xec
Default	0x00000000
Type	Read/Write
Description	This register configures the serial datapath (Serial CRC In/Out) .

Bit(s)	Name	Description	Value
21-19:3	sdp_in1_strb	Select source for SDP in 1 strobe signal. none: No strobe source strb_timer_grp1_tmr0: Timer group 1, timer 0 strb_timer_grp3_tmr1: Timer group 3, timer 1 strb_timer_grp3_tmr0: Timer group 3, timer 0 gio0: Synchronized gio in 0 gio2: Synchronized gio in 2 gio4: Synchronized gio in 4 gio6: Synchronized gio in 6	none=0 strb_timer_grp1_tmr0=1 strb_timer_grp3_tmr1=2 strb_timer_grp3_tmr0=3 gio0=4 gio2=5 gio4=6 gio6=7
18-17:2	sdp_in1_last	Select source for SDP in 1 last signal. none: No last source last_timer_grp1_tmr2: Timer group 1, timer 2 last_timer_grp3_tmr2: Timer group 3, timer 2 last_timer_grp3_tmr3: Timer group 3, timer 3	none=0 last_timer_grp1_tmr2=1 last_timer_grp3_tmr2=2 last_timer_grp3_tmr3=3
16-14:3	sdp_in1_data	Select source for SDP in 1 data. gio_in0: Synchronized gio in 0 gio_in1: Synchronized gio in 1 gio_in10: Synchronized gio in 10 gio_in11: Synchronized gio in 11 gio_in14: Synchronized gio in 14 gio_in15: Synchronized gio in 15 gio_in28: Synchronized gio in 28 gio_in29: Synchronized gio in 29	gio_in0=0 gio_in1=1 gio_in10=2 gio_in11=3 gio_in14=4 gio_in15=5 gio_in28=6 gio_in29=7

13-11:3	sdp_in0_strb	Select source for SDP in 0 strobe signal. none: No strobe source strb_timer_grp0_tmr0: Timer group 0, timer 0 strb_timer_grp2_tmr1: Timer group 2, timer 1 strb_timer_grp2_tmr0: Timer group 2, timer 0 gio0: Synchronized gio in 0 gio2: Synchronized gio in 2 gio4: Synchronized gio in 4 gio6: Synchronized gio in 6	none=0 strb_timer_grp0_tmr0=1 strb_timer_grp2_tmr1=2 strb_timer_grp2_tmr0=3 gio0=4 gio2=5 gio4=6 gio6=7
10-9:2	sdp_in0_last	Select source for SDP in 0 last signal. none: No last source last_timer_grp0_tmr2: Timer group 0, timer 2 last_timer_grp2_tmr2: Timer group 2, timer 2 last_timer_grp2_tmr3: Timer group 2, timer 3	none=0 last_timer_grp0_tmr2=1 last_timer_grp2_tmr2=2 last_timer_grp2_tmr3=3
8-6:3	sdp_in0_data	Select source for SDP in 0 data. gio_in4: Synchronized gio in 4 gio_in5: Synchronized gio in 5 gio_in18: Synchronized gio in 18 gio_in19: Synchronized gio in 19 gio_in20: Synchronized gio in 20 gio_in21: Synchronized gio in 21 gio_in26: Synchronized gio in 26 gio_in27: Synchronized gio in 27	gio_in4=0 gio_in5=1 gio_in18=2 gio_in19=3 gio_in20=4 gio_in21=5 gio_in26=6 gio_in27=7
5-3:3	sdp_out1_strb	Select source for SDP out 1 strobe signal. none: No strobe source strb_timer_grp1_tmr0: Timer group 1, timer 0 strb_timer_grp1_tmr1: Timer group 1, timer 1 strb_timer_grp3_tmr0: Timer group 3, timer 0 gio0: Synchronized gio in 0 gio2: Synchronized gio in 2 gio4: Synchronized gio in 4 gio6: Synchronized gio in 6	none=0 strb_timer_grp1_tmr0=1 strb_timer_grp1_tmr1=2 strb_timer_grp3_tmr0=3 gio0=4 gio2=5 gio4=6 gio6=7

2-0:3	sdp_out0_strb	Select source for SDP out 0 strobe signal. none: No strobe source strb_timer_grp0_tmr0: Timer group 0, timer 0 strb_timer_grp0_tmr1: Timer group 0, timer 1 strb_timer_grp2_tmr0: Timer group 2, timer 0 gio0: Synchronized gio in 0 gio2: Synchronized gio in 2 gio4: Synchronized gio in 4 gio6: Synchronized gio in 6	none=0 strb_timer_grp0_tmr0=1 strb_timer_grp0_tmr1=2 strb_timer_grp2_tmr0=3 gio0=4 gio2=5 gio4=6 gio6=7

25.27 iop_sw_cpu

Instance	Base Address
iop_sw_cpu	0xb0021200

25.27.1 rw_mc_ctrl

Address	0x0
Default	
Type	Read/Write
Description	Control register for the MC. A write to this register requests ownership of MC.

Bit(s)	Name	Description	Value
7	wr_spu1_mem	Enable writes to SPU1 memory. no: No writes to SPU1 memory yes: Write to SPU1 memory	no=0 yes=1
6	wr_spu0_mem	Enable writes to SPU0 memory. no: No writes to SPU0 memory yes: Write to SPU0 memory	no=0 yes=1
5-3:3	size	Size (in bytes) of data to read or write. MC can read/write at most four bytes.	
2-1:2	cmd	Command to perform. copy: Copy data from SMIF and write to I/O memory reg_copy: Data from mc_data is written to I/O memory rd: Read from SMIF to r_mc_data wr: Write rw_mc_data to SMIF	copy=0 reg_copy=1 rd=2 wr=3
0	keep_owner	Enables ownership to be kept after finished operation. no: Ownerships is automatically released yes: Ownership is kept	no=0 yes=1

25.27.2 rw_mc_data

Address	0x4
Default	
Type	Read/Write
Description	Data for write to system memory. Depending on size in rw_mc_ctrl some parts of value can be don't care. If rw_mc_ctrl.cmd is set to reg_copy the write to this register performs the write to I/O memory. If rw_mc_ctrl.keep_owner is not set the ownership of MC will be released.

Bit(s)	Name	Description	Value
31-0:32	val	Data value to be written to system memory.	

25.27.3 rw_mc_addr

Address	0x8
Default	
Type	Read/Write
Description	The system memory address to perform operation on. rw_mc_addr states the address which will be used by the selected command in rw_mc_ctrl.cmd. A write to this register starts the memory operation. If operation is wr or copy and rw_mc_ctrl.keep_owner isn't set the ownership will be lost when operation is done.

25.27.4 rs_mc_data/r_mc_data

Address	0xc/0x10
Default	
Type	Read with side effects/Read
Description	Data read from system memory at address rw_mc_addr. When reading rs_mc_data ownership of MC is released.

25.27.5 r_mc_stat

Address	0x14
Default	
Type	Read
Description	Status of the MC.

Bit(s)	Name	Description	Value
7	owned_by_spu1	MC is owned by SPU1. no: Not owned by SPU1 yes: Owned by SPU1	no=0 yes=1
6	owned_by_spu0	MC is owned by SPU0. no: Not owned by SPU0 yes: Owned by SPU0	no=0 yes=1
5	owned_by_mpu	MC is owned by MPU. no: Not owned by MPU yes: Owned by MPU	no=0 yes=1
4	owned_by_cpu	MC is owned by CPU. no: Not owned by CPU yes: Owned by CPU	no=0 yes=1
3	busy_spu1	MC is busy performing command for SPU1. no: Not busy yes: Busy	no=0 yes=1
2	busy_spu0	MC is busy performing command for SPU0. no: Not busy yes: Busy	no=0 yes=1
1	busy_mpu	MC is busy performing command for MPU. no: Not busy yes: Busy	no=0 yes=1
0	busy_cpu	MC is busy performing command for CPU. no: Not busy yes: Busy	no=0 yes=1

25.27.6 rw_bus0_clr_mask

Address	0x18
Default	0x00000000
Type	Read/Write
Description	Clear bits in BUS0.

Bit(s)	Name	Description	Value
31-24:8	byte3	Used to clear bits in BUS0[31:24].	
23-16:8	byte2	Used to clear bits in BUS0[23:16].	
15-8:8	byte1	Used to clear bits in BUS0[15:8].	
7-0:8	byte0	Used to clear bits in BUS0[7:0].	

25.27.7 rw_bus0_set_mask

Address	0x1c
Default	0x00000000
Type	Read/Write
Description	Set bits in BUS0.

Bit(s)	Name	Description	Value
31-24:8	byte3	Used to set bits in BUS0[31:24].	
23-16:8	byte2	Used to set bits in BUS0[23:16].	
15-8:8	byte1	Used to set bits in BUS0[15:8].	
7-0:8	byte0	Used to set bits in BUS0[7:0].	

25.27.8 rw_bus0_oe_clr_mask

Address	0x20
Default	0x00000000
Type	Read/Write
Description	Clear OE signals for BUS0.

Bit(s)	Name	Description	Value
3	byte3	Used to clear OE for BUS0[31:24].	
2	byte2	Used to clear OE for BUS0[23:16].	
1	byte1	Used to clear OE for BUS0[15:8].	
0	byte0	Used to clear OE for BUS0[7:0].	

25.27.9 rw_bus0_oe_set_mask

Address	0x24
Default	0x00000000
Type	Read/Write
Description	Set OE signals for BUS0.

Bit(s)	Name	Description	Value
3	byte3	Used to set OE for BUS0[31:24].	
2	byte2	Used to set OE for BUS0[23:16].	
1	byte1	Used to set OE for BUS0[15:8].	
0	byte0	Used to set OE for BUS0[7:0].	

25.27.10 r_bus0_in

Address	0x28
Default	
Type	Read
Description	Read register for BUS0.

25.27.11 rw_bus1_clr_mask

Address	0x2c
Default	0x00000000
Type	Read/Write
Description	Clear bits in BUS1.

Bit(s)	Name	Description	Value
31-24:8	byte3	Used to clear bits in BUS1[31:24].	
23-16:8	byte2	Used to clear bits in BUS1[23:16].	
15-8:8	byte1	Used to clear bits in BUS1[15:8].	
7-0:8	byte0	Used to clear bits in BUS1[7:0].	

25.27.12 rw_bus1_set_mask

Address	0x30
Default	0x00000000
Type	Read/Write
Description	Set bits in BUS1.

Bit(s)	Name	Description	Value
31-24:8	byte3	Used to set bits in BUS1[31:24].	
23-16:8	byte2	Used to set bits in BUS1[23:16].	
15-8:8	byte1	Used to set bits in BUS1[15:8].	
7-0:8	byte0	Used to set bits in BUS1[7:0].	

25.27.13 rw_bus1_oe_clr_mask

Address	0x34
Default	0x00000000
Type	Read/Write
Description	Clear OE signals for BUS1.

Bit(s)	Name	Description	Value
3	byte3	Used to clear OE for BUS1[31:24].	
2	byte2	Used to clear OE for BUS1[23:16].	
1	byte1	Used to clear OE for BUS1[15:8].	
0	byte0	Used to clear OE for BUS1[7:0].	

25.27.14 rw_bus1_oe_set_mask

Address	0x38
Default	0x00000000
Type	Read/Write
Description	Set OE signals for BUS1.

Bit(s)	Name	Description	Value
3	byte3	Used to set OE for BUS1[31:24].	
2	byte2	Used to set OE for BUS1[23:16].	
1	byte1	Used to set OE for BUS1[15:8].	
0	byte0	Used to set OE for BUS1[7:0].	

25.27.15 r_bus1_in

Address	0x3c
Default	
Type	Read
Description	Read register for BUS1.

25.27.16 rw_gio_clr_mask

Address	0x40
Default	0x00000000
Type	Read/Write
Description	Clear bits in GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Used to clear bits in GIO.	

25.27.17 rw_gio_set_mask

Address	0x44
Default	0x00000000
Type	Read/Write
Description	Set bits in GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Used to set bits in GIO.	

25.27.18 rw_gio_oe_clr_mask

Address	0x48
Default	0x00000000
Type	Read/Write
Description	Clear OE signals for GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Used to clear OE for GIO.	

25.27.19 rw_gio_oe_set_mask

Address	0x4c
Default	0x00000000
Type	Read/Write
Description	Set OE signals for GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Used to set OE for GIO.	

25.27.20 r_gio_in

Address	0x50
Default	
Type	Read
Description	Read register for GIO.

25.27.21 rw_intr0_mask

Address	0x54
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	spu1_15	Enable/disable spu1_15 interrupt. Interrupt from SPU1 software, bit 15. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
30	spu1_14	Enable/disable spu1_14 interrupt. Interrupt from SPU1 software, bit 14. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
29	spu1_13	Enable/disable spu1_13 interrupt. Interrupt from SPU1 software, bit 13. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
28	spu1_12	Enable/disable spu1_12 interrupt. Interrupt from SPU1 software, bit 12. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
27	spu1_11	Enable/disable spu1_11 interrupt. Interrupt from SPU1 software, bit 11. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
26	spu1_10	Enable/disable spu1_10 interrupt. Interrupt from SPU1 software, bit 10. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
25	spu1_9	Enable/disable spu1_9 interrupt. Interrupt from SPU1 software, bit 9. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
24	spu1_8	Enable/disable spu1_8 interrupt. Interrupt from SPU1 software, bit 8. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
23	spu0_7	Enable/disable spu0_7 interrupt. Interrupt from SPU0 software, bit 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

22	spu0_6	Enable/disable spu0_6 interrupt. Interrupt from SPU0 software, bit 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
21	spu0_5	Enable/disable spu0_5 interrupt. Interrupt from SPU0 software, bit 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
20	spu0_4	Enable/disable spu0_4 interrupt. Interrupt from SPU0 software, bit 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
19	spu0_3	Enable/disable spu0_3 interrupt. Interrupt from SPU0 software, bit 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
18	spu0_2	Enable/disable spu0_2 interrupt. Interrupt from SPU0 software, bit 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
17	spu0_1	Enable/disable spu0_1 interrupt. Interrupt from SPU0 software, bit 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
16	spu0_0	Enable/disable spu0_0 interrupt. Interrupt from SPU0 software, bit 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
15	mpu_15	Enable/disable mpu_15 interrupt. Interrupt from MPU software, bit 15. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
14	mpu_14	Enable/disable mpu_14 interrupt. Interrupt from MPU software, bit 14. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
13	mpu_13	Enable/disable mpu_13 interrupt. Interrupt from MPU software, bit 13. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
12	mpu_12	Enable/disable mpu_12 interrupt. Interrupt from MPU software, bit 12. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

11	mpu_11	Enable/disable mpu_11 interrupt. Interrupt from MPU software, bit 11. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
10	mpu_10	Enable/disable mpu_10 interrupt. Interrupt from MPU software, bit 10. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
9	mpu_9	Enable/disable mpu_9 interrupt. Interrupt from MPU software, bit 9. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
8	mpu_8	Enable/disable mpu_8 interrupt. Interrupt from MPU software, bit 8. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
7	mpu_7	Enable/disable mpu_7 interrupt. Interrupt from MPU software, bit 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
6	mpu_6	Enable/disable mpu_6 interrupt. Interrupt from MPU software, bit 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
5	mpu_5	Enable/disable mpu_5 interrupt. Interrupt from MPU software, bit 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	mpu_4	Enable/disable mpu_4 interrupt. Interrupt from MPU software, bit 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	mpu_3	Enable/disable mpu_3 interrupt. Interrupt from MPU software, bit 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	mpu_2	Enable/disable mpu_2 interrupt. Interrupt from MPU software, bit 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	mpu_1	Enable/disable mpu_1 interrupt. Interrupt from MPU software, bit 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

0	mpu_0	Enable/disable mpu_0 interrupt. Interrupt from MPU software, bit 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.27.22 rw_ack_intr0

Address	0x58
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	spu1_15	Acknowledge spu1_15 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
30	spu1_14	Acknowledge spu1_14 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
29	spu1_13	Acknowledge spu1_13 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
28	spu1_12	Acknowledge spu1_12 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
27	spu1_11	Acknowledge spu1_11 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
26	spu1_10	Acknowledge spu1_10 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
25	spu1_9	Acknowledge spu1_9 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
24	spu1_8	Acknowledge spu1_8 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
23	spu0_7	Acknowledge spu0_7 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
22	spu0_6	Acknowledge spu0_6 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
21	spu0_5	Acknowledge spu0_5 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
20	spu0_4	Acknowledge spu0_4 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

19	spu0_3	Acknowledge spu0_3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
18	spu0_2	Acknowledge spu0_2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
17	spu0_1	Acknowledge spu0_1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
16	spu0_0	Acknowledge spu0_0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
15	mpu_15	Acknowledge mpu_15 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
14	mpu_14	Acknowledge mpu_14 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
13	mpu_13	Acknowledge mpu_13 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
12	mpu_12	Acknowledge mpu_12 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
11	mpu_11	Acknowledge mpu_11 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
10	mpu_10	Acknowledge mpu_10 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
9	mpu_9	Acknowledge mpu_9 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
8	mpu_8	Acknowledge mpu_8 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
7	mpu_7	Acknowledge mpu_7 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
6	mpu_6	Acknowledge mpu_6 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
5	mpu_5	Acknowledge mpu_5 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

4	mpu_4	Acknowledge mpu_4 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
3	mpu_3	Acknowledge mpu_3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	mpu_2	Acknowledge mpu_2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	mpu_1	Acknowledge mpu_1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	mpu_0	Acknowledge mpu_0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.27.23 r_intr0

Address	0x5c
Default	
Type	Read
Description	Unmasked interrupts. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	spu1_15	Interrupt spu1_15 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	spu1_14	Interrupt spu1_14 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	spu1_13	Interrupt spu1_13 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	spu1_12	Interrupt spu1_12 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	spu1_11	Interrupt spu1_11 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	spu1_10	Interrupt spu1_10 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	spu1_9	Interrupt spu1_9 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	spu1_8	Interrupt spu1_8 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	spu0_7	Interrupt spu0_7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	spu0_6	Interrupt spu0_6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	spu0_5	Interrupt spu0_5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	spu0_4	Interrupt spu0_4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	spu0_3	Interrupt spu0_3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	spu0_2	Interrupt spu0_2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	spu0_1	Interrupt spu0_1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	spu0_0	Interrupt spu0_0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	mpu_15	Interrupt mpu_15 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	mpu_14	Interrupt mpu_14 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	mpu_13	Interrupt mpu_13 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	mpu_12	Interrupt mpu_12 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	mpu_11	Interrupt mpu_11 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	mpu_10	Interrupt mpu_10 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	mpu_9	Interrupt mpu_9 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	mpu_8	Interrupt mpu_8 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	mpu_7	Interrupt mpu_7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	mpu_6	Interrupt mpu_6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	mpu_5	Interrupt mpu_5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	mpu_4	Interrupt mpu_4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	mpu_3	Interrupt mpu_3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	mpu_2	Interrupt mpu_2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	mpu_1	Interrupt mpu_1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	mpu_0	Interrupt mpu_0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.27.24 r_masked_intr0

Address	0x60
Default	
Type	Read
Description	Masked interrupts. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	spu1_15	Interrupt spu1_15 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	spu1_14	Interrupt spu1_14 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	spu1_13	Interrupt spu1_13 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	spu1_12	Interrupt spu1_12 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	spu1_11	Interrupt spu1_11 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	spu1_10	Interrupt spu1_10 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	spu1_9	Interrupt spu1_9 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	spu1_8	Interrupt spu1_8 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	spu0_7	Interrupt spu0_7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	spu0_6	Interrupt spu0_6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	spu0_5	Interrupt spu0_5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	spu0_4	Interrupt spu0_4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	spu0_3	Interrupt spu0_3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	spu0_2	Interrupt spu0_2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	spu0_1	Interrupt spu0_1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	spu0_0	Interrupt spu0_0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	mpu_15	Interrupt mpu_15 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	mpu_14	Interrupt mpu_14 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	mpu_13	Interrupt mpu_13 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	mpu_12	Interrupt mpu_12 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	mpu_11	Interrupt mpu_11 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	mpu_10	Interrupt mpu_10 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	mpu_9	Interrupt mpu_9 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	mpu_8	Interrupt mpu_8 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	mpu_7	Interrupt mpu_7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	mpu_6	Interrupt mpu_6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	mpu_5	Interrupt mpu_5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	mpu_4	Interrupt mpu_4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	mpu_3	Interrupt mpu_3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	mpu_2	Interrupt mpu_2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	mpu_1	Interrupt mpu_1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	mpu_0	Interrupt mpu_0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.27.25 rw_intr1_mask

Address	0x64
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	spu1_7	Enable/disable spu1_7 interrupt. Interrupt from SPU1 software, bit 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
30	spu1_6	Enable/disable spu1_6 interrupt. Interrupt from SPU1 software, bit 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
29	spu1_5	Enable/disable spu1_5 interrupt. Interrupt from SPU1 software, bit 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
28	spu1_4	Enable/disable spu1_4 interrupt. Interrupt from SPU1 software, bit 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
27	spu1_3	Enable/disable spu1_3 interrupt. Interrupt from SPU1 software, bit 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
26	spu1_2	Enable/disable spu1_2 interrupt. Interrupt from SPU1 software, bit 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
25	spu1_1	Enable/disable spu1_1 interrupt. Interrupt from SPU1 software, bit 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
24	spu1_0	Enable/disable spu1_0 interrupt. Interrupt from SPU1 software, bit 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
23	spu0_15	Enable/disable spu0_15 interrupt. Interrupt from SPU0 software, bit 15. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

22	spu0_14	Enable/disable spu0_14 interrupt. Interrupt from SPU0 software, bit 14. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
21	spu0_13	Enable/disable spu0_13 interrupt. Interrupt from SPU0 software, bit 13. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
20	spu0_12	Enable/disable spu0_12 interrupt. Interrupt from SPU0 software, bit 12. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
19	spu0_11	Enable/disable spu0_11 interrupt. Interrupt from SPU0 software, bit 11. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
18	spu0_10	Enable/disable spu0_10 interrupt. Interrupt from SPU0 software, bit 10. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
17	spu0_9	Enable/disable spu0_9 interrupt. Interrupt from SPU0 software, bit 9. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
16	spu0_8	Enable/disable spu0_8 interrupt. Interrupt from SPU0 software, bit 8. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
15	mpu_31	Enable/disable mpu_31 interrupt. Interrupt from MPU software, bit 31. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
14	mpu_30	Enable/disable mpu_30 interrupt. Interrupt from MPU software, bit 30. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
13	mpu_29	Enable/disable mpu_29 interrupt. Interrupt from MPU software, bit 29. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
12	mpu_28	Enable/disable mpu_28 interrupt. Interrupt from MPU software, bit 28. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

11	mpu_27	Enable/disable mpu_27 interrupt. Interrupt from MPU software, bit 27. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
10	mpu_26	Enable/disable mpu_26 interrupt. Interrupt from MPU software, bit 26. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
9	mpu_25	Enable/disable mpu_25 interrupt. Interrupt from MPU software, bit 25. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
8	mpu_24	Enable/disable mpu_24 interrupt. Interrupt from MPU software, bit 24. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
7	mpu_23	Enable/disable mpu_23 interrupt. Interrupt from MPU software, bit 23. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
6	mpu_22	Enable/disable mpu_22 interrupt. Interrupt from MPU software, bit 22. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
5	mpu_21	Enable/disable mpu_21 interrupt. Interrupt from MPU software, bit 21. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	mpu_20	Enable/disable mpu_20 interrupt. Interrupt from MPU software, bit 20. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	mpu_19	Enable/disable mpu_19 interrupt. Interrupt from MPU software, bit 19. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	mpu_18	Enable/disable mpu_18 interrupt. Interrupt from MPU software, bit 18. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	mpu_17	Enable/disable mpu_17 interrupt. Interrupt from MPU software, bit 17. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

0	mpu_16	Enable/disable mpu_16 interrupt. Interrupt from MPU software, bit 16. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.27.26 rw_ack_intr1

Address	0x68
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	spu1_7	Acknowledge spu1_7 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
30	spu1_6	Acknowledge spu1_6 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
29	spu1_5	Acknowledge spu1_5 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
28	spu1_4	Acknowledge spu1_4 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
27	spu1_3	Acknowledge spu1_3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
26	spu1_2	Acknowledge spu1_2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
25	spu1_1	Acknowledge spu1_1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
24	spu1_0	Acknowledge spu1_0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
23	spu0_15	Acknowledge spu0_15 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
22	spu0_14	Acknowledge spu0_14 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
21	spu0_13	Acknowledge spu0_13 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
20	spu0_12	Acknowledge spu0_12 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

19	spu0_11	Acknowledge spu0_11 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
18	spu0_10	Acknowledge spu0_10 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
17	spu0_9	Acknowledge spu0_9 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
16	spu0_8	Acknowledge spu0_8 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
15	mpu_31	Acknowledge mpu_31 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
14	mpu_30	Acknowledge mpu_30 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
13	mpu_29	Acknowledge mpu_29 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
12	mpu_28	Acknowledge mpu_28 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
11	mpu_27	Acknowledge mpu_27 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
10	mpu_26	Acknowledge mpu_26 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
9	mpu_25	Acknowledge mpu_25 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
8	mpu_24	Acknowledge mpu_24 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
7	mpu_23	Acknowledge mpu_23 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
6	mpu_22	Acknowledge mpu_22 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
5	mpu_21	Acknowledge mpu_21 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

4	mpu_20	Acknowledge mpu_20 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
3	mpu_19	Acknowledge mpu_19 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	mpu_18	Acknowledge mpu_18 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	mpu_17	Acknowledge mpu_17 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	mpu_16	Acknowledge mpu_16 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.27.27 r_intr1

Address	0x6c
Default	
Type	Read
Description	Unmasked interrupts. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	spu1_7	Interrupt spu1_7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	spu1_6	Interrupt spu1_6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	spu1_5	Interrupt spu1_5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	spu1_4	Interrupt spu1_4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	spu1_3	Interrupt spu1_3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	spu1_2	Interrupt spu1_2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	spu1_1	Interrupt spu1_1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	spu1_0	Interrupt spu1_0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	spu0_15	Interrupt spu0_15 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	spu0_14	Interrupt spu0_14 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	spu0_13	Interrupt spu0_13 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	spu0_12	Interrupt spu0_12 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	spu0_11	Interrupt spu0_11 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	spu0_10	Interrupt spu0_10 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	spu0_9	Interrupt spu0_9 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	spu0_8	Interrupt spu0_8 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	mpu_31	Interrupt mpu_31 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	mpu_30	Interrupt mpu_30 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	mpu_29	Interrupt mpu_29 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	mpu_28	Interrupt mpu_28 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	mpu_27	Interrupt mpu_27 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	mpu_26	Interrupt mpu_26 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	mpu_25	Interrupt mpu_25 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	mpu_24	Interrupt mpu_24 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	mpu_23	Interrupt mpu_23 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	mpu_22	Interrupt mpu_22 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	mpu_21	Interrupt mpu_21 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	mpu_20	Interrupt mpu_20 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	mpu_19	Interrupt mpu_19 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	mpu_18	Interrupt mpu_18 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	mpu_17	Interrupt mpu_17 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	mpu_16	Interrupt mpu_16 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.27.28 r_masked_intr1

Address	0x70
Default	
Type	Read
Description	Masked interrupts. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	spu1_7	Interrupt spu1_7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	spu1_6	Interrupt spu1_6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	spu1_5	Interrupt spu1_5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	spu1_4	Interrupt spu1_4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	spu1_3	Interrupt spu1_3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	spu1_2	Interrupt spu1_2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	spu1_1	Interrupt spu1_1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	spu1_0	Interrupt spu1_0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	spu0_15	Interrupt spu0_15 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	spu0_14	Interrupt spu0_14 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	spu0_13	Interrupt spu0_13 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	spu0_12	Interrupt spu0_12 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	spu0_11	Interrupt spu0_11 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	spu0_10	Interrupt spu0_10 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	spu0_9	Interrupt spu0_9 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	spu0_8	Interrupt spu0_8 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	mpu_31	Interrupt mpu_31 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	mpu_30	Interrupt mpu_30 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	mpu_29	Interrupt mpu_29 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	mpu_28	Interrupt mpu_28 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	mpu_27	Interrupt mpu_27 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	mpu_26	Interrupt mpu_26 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	mpu_25	Interrupt mpu_25 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	mpu_24	Interrupt mpu_24 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	mpu_23	Interrupt mpu_23 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	mpu_22	Interrupt mpu_22 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	mpu_21	Interrupt mpu_21 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	mpu_20	Interrupt mpu_20 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	mpu_19	Interrupt mpu_19 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	mpu_18	Interrupt mpu_18 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	mpu_17	Interrupt mpu_17 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	mpu_16	Interrupt mpu_16 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.27.29 rw_intr2_mask

Address	0x74
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	timer_grp1	Enable/disable timer_grp1 interrupt. Interrupt from Timer group 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
30	timer_grp0	Enable/disable timer_grp0 interrupt. Interrupt from Timer group 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
29	trigger_grp7	Enable/disable trigger_grp7 interrupt. Interrupt from Trigger group 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
28	trigger_grp6	Enable/disable trigger_grp6 interrupt. Interrupt from Trigger group 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
27	trigger_grp5	Enable/disable trigger_grp5 interrupt. Interrupt from Trigger group 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
26	trigger_grp4	Enable/disable trigger_grp4 interrupt. Interrupt from Trigger group 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
25	trigger_grp3	Enable/disable trigger_grp3 interrupt. Interrupt from Trigger group 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
24	trigger_grp2	Enable/disable trigger_grp2 interrupt. Interrupt from Trigger group 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
23	trigger_grp1	Enable/disable trigger_grp1 interrupt. Interrupt from Trigger group 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

22	trigger_grp0	Enable/disable trigger_grp0 interrupt. Interrupt from Trigger group 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
21	fifo_out0_extra	Enable/disable fifo_extra_out0 interrupt. Interrupt from fifo_extra_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
20	fifo_in0_extra	Enable/disable fifo_extra_in0 interrupt. Interrupt from fifo_extra_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
19	fifo_out0	Enable/disable fifo_out0 interrupt. Interrupt from fifo_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
18	fifo_in0	Enable/disable fifo_in0 interrupt. Interrupt from fifo_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
17	dmc_out0	Enable/disable dmc_out0 interrupt. Interrupt from dmc_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
16	dmc_in0	Enable/disable dmc_in0 interrupt. Interrupt from dmc_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
15	spu0_7	Enable/disable spu0_7 interrupt. Interrupt from SPU0 software, bit 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
14	spu0_6	Enable/disable spu0_6 interrupt. Interrupt from SPU0 software, bit 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
13	spu0_5	Enable/disable spu0_5 interrupt. Interrupt from SPU0 software, bit 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
12	spu0_4	Enable/disable spu0_4 interrupt. Interrupt from SPU0 software, bit 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

11	spu0_3	Enable/disable spu0_3 interrupt. Interrupt from SPU0 software, bit 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
10	spu0_2	Enable/disable spu0_2 interrupt. Interrupt from SPU0 software, bit 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
9	spu0_1	Enable/disable spu0_1 interrupt. Interrupt from SPU0 software, bit 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
8	spu0_0	Enable/disable spu0_0 interrupt. Interrupt from SPU0 software, bit 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
7	mpu_7	Enable/disable mpu_7 interrupt. Interrupt from MPU software, bit 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
6	mpu_6	Enable/disable mpu_6 interrupt. Interrupt from MPU software, bit 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
5	mpu_5	Enable/disable mpu_5 interrupt. Interrupt from MPU software, bit 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	mpu_4	Enable/disable mpu_4 interrupt. Interrupt from MPU software, bit 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	mpu_3	Enable/disable mpu_3 interrupt. Interrupt from MPU software, bit 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	mpu_2	Enable/disable mpu_2 interrupt. Interrupt from MPU software, bit 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	mpu_1	Enable/disable mpu_1 interrupt. Interrupt from MPU software, bit 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

0	mpu_0	Enable/disable mpu_0 interrupt. Interrupt from MPU software, bit 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.27.30 rw_ack_intr2

Address	0x78
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts to CPU.

Bit(s)	Name	Description	Value
15	spu0_7	Acknowledge spu0_7 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
14	spu0_6	Acknowledge spu0_6 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
13	spu0_5	Acknowledge spu0_5 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
12	spu0_4	Acknowledge spu0_4 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
11	spu0_3	Acknowledge spu0_3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
10	spu0_2	Acknowledge spu0_2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
9	spu0_1	Acknowledge spu0_1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
8	spu0_0	Acknowledge spu0_0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
7	mpu_7	Acknowledge mpu_7 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
6	mpu_6	Acknowledge mpu_6 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
5	mpu_5	Acknowledge mpu_5 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
4	mpu_4	Acknowledge mpu_4 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

3	mpu_3	Acknowledge mpu_3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	mpu_2	Acknowledge mpu_2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	mpu_1	Acknowledge mpu_1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	mpu_0	Acknowledge mpu_0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.27.31 r_intr2

Address	0x7c
Default	
Type	Read
Description	Unmasked interrupts. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	timer_grp1	Interrupt timer_grp1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	timer_grp0	Interrupt timer_grp0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	trigger_grp7	Interrupt trigger_grp7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	trigger_grp6	Interrupt trigger_grp6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	trigger_grp5	Interrupt trigger_grp5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	trigger_grp4	Interrupt trigger_grp4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	trigger_grp3	Interrupt trigger_grp3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	trigger_grp2	Interrupt trigger_grp2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	trigger_grp1	Interrupt trigger_grp1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	trigger_grp0	Interrupt trigger_grp0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	fifo_out0_extra	Interrupt fifo_extra_out0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	fifo_in0_extra	Interrupt fifo_extra_in0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	fifo_out0	Interrupt fifo_out0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	fifo_in0	Interrupt fifo_in0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	dmc_out0	Interrupt dmc_out0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	dmc_in0	Interrupt dmc_in0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	spu0_7	Interrupt spu0_7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	spu0_6	Interrupt spu0_6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	spu0_5	Interrupt spu0_5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	spu0_4	Interrupt spu0_4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	spu0_3	Interrupt spu0_3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	spu0_2	Interrupt spu0_2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	spu0_1	Interrupt spu0_1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	spu0_0	Interrupt spu0_0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	mpu_7	Interrupt mpu_7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	mpu_6	Interrupt mpu_6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	mpu_5	Interrupt mpu_5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	mpu_4	Interrupt mpu_4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	mpu_3	Interrupt mpu_3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	mpu_2	Interrupt mpu_2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	mpu_1	Interrupt mpu_1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	mpu_0	Interrupt mpu_0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.27.32 r_masked_intr2

Address	0x80
Default	
Type	Read
Description	Masked interrupts. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	timer_grp1	Interrupt timer_grp1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	timer_grp0	Interrupt timer_grp0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	trigger_grp7	Interrupt trigger_grp7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	trigger_grp6	Interrupt trigger_grp6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	trigger_grp5	Interrupt trigger_grp5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	trigger_grp4	Interrupt trigger_grp4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	trigger_grp3	Interrupt trigger_grp3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	trigger_grp2	Interrupt trigger_grp2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	trigger_grp1	Interrupt trigger_grp1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	trigger_grp0	Interrupt trigger_grp0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	fifo_out0_extra	Interrupt fifo_out0_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	fifo_in0_extra	Interrupt fifo_in0_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	fifo_out0	Interrupt fifo_out0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	fifo_in0	Interrupt fifo_in0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	dmc_out0	Interrupt dmc_out0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	dmc_in0	Interrupt dmc_in0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	spu0_7	Interrupt spu0_7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	spu0_6	Interrupt spu0_6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	spu0_5	Interrupt spu0_5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	spu0_4	Interrupt spu0_4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	spu0_3	Interrupt spu0_3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	spu0_2	Interrupt spu0_2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	spu0_1	Interrupt spu0_1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	spu0_0	Interrupt spu0_0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	mpu_7	Interrupt mpu_7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	mpu_6	Interrupt mpu_6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	mpu_5	Interrupt mpu_5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	mpu_4	Interrupt mpu_4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	mpu_3	Interrupt mpu_3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	mpu_2	Interrupt mpu_2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	mpu_1	Interrupt mpu_1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	mpu_0	Interrupt mpu_0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.27.33 rw_intr3_mask

Address	0x84
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	timer_grp3	Enable/disable timer_grp3 interrupt. Interrupt from Timer group 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
30	timer_grp2	Enable/disable timer_grp2 interrupt. Interrupt from Timer group 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
29	trigger_grp7	Enable/disable trigger_grp7 interrupt. Interrupt from Trigger group 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
28	trigger_grp6	Enable/disable trigger_grp6 interrupt. Interrupt from Trigger group 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
27	trigger_grp5	Enable/disable trigger_grp5 interrupt. Interrupt from Trigger group 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
26	trigger_grp4	Enable/disable trigger_grp4 interrupt. Interrupt from Trigger group 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
25	trigger_grp3	Enable/disable trigger_grp3 interrupt. Interrupt from Trigger group 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
24	trigger_grp2	Enable/disable trigger_grp2 interrupt. Interrupt from Trigger group 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
23	trigger_grp1	Enable/disable trigger_grp1 interrupt. Interrupt from Trigger group 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

22	trigger_grp0	Enable/disable trigger_grp0 interrupt. Interrupt from Trigger group 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
21	fifo_out1_extra	Enable/disable fifo_extra_out1 interrupt. Interrupt from fifo_extra_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
20	fifo_in1_extra	Enable/disable fifo_extra_in1 interrupt. Interrupt from fifo_extra_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
19	fifo_out1	Enable/disable fifo_out1 interrupt. Interrupt from fifo_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
18	fifo_in1	Enable/disable fifo_in1 interrupt. Interrupt from fifo_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
17	dmc_out1	Enable/disable dmc_out1 interrupt. Interrupt from dmc_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
16	dmc_in1	Enable/disable dmc_in1 interrupt. Interrupt from dmc_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
15	spu1_7	Enable/disable spu1_7 interrupt. Interrupt from SPU1 software, bit 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
14	spu1_6	Enable/disable spu1_6 interrupt. Interrupt from SPU1 software, bit 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
13	spu1_5	Enable/disable spu1_5 interrupt. Interrupt from SPU1 software, bit 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
12	spu1_4	Enable/disable spu1_4 interrupt. Interrupt from SPU1 software, bit 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

11	spu1_3	Enable/disable spu1_3 interrupt. Interrupt from SPU1 software, bit 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
10	spu1_2	Enable/disable spu1_2 interrupt. Interrupt from SPU1 software, bit 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
9	spu1_1	Enable/disable spu1_1 interrupt. Interrupt from SPU1 software, bit 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
8	spu1_0	Enable/disable spu1_0 interrupt. Interrupt from SPU1 software, bit 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
7	mpu_23	Enable/disable mpu_23 interrupt. Interrupt from MPU software, bit 23. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
6	mpu_22	Enable/disable mpu_22 interrupt. Interrupt from MPU software, bit 22. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
5	mpu_21	Enable/disable mpu_21 interrupt. Interrupt from MPU software, bit 21. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	mpu_20	Enable/disable mpu_20 interrupt. Interrupt from MPU software, bit 20. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	mpu_19	Enable/disable mpu_19 interrupt. Interrupt from MPU software, bit 19. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	mpu_18	Enable/disable mpu_18 interrupt. Interrupt from MPU software, bit 18. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	mpu_17	Enable/disable mpu_17 interrupt. Interrupt from MPU software, bit 17. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

0	mpu_16	Enable/disable mpu_16 interrupt. Interrupt from MPU software, bit 16. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.27.34 rw_ack_intr3

Address	0x88
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts to CPU.

Bit(s)	Name	Description	Value
15	spu1_7	Acknowledge spu1_7 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
14	spu1_6	Acknowledge spu1_6 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
13	spu1_5	Acknowledge spu1_5 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
12	spu1_4	Acknowledge spu1_4 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
11	spu1_3	Acknowledge spu1_3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
10	spu1_2	Acknowledge spu1_2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
9	spu1_1	Acknowledge spu1_1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
8	spu1_0	Acknowledge spu1_0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
7	mpu_23	Acknowledge mpu_23 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
6	mpu_22	Acknowledge mpu_22 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
5	mpu_21	Acknowledge mpu_21 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
4	mpu_20	Acknowledge mpu_20 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

3	mpu_19	Acknowledge mpu_19 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	mpu_18	Acknowledge mpu_18 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	mpu_17	Acknowledge mpu_17 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	mpu_16	Acknowledge mpu_16 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.27.35 r_intr3

Address	0x8c
Default	
Type	Read
Description	Unmasked interrupts. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	timer_grp3	Interrupt timer_grp3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	timer_grp2	Interrupt timer_grp2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	trigger_grp7	Interrupt trigger_grp7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	trigger_grp6	Interrupt trigger_grp6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	trigger_grp5	Interrupt trigger_grp5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	trigger_grp4	Interrupt trigger_grp4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	trigger_grp3	Interrupt trigger_grp3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	trigger_grp2	Interrupt trigger_grp2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	trigger_grp1	Interrupt trigger_grp1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	trigger_grp0	Interrupt trigger_grp0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	fifo_out1_extra	Interrupt fifo_extra_out1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	fifo_in1_extra	Interrupt fifo_extra_in1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	fifo_out1	Interrupt fifo_out1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	fifo_in1	Interrupt fifo_in1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	dmc_out1	Interrupt dmc_out1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	dmc_in1	Interrupt dmc_in1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	spu1_7	Interrupt spu1_7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	spu1_6	Interrupt spu1_6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	spu1_5	Interrupt spu1_5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	spu1_4	Interrupt spu1_4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	spu1_3	Interrupt spu1_3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	spu1_2	Interrupt spu1_2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	spu1_1	Interrupt spu1_1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	spu1_0	Interrupt spu1_0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	mpu_23	Interrupt mpu_23 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	mpu_22	Interrupt mpu_22 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	mpu_21	Interrupt mpu_21 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	mpu_20	Interrupt mpu_20 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	mpu_19	Interrupt mpu_19 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	mpu_18	Interrupt mpu_18 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	mpu_17	Interrupt mpu_17 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	mpu_16	Interrupt mpu_16 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.27.36 r_masked_intr3

Address	0x90
Default	
Type	Read
Description	Masked interrupts. Interrupts to CPU.

Bit(s)	Name	Description	Value
31	timer_grp3	Interrupt timer_grp3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	timer_grp2	Interrupt timer_grp2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	trigger_grp7	Interrupt trigger_grp7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	trigger_grp6	Interrupt trigger_grp6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	trigger_grp5	Interrupt trigger_grp5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	trigger_grp4	Interrupt trigger_grp4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	trigger_grp3	Interrupt trigger_grp3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	trigger_grp2	Interrupt trigger_grp2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	trigger_grp1	Interrupt trigger_grp1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	trigger_grp0	Interrupt trigger_grp0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	fifo_out1_extra	Interrupt fifo_out1_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	fifo_in1_extra	Interrupt fifo_in1_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	fifo_out1	Interrupt fifo_out1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	fifo_in1	Interrupt fifo_in1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	dmc_out1	Interrupt dmc_out1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	dmc_in1	Interrupt dmc_in1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	spu1_7	Interrupt spu1_7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	spu1_6	Interrupt spu1_6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	spu1_5	Interrupt spu1_5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	spu1_4	Interrupt spu1_4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	spu1_3	Interrupt spu1_3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	spu1_2	Interrupt spu1_2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	spu1_1	Interrupt spu1_1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	spu1_0	Interrupt spu1_0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	mpu_23	Interrupt mpu_23 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	mpu_22	Interrupt mpu_22 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	mpu_21	Interrupt mpu_21 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	mpu_20	Interrupt mpu_20 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	mpu_19	Interrupt mpu_19 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	mpu_18	Interrupt mpu_18 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	mpu_17	Interrupt mpu_17 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	mpu_16	Interrupt mpu_16 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.28 iop_sw_mpu

Instance	Base Address
iop_sw_mpu	0xb0021300

25.28.1 rw_sw_cfg_owner

Address	0x0
Default	0x00000000
Type	Read/Write
Description	This register controls the owner of the Switch configuration registerbank.

Bit(s)	Name	Description	Value
1-0:2	cfg	Selects owner of sw_cfg. cpu: cpu is owner mpu: mpu is owner spu0: spu0 is owner spu1: spu1 is owner	cpu=0 mpu=1 spu0=2 spu1=3

25.28.2 rw_mc_ctrl

Address	0x4
Default	
Type	Read/Write
Description	Control register for the MC. A write to this register requests ownership of MC.

Bit(s)	Name	Description	Value
7	wr_spu1_mem	Enable writes to SPU1 memory. no: No writes to SPU1 memory yes: Write to SPU1 memory	no=0 yes=1
6	wr_spu0_mem	Enable writes to SPU0 memory. no: No writes to SPU0 memory yes: Write to SPU0 memory	no=0 yes=1
5-3:3	size	Size (in bytes) of data to read or write. MC can read/write at most four bytes.	
2-1:2	cmd	Command to perform. copy: Copy data from SMIF and write to I/O memory reg_copy: Data from mc_data is written to I/O memory rd: Read from SMIF to r_mc_data wr: Write rw_mc_data to SMIF	copy=0 reg_copy=1 rd=2 wr=3
0	keep_owner	Enables ownership to be kept after finished operation. no: Ownerships is automatically released yes: Ownership is kept	no=0 yes=1

25.28.3 rw_mc_data

Address	0x8
Default	
Type	Read/Write
Description	Data for write to system memory. Depending on size in rw_mc_ctrl some parts of value can be don't care. If rw_mc_ctrl.cmd is set to reg_copy the write to this register performs the write to I/O memory. If rw_mc_ctrl.keep_owner is not set the ownership of MC will be released.

Bit(s)	Name	Description	Value
31-0:32	val	Data value to be written to system memory.	

25.28.4 rw_mc_addr

Address	0xc
Default	
Type	Read/Write
Description	The system memory address to perform operation on. rw_mc_addr states the address which will be used by the selected command in rw_mc_ctrl.cmd. A write to this register starts the memory operation. If operation is wr or copy and rw_mc_ctrl.keep_owner isn't set the ownership will be lost when operation is done.

25.28.5 rs_mc_data/r_mc_data

Address	0x10/0x14
Default	
Type	Read with side effects/Read
Description	Data read from system memory at address rw_mc_addr. When reading rs_mc_data ownership of MC is released.

25.28.6 r_mc_stat

Address	0x18
Default	
Type	Read
Description	Status of the MC.

Bit(s)	Name	Description	Value
7	owned_by_spu1	MC is owned by SPU1. no: Not owned by SPU1 yes: Owned by SPU1	no=0 yes=1
6	owned_by_spu0	MC is owned by SPU0. no: Not owned by SPU0 yes: Owned by SPU0	no=0 yes=1
5	owned_by_mpu	MC is owned by MPU. no: Not owned by MPU yes: Owned by MPU	no=0 yes=1
4	owned_by_cpu	MC is owned by CPU. no: Not owned by CPU yes: Owned by CPU	no=0 yes=1
3	busy_spu1	MC is busy performing command for SPU1. no: Not busy yes: Busy	no=0 yes=1
2	busy_spu0	MC is busy performing command for SPU0. no: Not busy yes: Busy	no=0 yes=1
1	busy_mpu	MC is busy performing command for MPU. no: Not busy yes: Busy	no=0 yes=1
0	busy_cpu	MC is busy performing command for CPU. no: Not busy yes: Busy	no=0 yes=1

25.28.7 rw_bus0_clr_mask

Address	0x1c
Default	0x00000000
Type	Read/Write
Description	Clear bits in BUS0.

Bit(s)	Name	Description	Value
31-24:8	byte3	Used to clear bits in BUS0[31:24].	
23-16:8	byte2	Used to clear bits in BUS0[23:16].	
15-8:8	byte1	Used to clear bits in BUS0[15:8].	
7-0:8	byte0	Used to clear bits in BUS0[7:0].	

25.28.8 rw_bus0_set_mask

Address	0x20
Default	0x00000000
Type	Read/Write
Description	Set bits in BUS0.

Bit(s)	Name	Description	Value
31-24:8	byte3	Used to set bits in BUS0[31:24].	
23-16:8	byte2	Used to set bits in BUS0[23:16].	
15-8:8	byte1	Used to set bits in BUS0[15:8].	
7-0:8	byte0	Used to set bits in BUS0[7:0].	

25.28.9 rw_bus0_oe_clr_mask

Address	0x24
Default	0x00000000
Type	Read/Write
Description	Clear OE signals for BUS0.

Bit(s)	Name	Description	Value
3	byte3	Used to clear OE for BUS0[31:24].	
2	byte2	Used to clear OE for BUS0[23:16].	
1	byte1	Used to clear OE for BUS0[15:8].	
0	byte0	Used to clear OE for BUS0[7:0].	

25.28.10 rw_bus0_oe_set_mask

Address	0x28
Default	0x00000000
Type	Read/Write
Description	Set OE signals for BUS0.

Bit(s)	Name	Description	Value
3	byte3	Used to set OE for BUS0[31:24].	
2	byte2	Used to set OE for BUS0[23:16].	
1	byte1	Used to set OE for BUS0[15:8].	
0	byte0	Used to set OE for BUS0[7:0].	

25.28.11 r_bus0_in

Address	0x2c
Default	
Type	Read
Description	Read register for BUS0.

25.28.12 rw_bus1_clr_mask

Address	0x30
Default	0x00000000
Type	Read/Write
Description	Clear bits in BUS1.

Bit(s)	Name	Description	Value
31-24:8	byte3	Used to clear bits in BUS1[31:24].	
23-16:8	byte2	Used to clear bits in BUS1[23:16].	
15-8:8	byte1	Used to clear bits in BUS1[15:8].	
7-0:8	byte0	Used to clear bits in BUS1[7:0].	

25.28.13 rw_bus1_set_mask

Address	0x34
Default	0x00000000
Type	Read/Write
Description	Set bits in BUS1.

Bit(s)	Name	Description	Value
31-24:8	byte3	Used to set bits in BUS1[31:24].	
23-16:8	byte2	Used to set bits in BUS1[23:16].	
15-8:8	byte1	Used to set bits in BUS1[15:8].	
7-0:8	byte0	Used to set bits in BUS1[7:0].	

25.28.14 rw_bus1_oe_clr_mask

Address	0x38
Default	0x00000000
Type	Read/Write
Description	Clear OE signals for BUS1.

Bit(s)	Name	Description	Value
3	byte3	Used to clear OE for BUS1[31:24].	
2	byte2	Used to clear OE for BUS1[23:16].	
1	byte1	Used to clear OE for BUS1[15:8].	
0	byte0	Used to clear OE for BUS1[7:0].	

25.28.15 rw_bus1_oe_set_mask

Address	0x3c
Default	0x00000000
Type	Read/Write
Description	Set OE signals for BUS1.

Bit(s)	Name	Description	Value
3	byte3	Used to set OE for BUS1[31:24].	
2	byte2	Used to set OE for BUS1[23:16].	
1	byte1	Used to set OE for BUS1[15:8].	
0	byte0	Used to set OE for BUS1[7:0].	

25.28.16 r_bus1_in

Address	0x40
Default	
Type	Read
Description	Read register for BUS1.

25.28.17 rw_gio_clr_mask

Address	0x44
Default	0x00000000
Type	Read/Write
Description	Clear bits in GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Used to clear bits in GIO.	

25.28.18 rw_gio_set_mask

Address	0x48
Default	0x00000000
Type	Read/Write
Description	Set bits in GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Used to set bits in GIO.	

25.28.19 rw_gio_oe_clr_mask

Address	0x4c
Default	0x00000000
Type	Read/Write
Description	Clear OE signals for GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Used to clear OE for GIO.	

25.28.20 rw_gio_oe_set_mask

Address	0x50
Default	0x00000000
Type	Read/Write
Description	Set OE signals for GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Used to set OE for GIO.	

25.28.21 r_gio_in

Address	0x54
Default	
Type	Read
Description	Read register for GIO.

25.28.22 rw_cpu_intr

Address	0x58
Default	
Type	Read/Write
Description	This is used to set interrupts to CPU.

Bit(s)	Name	Description	Value
31	intr31	Set software interrupt 31, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
30	intr30	Set software interrupt 30, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
29	intr29	Set software interrupt 29, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
28	intr28	Set software interrupt 28, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
27	intr27	Set software interrupt 27, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
26	intr26	Set software interrupt 26, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
25	intr25	Set software interrupt 25, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
24	intr24	Set software interrupt 24, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
23	intr23	Set software interrupt 23, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
22	intr22	Set software interrupt 22, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
21	intr21	Set software interrupt 21, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
20	intr20	Set software interrupt 20, to CPU. set: Set interrupt nop: No operation	set=1 nop=0

19	intr19	Set software interrupt 19, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
18	intr18	Set software interrupt 18, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
17	intr17	Set software interrupt 17, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
16	intr16	Set software interrupt 16, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
15	intr15	Set software interrupt 15, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
14	intr14	Set software interrupt 14, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
13	intr13	Set software interrupt 13, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
12	intr12	Set software interrupt 12, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
11	intr11	Set software interrupt 11, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
10	intr10	Set software interrupt 10, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
9	intr9	Set software interrupt 9, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
8	intr8	Set software interrupt 8, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
7	intr7	Set software interrupt 7, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
6	intr6	Set software interrupt 6, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
5	intr5	Set software interrupt 5, to CPU. set: Set interrupt nop: No operation	set=1 nop=0

4	intr4	Set software interrupt 4, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
3	intr3	Set software interrupt 3, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
2	intr2	Set software interrupt 2, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
1	intr1	Set software interrupt 1, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
0	intr0	Set software interrupt 0, to CPU. set: Set interrupt nop: No operation	set=1 nop=0

25.28.23 r_cpu_intr

Address	0x5c
Default	
Type	Read
Description	This is used to check which MPU software interrupts are set to CPU (unmasked).

Bit(s)	Name	Description	Value
31	intr31	MPU software interrupt 31, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	intr30	MPU software interrupt 30, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	intr29	MPU software interrupt 29, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	intr28	MPU software interrupt 28, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	intr27	MPU software interrupt 27, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	intr26	MPU software interrupt 26, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	intr25	MPU software interrupt 25, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	intr24	MPU software interrupt 24, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	intr23	MPU software interrupt 23, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	intr22	MPU software interrupt 22, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	intr21	MPU software interrupt 21, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	intr20	MPU software interrupt 20, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	intr19	MPU software interrupt 19, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	intr18	MPU software interrupt 18, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	intr17	MPU software interrupt 17, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	intr16	MPU software interrupt 16, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	intr15	MPU software interrupt 15, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	intr14	MPU software interrupt 14, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	intr13	MPU software interrupt 13, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	intr12	MPU software interrupt 12, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	intr11	MPU software interrupt 11, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	intr10	MPU software interrupt 10, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	intr9	MPU software interrupt 9, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	intr8	MPU software interrupt 8, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	intr7	MPU software interrupt 7, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	intr6	MPU software interrupt 6, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	intr5	MPU software interrupt 5, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	intr4	MPU software interrupt 4, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	intr3	MPU software interrupt 3, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	intr2	MPU software interrupt 2, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	intr1	MPU software interrupt 1, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	intr0	MPU software interrupt 0, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.28.24 rw_intr_grp0_mask

Address	0x60
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupt group0, containing intr[3:0].

Bit(s)	Name	Description	Value
31	dmc_in1	Enable/disable dmc_in1 interrupt. Interrupt from dmc_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
30	fifo_in1_extra	Enable/disable fifo_in1_extra interrupt. Interrupt from extra register bank of fifo_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
29	fifo_in1	Enable/disable fifo_in1 interrupt. Interrupt from fifo_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
28	timer_grp3	Enable/disable timer_grp3 interrupt. Interrupt from Timer group 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
27	trigger_grp7	Enable/disable trigger_grp7 interrupt. Interrupt from Trigger group 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
26	trigger_grp3	Enable/disable trigger_grp3 interrupt. Interrupt from Trigger group 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
25	spu1_intr3	Enable/disable spu1_intr3 interrupt. Interrupt from SPU1 software, bit 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
24	spu0_intr3	Enable/disable spu0_intr3 interrupt. Interrupt from SPU0 software, bit 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
23	dmc_out1	Enable/disable dmc_out1 interrupt. Interrupt from dmc_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

22	fifo_out1_extra	Enable/disable fifo_out1_extra interrupt. Interrupt from extra register bank of fifo_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
21	fifo_out1	Enable/disable fifo_out1 interrupt. Interrupt from fifo_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
20	timer_grp2	Enable/disable timer_grp2 interrupt. Interrupt from Timer group 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
19	trigger_grp6	Enable/disable trigger_grp6 interrupt. Interrupt from Trigger group 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
18	trigger_grp2	Enable/disable trigger_grp2 interrupt. Interrupt from Trigger group 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
17	spu1_intr2	Enable/disable spu1_intr2 interrupt. Interrupt from SPU1 software, bit 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
16	spu0_intr2	Enable/disable spu0_intr2 interrupt. Interrupt from SPU0 software, bit 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
15	dmc_in0	Enable/disable dmc_in0 interrupt. Interrupt from dmc_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
14	fifo_in0_extra	Enable/disable fifo_in0_extra interrupt. Interrupt from extra register bank of fifo_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
13	fifo_in0	Enable/disable fifo_in0 interrupt. Interrupt from fifo_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
12	timer_grp1	Enable/disable timer_grp1 interrupt. Interrupt from Timer group 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

11	trigger_grp5	Enable/disable trigger_grp5 interrupt. Interrupt from Trigger group 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
10	trigger_grp1	Enable/disable trigger_grp1 interrupt. Interrupt from Trigger group 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
9	spu1_intr1	Enable/disable spu1_intr1 interrupt. Interrupt from SPU1 software, bit 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
8	spu0_intr1	Enable/disable spu0_intr1 interrupt. Interrupt from SPU0 software, bit 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
7	dmc_out0	Enable/disable dmc_out0 interrupt. Interrupt from dmc_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
6	fifo_out0_extra	Enable/disable fifo_out0_extra interrupt. Interrupt from extra register bank of fifo_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
5	fifo_out0	Enable/disable fifo_out0 interrupt. Interrupt from fifo_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	timer_grp0	Enable/disable timer_grp0 interrupt. Interrupt from Timer group 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	trigger_grp4	Enable/disable trigger_grp4 interrupt. Interrupt from Trigger group 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	trigger_grp0	Enable/disable trigger_grp0 interrupt. Interrupt from Trigger group 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	spu1_intr0	Enable/disable spu1_intr0 interrupt. Interrupt from SPU1 software, bit 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

0	spu0_intr0	Enable/disable spu0_intr0 interrupt. Interrupt from SPU0 software, bit 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.28.25 rw_ack_intr_grp0

Address	0x64
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupt group0, containing intr[3:0].

Bit(s)	Name	Description	Value
25	spu1_intr3	Acknowledge spu1_intr3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
24	spu0_intr3	Acknowledge spu0_intr3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
17	spu1_intr2	Acknowledge spu1_intr2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
16	spu0_intr2	Acknowledge spu0_intr2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
9	spu1_intr1	Acknowledge spu1_intr1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
8	spu0_intr1	Acknowledge spu0_intr1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	spu1_intr0	Acknowledge spu1_intr0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	spu0_intr0	Acknowledge spu0_intr0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.28.26 r_intr_grp0

Address	0x68
Default	
Type	Read
Description	Unmasked interrupts. Interrupt group0, containing intr[3:0].

Bit(s)	Name	Description	Value
31	dmc_in1	Interrupt dmc_in1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	fifo_in1_extra	Interrupt fifo_in1_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	fifo_in1	Interrupt fifo_in1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	timer_grp3	Interrupt timer_grp3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	trigger_grp7	Interrupt trigger_grp7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	trigger_grp3	Interrupt trigger_grp3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	spu1_intr3	Interrupt spu1_intr3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	spu0_intr3	Interrupt spu0_intr3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	dmc_out1	Interrupt dmc_out1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	fifo_out1_extra	Interrupt fifo_out1_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	fifo_out1	Interrupt fifo_out1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	timer_grp2	Interrupt timer_grp2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	trigger_grp6	Interrupt trigger_grp6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	trigger_grp2	Interrupt trigger_grp2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	spu1_intr2	Interrupt spu1_intr2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	spu0_intr2	Interrupt spu0_intr2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	dmc_in0	Interrupt dmc_in0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	fifo_in0_extra	Interrupt fifo_in0_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	fifo_in0	Interrupt fifo_in0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	timer_grp1	Interrupt timer_grp1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	trigger_grp5	Interrupt trigger_grp5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	trigger_grp1	Interrupt trigger_grp1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	spu1_intr1	Interrupt spu1_intr1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	spu0_intr1	Interrupt spu0_intr1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	dmc_out0	Interrupt dmc_out0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	fifo_out0_extra	Interrupt fifo_out0_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	fifo_out0	Interrupt fifo_out0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	timer_grp0	Interrupt timer_grp0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	trigger_grp4	Interrupt trigger_grp4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	trigger_grp0	Interrupt trigger_grp0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	spu1_intr0	Interrupt spu1_intr0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	spu0_intr0	Interrupt spu0_intr0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.28.27 r_masked_intr_grp0

Address	0x6c
Default	
Type	Read
Description	Masked interrupts. Interrupt group0, containing intr[3:0].

Bit(s)	Name	Description	Value
31	dmc_in1	Interrupt dmc_in1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	fifo_in1_extra	Interrupt fifo_in1_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	fifo_in1	Interrupt fifo_in1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	timer_grp3	Interrupt timer_grp3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	trigger_grp7	Interrupt trigger_grp7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	trigger_grp3	Interrupt trigger_grp3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	spu1_intr3	Interrupt spu1_intr3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	spu0_intr3	Interrupt spu0_intr3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	dmc_out1	Interrupt dmc_out1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	fifo_out1_extra	Interrupt fifo_out1_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	fifo_out1	Interrupt fifo_out1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	timer_grp2	Interrupt timer_grp2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	trigger_grp6	Interrupt trigger_grp6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	trigger_grp2	Interrupt trigger_grp2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	spu1_intr2	Interrupt spu1_intr2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	spu0_intr2	Interrupt spu0_intr2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	dmc_in0	Interrupt dmc_in0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	fifo_in0_extra	Interrupt fifo_in0_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	fifo_in0	Interrupt fifo_in0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	timer_grp1	Interrupt timer_grp1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	trigger_grp5	Interrupt trigger_grp5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	trigger_grp1	Interrupt trigger_grp1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	spu1_intr1	Interrupt spu1_intr1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	spu0_intr1	Interrupt spu0_intr1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	dmc_out0	Interrupt dmc_out0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	fifo_out0_extra	Interrupt fifo_out0_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	fifo_out0	Interrupt fifo_out0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	timer_grp0	Interrupt timer_grp0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	trigger_grp4	Interrupt trigger_grp4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	trigger_grp0	Interrupt trigger_grp0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	spu1_intr0	Interrupt spu1_intr0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	spu0_intr0	Interrupt spu0_intr0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.28.28 rw_intr_grp1_mask

Address	0x70
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupt group1, containing intr[7:4].

Bit(s)	Name	Description	Value
31	dmc_in1	Enable/disable dmc_in1 interrupt. Interrupt from dmc_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
30	fifo_out1_extra	Enable/disable fifo_out1_extra interrupt. Interrupt from extra register bank of fifo_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
29	fifo_out0	Enable/disable fifo_out0 interrupt. Interrupt from fifo_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
28	timer_grp3	Enable/disable timer_grp3 interrupt. Interrupt from Timer group 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
27	trigger_grp4	Enable/disable trigger_grp4 interrupt. Interrupt from Trigger group 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
26	trigger_grp3	Enable/disable trigger_grp3 interrupt. Interrupt from Trigger group 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
25	spu1_intr7	Enable/disable spu1_intr7 interrupt. Interrupt from SPU1 software, bit 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
24	spu0_intr7	Enable/disable spu0_intr7 interrupt. Interrupt from SPU0 software, bit 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
23	dmc_out1	Enable/disable dmc_out1 interrupt. Interrupt from dmc_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

22	fifo_in1_extra	Enable/disable fifo_in1_extra interrupt. Interrupt from extra register bank of fifo_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
21	fifo_in1	Enable/disable fifo_in1 interrupt. Interrupt from fifo_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
20	timer_grp2	Enable/disable timer_grp2 interrupt. Interrupt from Timer group 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
19	trigger_grp7	Enable/disable trigger_grp7 interrupt. Interrupt from Trigger group 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
18	trigger_grp2	Enable/disable trigger_grp2 interrupt. Interrupt from Trigger group 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
17	spu1_intr6	Enable/disable spu1_intr6 interrupt. Interrupt from SPU1 software, bit 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
16	spu0_intr6	Enable/disable spu0_intr6 interrupt. Interrupt from SPU0 software, bit 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
15	dmc_in0	Enable/disable dmc_in0 interrupt. Interrupt from dmc_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
14	fifo_out0_extra	Enable/disable fifo_out0_extra interrupt. Interrupt from extra register bank of fifo_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
13	fifo_out1	Enable/disable fifo_out1 interrupt. Interrupt from fifo_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
12	timer_grp1	Enable/disable timer_grp1 interrupt. Interrupt from Timer group 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

11	trigger_grp6	Enable/disable trigger_grp6 interrupt. Interrupt from Trigger group 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
10	trigger_grp1	Enable/disable trigger_grp1 interrupt. Interrupt from Trigger group 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
9	spu1_intr5	Enable/disable spu1_intr5 interrupt. Interrupt from SPU1 software, bit 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
8	spu0_intr5	Enable/disable spu0_intr5 interrupt. Interrupt from SPU0 software, bit 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
7	dmc_out0	Enable/disable dmc_out0 interrupt. Interrupt from dmc_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
6	fifo_in0_extra	Enable/disable fifo_in0_extra interrupt. Interrupt from extra register bank of fifo_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
5	fifo_in0	Enable/disable fifo_in0 interrupt. Interrupt from fifo_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	timer_grp0	Enable/disable timer_grp0 interrupt. Interrupt from Timer group 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	trigger_grp5	Enable/disable trigger_grp5 interrupt. Interrupt from Trigger group 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	trigger_grp0	Enable/disable trigger_grp0 interrupt. Interrupt from Trigger group 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	spu1_intr4	Enable/disable spu1_intr4 interrupt. Interrupt from SPU1 software, bit 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

0	spu0_intr4	Enable/disable spu0_intr4 interrupt. Interrupt from SPU0 software, bit 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.28.29 rw_ack_intr_grp1

Address	0x74
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupt group1, containing intr[7:4].

Bit(s)	Name	Description	Value
25	spu1_intr7	Acknowledge spu1_intr7 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
24	spu0_intr7	Acknowledge spu0_intr7 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
17	spu1_intr6	Acknowledge spu1_intr6 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
16	spu0_intr6	Acknowledge spu0_intr6 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
9	spu1_intr5	Acknowledge spu1_intr5 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
8	spu0_intr5	Acknowledge spu0_intr5 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	spu1_intr4	Acknowledge spu1_intr4 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	spu0_intr4	Acknowledge spu0_intr4 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.28.30 r_intr_grp1

Address	0x78
Default	
Type	Read
Description	Unmasked interrupts. Interrupt group1, containing intr[7:4].

Bit(s)	Name	Description	Value
31	dmc_in1	Interrupt dmc_in1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	fifo_out1_extra	Interrupt fifo_out1_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	fifo_out0	Interrupt fifo_out0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	timer_grp3	Interrupt timer_grp3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	trigger_grp4	Interrupt trigger_grp4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	trigger_grp3	Interrupt trigger_grp3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	spu1_intr7	Interrupt spu1_intr7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	spu0_intr7	Interrupt spu0_intr7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	dmc_out1	Interrupt dmc_out1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	fifo_in1_extra	Interrupt fifo_in1_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	fifo_in1	Interrupt fifo_in1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	timer_grp2	Interrupt timer_grp2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	trigger_grp7	Interrupt trigger_grp7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	trigger_grp2	Interrupt trigger_grp2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	spu1_intr6	Interrupt spu1_intr6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	spu0_intr6	Interrupt spu0_intr6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	dmc_in0	Interrupt dmc_in0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	fifo_out0_extra	Interrupt fifo_out0_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	fifo_out1	Interrupt fifo_out1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	timer_grp1	Interrupt timer_grp1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	trigger_grp6	Interrupt trigger_grp6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	trigger_grp1	Interrupt trigger_grp1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	spu1_intr5	Interrupt spu1_intr5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	spu0_intr5	Interrupt spu0_intr5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	dmc_out0	Interrupt dmc_out0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	fifo_in0_extra	Interrupt fifo_in0_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	fifo_in0	Interrupt fifo_in0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	timer_grp0	Interrupt timer_grp0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	trigger_grp5	Interrupt trigger_grp5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	trigger_grp0	Interrupt trigger_grp0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	spu1_intr4	Interrupt spu1_intr4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	spu0_intr4	Interrupt spu0_intr4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.28.31 r_masked_intr_grp1

Address	0x7c
Default	
Type	Read
Description	Masked interrupts. Interrupt group1, containing intr[7:4].

Bit(s)	Name	Description	Value
31	dmc_in1	Interrupt dmc_in1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	fifo_out1_extra	Interrupt fifo_out1_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	fifo_out0	Interrupt fifo_out0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	timer_grp3	Interrupt timer_grp3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	trigger_grp4	Interrupt trigger_grp4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	trigger_grp3	Interrupt trigger_grp3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	spu1_intr7	Interrupt spu1_intr7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	spu0_intr7	Interrupt spu0_intr7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	dmc_out1	Interrupt dmc_out1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	fifo_in1_extra	Interrupt fifo_in1_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	fifo_in1	Interrupt fifo_in1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	timer_grp2	Interrupt timer_grp2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	trigger_grp7	Interrupt trigger_grp7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	trigger_grp2	Interrupt trigger_grp2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	spu1_intr6	Interrupt spu1_intr6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	spu0_intr6	Interrupt spu0_intr6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	dmc_in0	Interrupt dmc_in0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	fifo_out0_extra	Interrupt fifo_out0_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	fifo_out1	Interrupt fifo_out1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	timer_grp1	Interrupt timer_grp1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	trigger_grp6	Interrupt trigger_grp6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	trigger_grp1	Interrupt trigger_grp1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	spu1_intr5	Interrupt spu1_intr5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	spu0_intr5	Interrupt spu0_intr5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	dmc_out0	Interrupt dmc_out0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	fifo_in0_extra	Interrupt fifo_in0_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	fifo_in0	Interrupt fifo_in0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	timer_grp0	Interrupt timer_grp0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	trigger_grp5	Interrupt trigger_grp5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	trigger_grp0	Interrupt trigger_grp0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	spu1_intr4	Interrupt spu1_intr4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	spu0_intr4	Interrupt spu0_intr4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.28.32 rw_intr_grp2_mask

Address	0x80
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupt group2, containing intr[11:8].

Bit(s)	Name	Description	Value
31	dmc_in1	Enable/disable dmc_in1 interrupt. Interrupt from dmc_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
30	fifo_in0_extra	Enable/disable fifo_in0_extra interrupt. Interrupt from extra register bank of fifo_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
29	fifo_in0	Enable/disable fifo_in0 interrupt. Interrupt from fifo_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
28	timer_grp3	Enable/disable timer_grp3 interrupt. Interrupt from Timer group 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
27	trigger_grp5	Enable/disable trigger_grp5 interrupt. Interrupt from Trigger group 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
26	trigger_grp3	Enable/disable trigger_grp3 interrupt. Interrupt from Trigger group 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
25	spu1_intr11	Enable/disable spu1_intr11 interrupt. Interrupt from SPU1 software, bit 11. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
24	spu0_intr11	Enable/disable spu0_intr11 interrupt. Interrupt from SPU0 software, bit 11. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
23	dmc_out1	Enable/disable dmc_out1 interrupt. Interrupt from dmc_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

22	fifo_out0_extra	Enable/disable fifo_out0_extra interrupt. Interrupt from extra register bank of fifo_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
21	fifo_out0	Enable/disable fifo_out0 interrupt. Interrupt from fifo_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
20	timer_grp2	Enable/disable timer_grp2 interrupt. Interrupt from Timer group 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
19	trigger_grp4	Enable/disable trigger_grp4 interrupt. Interrupt from Trigger group 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
18	trigger_grp2	Enable/disable trigger_grp2 interrupt. Interrupt from Trigger group 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
17	spu1_intr10	Enable/disable spu1_intr10 interrupt. Interrupt from SPU1 software, bit 10. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
16	spu0_intr10	Enable/disable spu0_intr10 interrupt. Interrupt from SPU0 software, bit 10. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
15	dmc_in0	Enable/disable dmc_in0 interrupt. Interrupt from dmc_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
14	fifo_in1_extra	Enable/disable fifo_in1_extra interrupt. Interrupt from extra register bank of fifo_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
13	fifo_in1	Enable/disable fifo_in1 interrupt. Interrupt from fifo_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
12	timer_grp1	Enable/disable timer_grp1 interrupt. Interrupt from Timer group 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

11	trigger_grp7	Enable/disable trigger_grp7 interrupt. Interrupt from Trigger group 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
10	trigger_grp1	Enable/disable trigger_grp1 interrupt. Interrupt from Trigger group 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
9	spu1_intr9	Enable/disable spu1_intr9 interrupt. Interrupt from SPU1 software, bit 9. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
8	spu0_intr9	Enable/disable spu0_intr9 interrupt. Interrupt from SPU0 software, bit 9. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
7	dmc_out0	Enable/disable dmc_out0 interrupt. Interrupt from dmc_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
6	fifo_out1_extra	Enable/disable fifo_out1_extra interrupt. Interrupt from extra register bank of fifo_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
5	fifo_out1	Enable/disable fifo_out1 interrupt. Interrupt from fifo_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	timer_grp0	Enable/disable timer_grp0 interrupt. Interrupt from Timer group 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	trigger_grp6	Enable/disable trigger_grp6 interrupt. Interrupt from Trigger group 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	trigger_grp0	Enable/disable trigger_grp0 interrupt. Interrupt from Trigger group 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	spu1_intr8	Enable/disable spu1_intr8 interrupt. Interrupt from SPU1 software, bit 8. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

0	spu0_intr8	Enable/disable spu0_intr8 interrupt. Interrupt from SPU0 software, bit 8. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.28.33 rw_ack_intr_grp2

Address	0x84
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupt group2, containing intr[11:8].

Bit(s)	Name	Description	Value
25	spu1_intr11	Acknowledge spu1_intr11 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
24	spu0_intr11	Acknowledge spu0_intr11 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
17	spu1_intr10	Acknowledge spu1_intr10 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
16	spu0_intr10	Acknowledge spu0_intr10 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
9	spu1_intr9	Acknowledge spu1_intr9 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
8	spu0_intr9	Acknowledge spu0_intr9 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	spu1_intr8	Acknowledge spu1_intr8 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	spu0_intr8	Acknowledge spu0_intr8 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.28.34 r_intr_grp2

Address	0x88
Default	
Type	Read
Description	Unmasked interrupts. Interrupt group2, containing intr[11:8].

Bit(s)	Name	Description	Value
31	dmc_in1	Interrupt dmc_in1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	fifo_in0_extra	Interrupt fifo_in0_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	fifo_in0	Interrupt fifo_in0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	timer_grp3	Interrupt timer_grp3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	trigger_grp5	Interrupt trigger_grp5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	trigger_grp3	Interrupt trigger_grp3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	spu1_intr11	Interrupt spu1_intr11 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	spu0_intr11	Interrupt spu0_intr11 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	dmc_out1	Interrupt dmc_out1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	fifo_out0_extra	Interrupt fifo_out0_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	fifo_out0	Interrupt fifo_out0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	timer_grp2	Interrupt timer_grp2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	trigger_grp4	Interrupt trigger_grp4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	trigger_grp2	Interrupt trigger_grp2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	spu1_intr10	Interrupt spu1_intr10 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	spu0_intr10	Interrupt spu0_intr10 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	dmc_in0	Interrupt dmc_in0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	fifo_in1_extra	Interrupt fifo_in1_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	fifo_in1	Interrupt fifo_in1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	timer_grp1	Interrupt timer_grp1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	trigger_grp7	Interrupt trigger_grp7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	trigger_grp1	Interrupt trigger_grp1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	spu1_intr9	Interrupt spu1_intr9 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	spu0_intr9	Interrupt spu0_intr9 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	dmc_out0	Interrupt dmc_out0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	fifo_out1_extra	Interrupt fifo_out1_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	fifo_out1	Interrupt fifo_out1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	timer_grp0	Interrupt timer_grp0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	trigger_grp6	Interrupt trigger_grp6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	trigger_grp0	Interrupt trigger_grp0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	spu1_intr8	Interrupt spu1_intr8 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	spu0_intr8	Interrupt spu0_intr8 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.28.35 r_masked_intr_grp2

Address	0x8c
Default	
Type	Read
Description	Masked interrupts. Interrupt group2, containing intr[11:8].

Bit(s)	Name	Description	Value
31	dmc_in1	Interrupt dmc_in1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	fifo_in0_extra	Interrupt fifo_in0_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	fifo_in0	Interrupt fifo_in0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	timer_grp3	Interrupt timer_grp3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	trigger_grp5	Interrupt trigger_grp5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	trigger_grp3	Interrupt trigger_grp3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	spu1_intr11	Interrupt spu1_intr11 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	spu0_intr11	Interrupt spu0_intr11 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	dmc_out1	Interrupt dmc_out1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	fifo_out0_extra	Interrupt fifo_out0_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	fifo_out0	Interrupt fifo_out0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	timer_grp2	Interrupt timer_grp2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	trigger_grp4	Interrupt trigger_grp4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	trigger_grp2	Interrupt trigger_grp2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	spu1_intr10	Interrupt spu1_intr10 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	spu0_intr10	Interrupt spu0_intr10 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	dmc_in0	Interrupt dmc_in0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	fifo_in1_extra	Interrupt fifo_in1_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	fifo_in1	Interrupt fifo_in1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	timer_grp1	Interrupt timer_grp1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	trigger_grp7	Interrupt trigger_grp7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	trigger_grp1	Interrupt trigger_grp1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	spu1_intr9	Interrupt spu1_intr9 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	spu0_intr9	Interrupt spu0_intr9 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	dmc_out0	Interrupt dmc_out0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	fifo_out1_extra	Interrupt fifo_out1_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	fifo_out1	Interrupt fifo_out1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	timer_grp0	Interrupt timer_grp0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	trigger_grp6	Interrupt trigger_grp6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	trigger_grp0	Interrupt trigger_grp0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	spu1_intr8	Interrupt spu1_intr8 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	spu0_intr8	Interrupt spu0_intr8 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.28.36 rw_intr_grp3_mask

Address	0x90
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupt group3, containing intr[15:12].

Bit(s)	Name	Description	Value
31	dmc_in1	Enable/disable dmc_in1 interrupt. Interrupt from dmc_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
30	fifo_out1_extra	Enable/disable fifo_out1_extra interrupt. Interrupt from extra register bank of fifo_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
29	fifo_out1	Enable/disable fifo_out1 interrupt. Interrupt from fifo_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
28	timer_grp3	Enable/disable timer_grp3 interrupt. Interrupt from Timer group 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
27	trigger_grp6	Enable/disable trigger_grp6 interrupt. Interrupt from Trigger group 6. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
26	trigger_grp3	Enable/disable trigger_grp3 interrupt. Interrupt from Trigger group 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
25	spu1_intr15	Enable/disable spu1_intr15 interrupt. Interrupt from SPU1 software, bit 15. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
24	spu0_intr15	Enable/disable spu0_intr15 interrupt. Interrupt from SPU0 software, bit 15. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
23	dmc_out1	Enable/disable dmc_out1 interrupt. Interrupt from dmc_out1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

22	fifo_in0_extra	Enable/disable fifo_in0_extra interrupt. Interrupt from extra register bank of fifo_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
21	fifo_in0	Enable/disable fifo_in0 interrupt. Interrupt from fifo_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
20	timer_grp2	Enable/disable timer_grp2 interrupt. Interrupt from Timer group 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
19	trigger_grp5	Enable/disable trigger_grp5 interrupt. Interrupt from Trigger group 5. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
18	trigger_grp2	Enable/disable trigger_grp2 interrupt. Interrupt from Trigger group 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
17	spu1_intr14	Enable/disable spu1_intr14 interrupt. Interrupt from SPU1 software, bit 14. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
16	spu0_intr14	Enable/disable spu0_intr14 interrupt. Interrupt from SPU0 software, bit 14. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
15	dmc_in0	Enable/disable dmc_in0 interrupt. Interrupt from dmc_in0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
14	fifo_out0_extra	Enable/disable fifo_out0_extra interrupt. Interrupt from extra register bank of fifo_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
13	fifo_out0	Enable/disable fifo_out0 interrupt. Interrupt from fifo_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
12	timer_grp1	Enable/disable timer_grp1 interrupt. Interrupt from Timer group 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

11	trigger_grp4	Enable/disable trigger_grp4 interrupt. Interrupt from Trigger group 4. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
10	trigger_grp1	Enable/disable trigger_grp1 interrupt. Interrupt from Trigger group 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
9	spu1_intr13	Enable/disable spu1_intr13 interrupt. Interrupt from SPU1 software, bit 13. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
8	spu0_intr13	Enable/disable spu0_intr13 interrupt. Interrupt from SPU0 software, bit 13. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
7	dmc_out0	Enable/disable dmc_out0 interrupt. Interrupt from dmc_out0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
6	fifo_in1_extra	Enable/disable fifo_in1_extra interrupt. Interrupt from extra register bank of fifo_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
5	fifo_in1	Enable/disable fifo_in1 interrupt. Interrupt from fifo_in1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	timer_grp0	Enable/disable timer_grp0 interrupt. Interrupt from Timer group 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	trigger_grp7	Enable/disable trigger_grp7 interrupt. Interrupt from Trigger group 7. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	trigger_grp0	Enable/disable trigger_grp0 interrupt. Interrupt from Trigger group 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	spu1_intr12	Enable/disable spu1_intr12 interrupt. Interrupt from SPU1 software, bit 12. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

0	spu0_intr12	Enable/disable spu0_intr12 interrupt. Interrupt from SPU0 software, bit 12. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.28.37 rw_ack_intr_grp3

Address	0x94
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupt group3, containing intr[15:12].

Bit(s)	Name	Description	Value
25	spu1_intr15	Acknowledge spu1_intr15 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
24	spu0_intr15	Acknowledge spu0_intr15 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
17	spu1_intr14	Acknowledge spu1_intr14 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
16	spu0_intr14	Acknowledge spu0_intr14 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
9	spu1_intr13	Acknowledge spu1_intr13 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
8	spu0_intr13	Acknowledge spu0_intr13 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	spu1_intr12	Acknowledge spu1_intr12 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	spu0_intr12	Acknowledge spu0_intr12 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.28.38 r_intr_grp3

Address	0x98
Default	
Type	Read
Description	Unmasked interrupts. Interrupt group3, containing intr[15:12].

Bit(s)	Name	Description	Value
31	dmc_in1	Interrupt dmc_in1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	fifo_out1_extra	Interrupt fifo_out1_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	fifo_out1	Interrupt fifo_out1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	timer_grp3	Interrupt timer_grp3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	trigger_grp6	Interrupt trigger_grp6 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	trigger_grp3	Interrupt trigger_grp3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	spu1_intr15	Interrupt spu1_intr15 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	spu0_intr15	Interrupt spu0_intr15 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	dmc_out1	Interrupt dmc_out1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	fifo_in0_extra	Interrupt fifo_in0_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	fifo_in0	Interrupt fifo_in0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	timer_grp2	Interrupt timer_grp2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	trigger_grp5	Interrupt trigger_grp5 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	trigger_grp2	Interrupt trigger_grp2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	spu1_intr14	Interrupt spu1_intr14 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	spu0_intr14	Interrupt spu0_intr14 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	dmc_in0	Interrupt dmc_in0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	fifo_out0_extra	Interrupt fifo_out0_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	fifo_out0	Interrupt fifo_out0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	timer_grp1	Interrupt timer_grp1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	trigger_grp4	Interrupt trigger_grp4 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	trigger_grp1	Interrupt trigger_grp1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	spu1_intr13	Interrupt spu1_intr13 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	spu0_intr13	Interrupt spu0_intr13 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	dmc_out0	Interrupt dmc_out0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	fifo_in1_extra	Interrupt fifo_in1_extra active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	fifo_in1	Interrupt fifo_in1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	timer_grp0	Interrupt timer_grp0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	trigger_grp7	Interrupt trigger_grp7 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	trigger_grp0	Interrupt trigger_grp0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	spu1_intr12	Interrupt spu1_intr12 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	spu0_intr12	Interrupt spu0_intr12 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.28.39 r_masked_intr_grp3

Address	0x9c
Default	
Type	Read
Description	Masked interrupts. Interrupt group3, containing intr[15:12].

Bit(s)	Name	Description	Value
31	dmc_in1	Interrupt dmc_in1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	fifo_out1_extra	Interrupt fifo_out1_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	fifo_out1	Interrupt fifo_out1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	timer_grp3	Interrupt timer_grp3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	trigger_grp6	Interrupt trigger_grp6 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	trigger_grp3	Interrupt trigger_grp3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	spu1_intr15	Interrupt spu1_intr15 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	spu0_intr15	Interrupt spu0_intr15 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	dmc_out1	Interrupt dmc_out1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	fifo_in0_extra	Interrupt fifo_in0_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	fifo_in0	Interrupt fifo_in0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	timer_grp2	Interrupt timer_grp2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	trigger_grp5	Interrupt trigger_grp5 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	trigger_grp2	Interrupt trigger_grp2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	spu1_intr14	Interrupt spu1_intr14 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	spu0_intr14	Interrupt spu0_intr14 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	dmc_in0	Interrupt dmc_in0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	fifo_out0_extra	Interrupt fifo_out0_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	fifo_out0	Interrupt fifo_out0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	timer_grp1	Interrupt timer_grp1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	trigger_grp4	Interrupt trigger_grp4 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	trigger_grp1	Interrupt trigger_grp1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	spu1_intr13	Interrupt spu1_intr13 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	spu0_intr13	Interrupt spu0_intr13 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	dmc_out0	Interrupt dmc_out0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	fifo_in1_extra	Interrupt fifo_in1_extra active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	fifo_in1	Interrupt fifo_in1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	timer_grp0	Interrupt timer_grp0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	trigger_grp7	Interrupt trigger_grp7 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	trigger_grp0	Interrupt trigger_grp0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	spu1_intr12	Interrupt spu1_intr12 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	spu0_intr12	Interrupt spu0_intr12 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.29 iop_sw_spu

Instance	Base Address
iop_sw_spu0	0xb0021400
iop_sw_spu1	0xb0021500

25.29.1 rw_mc_ctrl

Address	0x0
Default	
Type	Read/Write
Description	Control register for the MC. A write to this register requests ownership of MC.

Bit(s)	Name	Description	Value
7	wr_spu1_mem	Enable writes to SPU1 memory. no: No writes to SPU1 memory yes: Write to SPU1 memory	no=0 yes=1
6	wr_spu0_mem	Enable writes to SPU0 memory. no: No writes to SPU0 memory yes: Write to SPU0 memory	no=0 yes=1
5-3:3	size	Size (in bytes) of data to read or write. MC can read/write at most four bytes.	
2-1:2	cmd	Command to perform. copy: Copy data from SMIF and write to I/O memory reg_copy: Data from mc_data is written to I/O memory rd: Read from SMIF to r_mc_data wr: Write rw_mc_data to SMIF	copy=0 reg_copy=1 rd=2 wr=3
0	keep_owner	Enables ownership to be kept after finished operation. no: Ownerships is automatically released yes: Ownership is kept	no=0 yes=1

25.29.2 rw_mc_data

Address	0x4
Default	
Type	Read/Write
Description	Data for write to system memory. Depending on size in rw_mc_ctrl some parts of value can be don't care. If rw_mc_ctrl.cmd is set to reg_copy the write to this register performs the write to I/O memory. If rw_mc_ctrl.keep_owner is not set the ownership of MC will be released.

Bit(s)	Name	Description	Value
31-0:32	val	Data value to be written to system memory.	

25.29.3 rw_mc_addr

Address	0x8
Default	
Type	Read/Write
Description	The system memory address to perform operation on. rw_mc_addr states the address which will be used by the selected command in rw_mc_ctrl.cmd. A write to this register starts the memory operation. If operation is wr or copy and rw_mc_ctrl.keep_owner isn't set the ownership will be lost when operation is done.

25.29.4 rs_mc_data/r_mc_data

Address	0xc/0x10
Default	
Type	Read with side effects/Read
Description	Data read from system memory at address rw_mc_addr. When reading rs_mc_data ownership of MC is released.

25.29.5 r_mc_stat

Address	0x14
Default	
Type	Read
Description	Status of the MC.

Bit(s)	Name	Description	Value
7	owned_by_spu1	MC is owned by SPU1. no: Not owned by SPU1 yes: Owned by SPU1	no=0 yes=1
6	owned_by_spu0	MC is owned by SPU0. no: Not owned by SPU0 yes: Owned by SPU0	no=0 yes=1
5	owned_by_mpu	MC is owned by MPU. no: Not owned by MPU yes: Owned by MPU	no=0 yes=1
4	owned_by_cpu	MC is owned by CPU. no: Not owned by CPU yes: Owned by CPU	no=0 yes=1
3	busy_spu1	MC is busy performing command for SPU1. no: Not busy yes: Busy	no=0 yes=1
2	busy_spu0	MC is busy performing command for SPU0. no: Not busy yes: Busy	no=0 yes=1
1	busy_mpu	MC is busy performing command for MPU. no: Not busy yes: Busy	no=0 yes=1
0	busy_cpu	MC is busy performing command for CPU. no: Not busy yes: Busy	no=0 yes=1

25.29.6 rw_bus0_clr_mask

Address	0x18
Default	0x00000000
Type	Read/Write
Description	Clear bits in BUS0.

Bit(s)	Name	Description	Value
31-24:8	byte3	Used to clear bits in BUS0[31:24].	
23-16:8	byte2	Used to clear bits in BUS0[23:16].	
15-8:8	byte1	Used to clear bits in BUS0[15:8].	
7-0:8	byte0	Used to clear bits in BUS0[7:0].	

25.29.7 rw_bus0_set_mask

Address	0x1c
Default	0x00000000
Type	Read/Write
Description	Set bits in BUS0.

Bit(s)	Name	Description	Value
31-24:8	byte3	Used to set bits in BUS0[31:24].	
23-16:8	byte2	Used to set bits in BUS0[23:16].	
15-8:8	byte1	Used to set bits in BUS0[15:8].	
7-0:8	byte0	Used to set bits in BUS0[7:0].	

25.29.8 rw_bus0_oe_clr_mask

Address	0x20
Default	0x00000000
Type	Read/Write
Description	Clear OE signals for BUS0.

Bit(s)	Name	Description	Value
3	byte3	Used to clear OE for BUS0[31:24].	
2	byte2	Used to clear OE for BUS0[23:16].	
1	byte1	Used to clear OE for BUS0[15:8].	
0	byte0	Used to clear OE for BUS0[7:0].	

25.29.9 rw_bus0_oe_set_mask

Address	0x24
Default	0x00000000
Type	Read/Write
Description	Set OE signals for BUS0.

Bit(s)	Name	Description	Value
3	byte3	Used to set OE for BUS0[31:24].	
2	byte2	Used to set OE for BUS0[23:16].	
1	byte1	Used to set OE for BUS0[15:8].	
0	byte0	Used to set OE for BUS0[7:0].	

25.29.10 r_bus0_in

Address	0x28
Default	
Type	Read
Description	Read register for BUS0.

25.29.11 rw_bus1_clr_mask

Address	0x2c
Default	0x00000000
Type	Read/Write
Description	Clear bits in BUS1.

Bit(s)	Name	Description	Value
31-24:8	byte3	Used to clear bits in BUS1[31:24].	
23-16:8	byte2	Used to clear bits in BUS1[23:16].	
15-8:8	byte1	Used to clear bits in BUS1[15:8].	
7-0:8	byte0	Used to clear bits in BUS1[7:0].	

25.29.12 rw_bus1_set_mask

Address	0x30
Default	0x00000000
Type	Read/Write
Description	Set bits in BUS1.

Bit(s)	Name	Description	Value
31-24:8	byte3	Used to set bits in BUS1[31:24].	
23-16:8	byte2	Used to set bits in BUS1[23:16].	
15-8:8	byte1	Used to set bits in BUS1[15:8].	
7-0:8	byte0	Used to set bits in BUS1[7:0].	

25.29.13 rw_bus1_oe_clr_mask

Address	0x34
Default	0x00000000
Type	Read/Write
Description	Clear OE signals for BUS1.

Bit(s)	Name	Description	Value
3	byte3	Used to clear OE for BUS1[31:24].	
2	byte2	Used to clear OE for BUS1[23:16].	
1	byte1	Used to clear OE for BUS1[15:8].	
0	byte0	Used to clear OE for BUS1[7:0].	

25.29.14 rw_bus1_oe_set_mask

Address	0x38
Default	0x00000000
Type	Read/Write
Description	Set OE signals for BUS1.

Bit(s)	Name	Description	Value
3	byte3	Used to set OE for BUS1[31:24].	
2	byte2	Used to set OE for BUS1[23:16].	
1	byte1	Used to set OE for BUS1[15:8].	
0	byte0	Used to set OE for BUS1[7:0].	

25.29.15 r_bus1_in

Address	0x3c
Default	
Type	Read
Description	Read register for BUS1.

25.29.16 rw_gio_clr_mask

Address	0x40
Default	0x00000000
Type	Read/Write
Description	Clear bits in GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Used to clear bits in GIO.	

25.29.17 rw_gio_set_mask

Address	0x44
Default	0x00000000
Type	Read/Write
Description	Set bits in GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Used to set bits in GIO.	

25.29.18 rw_gio_oe_clr_mask

Address	0x48
Default	0x00000000
Type	Read/Write
Description	Clear OE signals for GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Used to clear OE for GIO.	

25.29.19 rw_gio_oe_set_mask

Address	0x4c
Default	0x00000000
Type	Read/Write
Description	Set OE signals for GIO.

Bit(s)	Name	Description	Value
31-0:32	val	Used to set OE for GIO.	

25.29.20 r_gio_in

Address	0x50
Default	
Type	Read
Description	Read register for GIO.

25.29.21 rw_bus0_clr_mask_lo

Address	0x54
Default	
Type	Read/Write
Description	Clear bits in BUS0, Note: BUS0[31:16] is not affected.

Bit(s)	Name	Description	Value
15-8:8	byte1	Used to clear bits in BUS0[15:8].	
7-0:8	byte0	Used to clear bits in BUS0[7:0].	

25.29.22 rw_bus0_clr_mask_hi

Address	0x58
Default	
Type	Read/Write
Description	Clear bits in BUS0, Note: BUS0[15:0] is not affected.

Bit(s)	Name	Description	Value
15-8:8	byte3	Used to clear bits in BUS0[31:24].	
7-0:8	byte2	Used to clear bits in BUS0[23:16].	

25.29.23 rw_bus0_set_mask_lo

Address	0x5c
Default	
Type	Read/Write
Description	Set bits in BUS0, Note: BUS0[31:16] is not affected.

Bit(s)	Name	Description	Value
15-8:8	byte1	Used to set bits in BUS0[15:8].	
7-0:8	byte0	Used to set bits in BUS0[7:0].	

25.29.24 rw_bus0_set_mask_hi

Address	0x60
Default	
Type	Read/Write
Description	Set bits in BUS0, Note: BUS0[15:0] is not affected.

Bit(s)	Name	Description	Value
15-8:8	byte3	Used to set bits in BUS0[31:24].	
7-0:8	byte2	Used to set bits in BUS0[23:16].	

25.29.25 rw_bus1_clr_mask_lo

Address	0x64
Default	
Type	Read/Write
Description	Clear bits in BUS1, Note: BUS1[31:16] is not affected.

Bit(s)	Name	Description	Value
15-8:8	byte1	Used to clear bits in BUS1[15:8].	
7-0:8	byte0	Used to clear bits in BUS1[7:0].	

25.29.26 rw_bus1_clr_mask_hi

Address	0x68
Default	
Type	Read/Write
Description	Clear bits in BUS1, Note: BUS1[15:0] is not affected.

Bit(s)	Name	Description	Value
15-8:8	byte3	Used to clear bits in BUS1[31:24].	
7-0:8	byte2	Used to clear bits in BUS1[23:16].	

25.29.27 rw_bus1_set_mask_lo

Address	0x6c
Default	
Type	Read/Write
Description	Set bits in BUS1, Note: BUS1[31:16] is not affected.

Bit(s)	Name	Description	Value
15-8:8	byte1	Used to set bits in BUS1[15:8].	
7-0:8	byte0	Used to set bits in BUS1[7:0].	

25.29.28 rw_bus1_set_mask_hi

Address	0x70
Default	
Type	Read/Write
Description	Set bits in BUS1, Note: BUS1[15:0] is not affected.

Bit(s)	Name	Description	Value
15-8:8	byte3	Used to set bits in BUS1[31:24].	
7-0:8	byte2	Used to set bits in BUS1[23:16].	

25.29.29 rw_gio_clr_mask_lo

Address	0x74
Default	
Type	Read/Write
Description	Clear bits in GIO, Note: GIO[31:16] is not affected.

Bit(s)	Name	Description	Value
15-0:16	val	Used to clear bits in GIO[15:0].	

25.29.30 rw_gio_clr_mask_hi

Address	0x78
Default	
Type	Read/Write
Description	Clear bits in GIO, Note: GIO[15:0] is not affected.

Bit(s)	Name	Description	Value
15-0:16	val	Used to clear bits in GIO[31:16].	

25.29.31 rw_gio_set_mask_lo

Address	0x7c
Default	
Type	Read/Write
Description	Set bits in GIO, Note: GIO[31:16] is not affected.

Bit(s)	Name	Description	Value
15-0:16	val	Used to set bits in GIO[15:0].	

25.29.32 rw_gio_set_mask_hi

Address	0x80
Default	
Type	Read/Write
Description	Set bits in GIO, Note: GIO[15:0] is not affected.

Bit(s)	Name	Description	Value
15-0:16	val	Used to set bits in GIO[31:16].	

25.29.33 rw_gio_oe_clr_mask_lo

Address	0x84
Default	
Type	Read/Write
Description	Clear OE signals for GIO, Note: OE for GIO[31:16] is not affected.

Bit(s)	Name	Description	Value
15-0:16	val	Used to clear OE for GIO[15:0].	

25.29.34 rw_gio_oe_clr_mask_hi

Address	0x88
Default	
Type	Read/Write
Description	Clear OE signals for GIO, Note: OE for GIO[15:0] is not affected.

Bit(s)	Name	Description	Value
15-0:16	val	Used to clear OE for GIO[31:16].	

25.29.35 rw_gio_oe_set_mask_lo

Address	0x8c
Default	
Type	Read/Write
Description	Set OE signals for GIO, Note: OE for GIO[31:16] is not affected.

Bit(s)	Name	Description	Value
15-0:16	val	Used to set OE for GIO[15:0].	

25.29.36 rw_gio_oe_set_mask_hi

Address	0x90
Default	
Type	Read/Write
Description	Set OE signals for GIO, Note: OE for GIO[15:0] is not affected.

Bit(s)	Name	Description	Value
15-0:16	val	Used to set OE for GIO[31:16].	

25.29.37 rw_cpu_intr

Address	0x94
Default	
Type	Read/Write
Description	This is used to set interrupts to CPU.

Bit(s)	Name	Description	Value
15	intr15	Set software interrupt 15, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
14	intr14	Set software interrupt 14, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
13	intr13	Set software interrupt 13, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
12	intr12	Set software interrupt 12, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
11	intr11	Set software interrupt 11, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
10	intr10	Set software interrupt 10, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
9	intr9	Set software interrupt 9, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
8	intr8	Set software interrupt 8, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
7	intr7	Set software interrupt 7, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
6	intr6	Set software interrupt 6, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
5	intr5	Set software interrupt 5, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
4	intr4	Set software interrupt 4, to CPU. set: Set interrupt nop: No operation	set=1 nop=0

3	intr3	Set software interrupt 3, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
2	intr2	Set software interrupt 2, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
1	intr1	Set software interrupt 1, to CPU. set: Set interrupt nop: No operation	set=1 nop=0
0	intr0	Set software interrupt 0, to CPU. set: Set interrupt nop: No operation	set=1 nop=0

25.29.38 r_cpu_intr

Address	0x98
Default	
Type	Read
Description	This is used to check which SPU software interrupts are set to CPU (unmasked).

Bit(s)	Name	Description	Value
15	intr15	SPU software interrupt 15, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	intr14	SPU software interrupt 14, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	intr13	SPU software interrupt 13, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	intr12	SPU software interrupt 12, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	intr11	SPU software interrupt 11, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	intr10	SPU software interrupt 10, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	intr9	SPU software interrupt 9, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	intr8	SPU software interrupt 8, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	intr7	SPU software interrupt 7, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	intr6	SPU software interrupt 6, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	intr5	SPU software interrupt 5, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
4	intr4	SPU software interrupt 4, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

3	intr3	SPU software interrupt 3, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	intr2	SPU software interrupt 2, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	intr1	SPU software interrupt 1, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	intr0	SPU software interrupt 0, to CPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.29.39 r_hw_intr

Address	0x9c
Default	
Type	Read
Description	This is used to check which interrupt are set from all hw modules (unmasked).

Bit(s)	Name	Description	Value
23	dmc_in1	Interrupt (unmasked) from dmc_in1. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	dmc_out1	Interrupt (unmasked) from dmc_out1. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	dmc_in0	Interrupt (unmasked) from dmc_in0. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	dmc_out0	Interrupt (unmasked) from dmc_out0. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
19	fifo_in1_extra	Interrupt (unmasked) from fifo_in1_extra. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	fifo_in1	Interrupt (unmasked) from fifo_in1. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	fifo_out1_extra	Interrupt (unmasked) from fifo_out1_extra. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	fifo_out1	Interrupt (unmasked) from fifo_out1. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	fifo_in0_extra	Interrupt (unmasked) from fifo_in0_extra. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	fifo_in0	Interrupt (unmasked) from fifo_in0. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	fifo_out0_extra	Interrupt (unmasked) from fifo_out0_extra. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	fifo_out0	Interrupt (unmasked) from fifo_out0. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

11	timer_grp3	Interrupt (unmasked) from Timer group 3. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	timer_grp2	Interrupt (unmasked) from Timer group 2. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	timer_grp1	Interrupt (unmasked) from Timer group 1. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	timer_grp0	Interrupt (unmasked) from Timer group 0. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	trigger_grp7	Interrupt (unmasked) from Trigger group 7. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	trigger_grp6	Interrupt (unmasked) from Trigger group 6. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	trigger_grp5	Interrupt (unmasked) from Trigger group 5. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
4	trigger_grp4	Interrupt (unmasked) from Trigger group 4. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	trigger_grp3	Interrupt (unmasked) from Trigger group 3. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	trigger_grp2	Interrupt (unmasked) from Trigger group 2. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	trigger_grp1	Interrupt (unmasked) from Trigger group 1. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	trigger_grp0	Interrupt (unmasked) from Trigger group 0. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.29.40 rw_mpu_intr

Address	0xa0
Default	
Type	Read/Write
Description	This is used to set interrupts to MPU.

Bit(s)	Name	Description	Value
15	intr15	Set software interrupt 15, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
14	intr14	Set software interrupt 14, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
13	intr13	Set software interrupt 13, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
12	intr12	Set software interrupt 12, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
11	intr11	Set software interrupt 11, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
10	intr10	Set software interrupt 10, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
9	intr9	Set software interrupt 9, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
8	intr8	Set software interrupt 8, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
7	intr7	Set software interrupt 7, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
6	intr6	Set software interrupt 6, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
5	intr5	Set software interrupt 5, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
4	intr4	Set software interrupt 4, to MPU. set: Set interrupt nop: No operation	set=1 nop=0

3	intr3	Set software interrupt 3, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
2	intr2	Set software interrupt 2, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
1	intr1	Set software interrupt 1, to MPU. set: Set interrupt nop: No operation	set=1 nop=0
0	intr0	Set software interrupt 0, to MPU. set: Set interrupt nop: No operation	set=1 nop=0

25.29.41 r_mpu_intr

Address	0xa4
Default	
Type	Read
Description	This is used to check which interrupts are set to MPU (unmasked).

Bit(s)	Name	Description	Value
31	other_spu_intr15	software interrupt 15 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
30	other_spu_intr14	software interrupt 14 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
29	other_spu_intr13	software interrupt 13 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
28	other_spu_intr12	software interrupt 12 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
27	other_spu_intr11	software interrupt 11 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
26	other_spu_intr10	software interrupt 10 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
25	other_spu_intr9	software interrupt 9 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
24	other_spu_intr8	software interrupt 8 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
23	other_spu_intr7	software interrupt 7 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
22	other_spu_intr6	software interrupt 6 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
21	other_spu_intr5	software interrupt 5 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
20	other_spu_intr4	software interrupt 4 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

19	other_spu_intr3	software interrupt 3 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
18	other_spu_intr2	software interrupt 2 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
17	other_spu_intr1	software interrupt 1 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
16	other_spu_intr0	software interrupt 0 from the other SPU to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
15	intr15	software interrupt 15, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
14	intr14	software interrupt 14, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
13	intr13	software interrupt 13, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
12	intr12	software interrupt 12, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
11	intr11	software interrupt 11, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
10	intr10	software interrupt 10, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
9	intr9	software interrupt 9, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
8	intr8	software interrupt 8, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	intr7	software interrupt 7, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	intr6	software interrupt 6, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	intr5	software interrupt 5, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

4	intr4	software interrupt 4, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	intr3	software interrupt 3, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	intr2	software interrupt 2, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	intr1	software interrupt 1, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	intr0	software interrupt 0, to MPU. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.30 iop_timer_grp

Instance	Base Address
iop_timer_grp0	0xb0020a80
iop_timer_grp1	0xb0020b00
iop_timer_grp2	0xb0020b80
iop_timer_grp3	0xb0020c00

25.30.1 rw_cfg

Address	0x0
Default	0x00000002
Type	Read/Write
Description	Timer group configuration.

Bit(s)	Name	Description	Value
18-11:8	clk_div	Divisor for predividing the I/O Processor system clock. The predevided clock will have a period length equal to (clk_div * system clock period). clk_div must be an even number (bit 0 of clk_div is always ignored). If clk_div equals 0 the predevided clock will have a period length of (256 * system clock period). The divided clock can be used as clock source for a Timer when rw_tmr_cfg.clk_src is set to div_clk200.	
10-3:8	clk_gen_div	Divisor for predividing the output from the clock generator. The predevided clock will have a period length equal to (clk_gen_div * clock generator period). If clk_gen_div equals 0 the predevided clock will have a period length of (256 * clock generator period). The divided clock can be used as clock source for a Timer when rw_tmr_cfg.clk_src is set to div_clk_gen.	
2-1:2	trig	Select if the clock source of the clock generator should be triggered when it is high or on positive or negative edge. trig is only valid when clk_src equals ext. hi: When clock source is high pos: Positive edge of the clock source neg: Negative edge of the clock source pos_neg: Both positive and negative edge of the clock source	hi=0 pos=1 neg=2 pos_neg=3
0	clk_src	Selects which clock should be used for the countdown in the clock generator. clk200: The internal I/O Processor clock (200 MHz) ext: The external clock is selected by the Switch	clk200=0 ext=1

25.30.2 rw_half_period

Address	0x4
Default	
Type	Read/Write
Description	The generated clock from the clock generator has two different half periods. The occurrence quota of the two half periods are described by quota_hi and quota_lo.

Bit(s)	Name	Description	Value
30	quota_hi_sel	Select which half period should have the highest quota, the half period with length rw_half_period_len or rw_half_period_len+1. short_period: The short period has the highest occurrence long_period: The long period has the highest occurrence	short_period=0 long_period=1
29-15:15	quota_hi	The highest occurrence quota.	
14-0:15	quota_lo	The lowest occurrence quota.	

25.30.3 rw_half_period_len

Address	0x8
Default	
Type	Read/Write
Description	Half period length. The generated clock from the clock generator consists of two different half periods. rw_half_period_len contains the length of the smaller half period. The other half period has the length rw_half_period_len+1. The length is measured by the number of system clock cycles. The clock generator will be reset when writing to rw_half_period_len, i.e. its counter will immediately be updated with the new value of rw_half_period_len.

25.30.4 rw_tmr_cfg

Address	0xc, 0x10, 0x14, 0x18
Default	0x00018000, 0x0001a900, 0x0001d200, 0x0001fb00
Type	Read/Write
Description	Configures a timer. Each timer has its own configuration register.

Bit(s)	Name	Description	Value
17	rst_at_en_strb	Select if the enable strobe will reset the timer counter, or not. no: The counter is not reset by the enable strobe yes: The counter is reset by the enable strobe	no=0 yes=1
16	dis_only_by_reg	Select if the timer can only be disabled by rw_cmd.dis and not by the strobe from other timers. no: The timer can be disabled by other timers or by rw_cmd.en yes: The timer is only disabled by rw_cmd.en	no=0 yes=1
15	en_only_by_reg	Select if the timer can only be enabled by rw_cmd.en and not by the strobe from other timers. no: The timer can be enabled by other timers or by rw_cmd.en yes: The timer is only enabled by rw_cmd.en	no=0 yes=1
14-13:2	dis_by_tmr	Select which timer strobe within the same group that is used to disable the current timer. The timer can not disable itself. If the value of dis_by_tmr is the same as the number of the current timer, it will use an external signal from the switch (e.g. strobe from a Trigger group). dis_by_tmr is ignored when dis_only_by_reg equals yes.	
12-11:2	en_by_tmr	Select which timer strobe within the same group that is used to enable the current timer. The timer can not enable itself. If the value of en_by_tmr is the same as the number of the current timer, it will use an external signal from the switch (e.g. strobe from a Trigger group). en_by_tmr is ignored when en_only_by_reg equals yes.	
10	inv	Decide if the output strobe should be inverted or not. no: Normal yes: Inverted	no=0 yes=1

9-8:2	active_on_tmr	The active period of the output signal from the timer can be controlled by the output signal from another timer within the same Timer group, selected by active_on_tmr. This is achieved by combining the output signals of the timers together using an AND-operation. The result of this operation is available as one of the output strobes from the Timer Group.	
7	out_mode	Output strobe mode. When the timer has counted down to zero, the output strobe can either be a pulse (the strobe is high during one I/O Processor clock cycle) or it will toggle the output value. pulse: Pulse mode toggle: Toggle mode	pulse=0 toggle=1
6-5:2	run_mode	Enable and disable mode. stop: Stop timer and reset it after it is disabled pause: Pause, i.e. the timer is not reset when disabled complete: Complete the current cycle then stop once: Run once then stop	stop=0 pause=1 complete=2 once=3
4-3:2	strb	Select if the clock source (clk_src) should be triggered when it is high or on positive or negative edge. Note: strb must be set to hi if clk_src equals clk200. hi: High strobe pos: Positive edge of strobe neg: Negative edge of strobe pos_neg: Both positive and negative edge of strobe	hi=0 pos=1 neg=2 pos_neg=3

2-0:3	clk_src	Selects which clock source should be used for timer countdown. When the timer is disabled it will only count down when writing to rw_cmd.strb. The divisor for the divided I/O Processor clock and the divided clock generator clock is found in rw_cfg.clk_div and rw_cfg.clk_gen_div respectively. If tmr is selected, the Timer used as clock source must be the previous one in the timer-ring, i.e Timer 0 can have Timer 3 as clock source, Timer 1 can have Timer 0 as clock source, Timer 2 can have Timer 1 as clock source, Timer 3 can have Timer 2 as clock source clk200: I/O Processor clock div_clk200: Predivided I/O Processor clock clk_gen: Clock generator div_clk_gen: Predevided clock generator tmr: Timer	clk200=0 div_clk200=1 clk_gen=2 div_clk_gen=3 tmr=4

25.30.5 rw_tmr_len

Address	0x2c, 0x30, 0x34, 0x38
Default	0x00000000, 0x00000000, 0x00000000, 0x00000000
Type	Read/Write
Description	Set the timer length. The timer can count up to 2^16 cycles, i.e. the timer length is between 5ns to ~300us, when using a 200MHz clock as the clock source. Each timer has its own length register.

Bit(s)	Name	Description	Value
15-0:16	val	The timer counts down from val to 0 before changing the output strobe. In other words the timer will change its output strobe every (val + 1) cycles of clk_src. rs_tmr_cnt will immediately be updated with the new value of val, if a value is written to val when the timer is disabled. Otherwise, if the timer is enabled when a value is written to val, the rs_tmr_cnt will not be updated with the new value until after the counter has counted down to zero.	

25.30.6 rw_cmd

Address	0x3c
Default	
Type	Read/Write
Description	Timer commands for the whole group. Each bit in a field represents one timer.

Bit(s)	Name	Description	Value
15-12:4	strb	Strobe one or more timers. The strobe will count down r_tmr_cnt one step. The timer will count down even if the timer is disabled.	
11-8:4	dis	Disable one or more timers.	
7-4:4	en	Enable one or more timers.	
3-0:4	rst	Reset one or more timers. The Timer counter will then be set to rw_tmr_len and the output will be set to zero. The Timer will also be disabled unless the corresponding bit in en is set at the same time as resetting the Timer.	

25.30.7 r_clk_gen_cnt

Address	0x40
Default	
Type	Read
Description	The current value of the clock generator counter. The clock generator will change its output strobe when r_clk_gen_cnt reaches zero.

25.30.8 rs_tmr_cnt/r_tmr_cnt

Address	0x44, 0x4c, 0x54, 0x5c
Default	
Type	Read with side effects/Read
Description	The current value of the timer counter.

Bit(s)	Name	Description	Value
15-0:16	val	The timer will generate an interrupt or change its output strobe when val reaches zero. When reading rs_tmr_cnt, val will be set to the value of the timer length register.	

25.30.9 rw_intr_mask

Address	0x64
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts from the timer. An interrupt is generated when the timer has counted down to zero. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	tmr3	Enable/disable tmr3 interrupt. Interrupt from timer 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	tmr2	Enable/disable tmr2 interrupt. Interrupt from timer 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	tmr1	Enable/disable tmr1 interrupt. Interrupt from timer 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	tmr0	Enable/disable tmr0 interrupt. Interrupt from timer 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.30.10 rw_ack_intr

Address	0x68
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts from the timer. An interrupt is generated when the timer has counted down to zero.

Bit(s)	Name	Description	Value
3	tmr3	Acknowledge tmr3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	tmr2	Acknowledge tmr2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	tmr1	Acknowledge tmr1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	tmr0	Acknowledge tmr0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.30.11 r_intr

Address	0x6c
Default	
Type	Read
Description	Interrupts before the mask. Interrupts from the timer. An interrupt is generated when the timer has counted down to zero. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	tmr3	Interrupt tmr3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	tmr2	Interrupt tmr2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	tmr1	Interrupt tmr1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	tmr0	Interrupt tmr0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.30.12 r_masked_intr

Address	0x70
Default	
Type	Read
Description	Interrupts after the mask. Interrupts from the timer. An interrupt is generated when the timer has counted down to zero. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	tmr3	Interrupt tmr3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	tmr2	Interrupt tmr2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	tmr1	Interrupt tmr1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	tmr0	Interrupt tmr0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.31 iop_trigger_grp

Instance	Base Address
iop_trigger_grp0	0xb0020140
iop_trigger_grp1	0xb0020180
iop_trigger_grp2	0xb00201c0
iop_trigger_grp3	0xb0020200
iop_trigger_grp4	0xb0020240
iop_trigger_grp5	0xb0020280
iop_trigger_grp6	0xb00202c0
iop_trigger_grp7	0xb0020300

25.31.1 rw_cfg

Address	0x0, 0x4, 0x8, 0xc
Default	0x00000000, 0x00000000, 0x00000000, 0x00000000
Type	Read/Write
Description	Configures a Trigger. Each Trigger has its own configuration register.

Bit(s)	Name	Description	Value
7	dis_only_by_reg	Select if the Trigger can only be disabled by rw_cmd.dis and not by an external strobe. no: The Trigger can be disabled by external strobes or by rw_cmd.dis yes: The Trigger can only be enabled by rw_cmd.dis	no=0 yes=1
6	en_only_by_reg	Select if the Trigger can only be enabled by rw_cmd.en and not by an external strobe. no: The Trigger can be enabled by other timers or by rw_cmd.en yes: The Trigger is only enabled by rw_cmd.en	no=0 yes=1

5-3:3	trig	Triggering condition. Some configurations of trig will make the Trigger trigger if the signal was high (or low) just after the Trigger was enabled. off: Never trigger rise: Trigger on rising edge fall: Trigger on falling edge rise_fall: Trigger on both rising and falling edges rise_fall_lo: Trigger on both edges or if in=0 when enabled rise_hi: Trigger on rising edge or if in=1 when enabled fall_lo: Trigger on falling edge or if in=0 when enabled rise_fall_hi: Trigger on both edges or if in=1 when enabled	off=0 rise=1 fall=2 rise_fall=3 rise_fall_lo=4 rise_hi=5 fall_lo=6 rise_fall_hi=7
2	once	Sense once. Configure if the Trigger shall disable itself automatically or not after an edge detection of the incoming signal. no: The Trigger will not disable itself automatically yes: The Trigger will disable itself after an edge is triggered	no=0 yes=1
1-0:2	action	Output signal action. Configure how the output signal shall act after an edge is triggered. pulse: Pulse output rise: Set output to 1 fall: Set output to 0 toggle: Toggle output	pulse=0 rise=1 fall=2 toggle=3

25.31.2 rw_cmd

Address	0x10
Default	
Type	Read/Write
Description	Trigger commands. Note: Setting a bit in dis and the corresponding bit in en at the same time will disable the Trigger if it is currently enabled and enable it if it is currently disabled.

Bit(s)	Name	Description	Value
7-4:4	en	Writing to en will enable one or more Triggers. Each bit represents one Trigger in this group.	
3-0:4	dis	Writing to dis will disable one or more Triggers. Each bit represents one Trigger in this group.	

25.31.3 rw_intr_mask

Address	0x14
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts from the Trigger. Depending on the configured triggering condition, an interrupt is generated when a rising or falling edge of the incoming signal is detected. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	trig3	Enable/disable trig3 interrupt. Interrupt from Trigger 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	trig2	Enable/disable trig2 interrupt. Interrupt from Trigger 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	trig1	Enable/disable trig1 interrupt. Interrupt from Trigger 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	trig0	Enable/disable trig0 interrupt. Interrupt from Trigger 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.31.4 rw_ack_intr

Address	0x18
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts from the Trigger. Depending on the configured triggering condition, an interrupt is generated when a rising or falling edge of the incoming signal is detected.

Bit(s)	Name	Description	Value
3	trig3	Acknowledge trig3 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	trig2	Acknowledge trig2 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	trig1	Acknowledge trig1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	trig0	Acknowledge trig0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.31.5 r_intr

Address	0x1c
Default	
Type	Read
Description	Interrupts before the mask. Interrupts from the Trigger. Depending on the configured triggering condition, an interrupt is generated when a rising or falling edge of the incoming signal is detected. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	trig3	Interrupt trig3 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	trig2	Interrupt trig2 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	trig1	Interrupt trig1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	trig0	Interrupt trig0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.31.6 r_masked_intr

Address	0x20
Default	
Type	Read
Description	Interrupts after the mask. Interrupts from the Trigger. Depending on the configured triggering condition, an interrupt is generated when a rising or falling edge of the incoming signal is detected. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	trig3	Interrupt trig3 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	trig2	Interrupt trig2 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	trig1	Interrupt trig1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	trig0	Interrupt trig0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.32 iop_scrc_in

Instance	Base Address
iop_scrc_in0	0xb0020880
iop_scrc_in1	0xb0020900

25.32.1 rw_cfg

Address	0x0
Default	0x00000000
Type	Read/Write
Description	Configures the Serial CRC

Bit(s)	Name	Description	Value
1-0:2	trig	Select if the strobe for serial input should be triggered when it is high or on positive or negative edge. hi: High strobe pos: Positive edge of strobe neg: Negative edge of strobe pos_neg: Both positive and negative edge of strobe	hi=0 pos=1 neg=2 pos_neg=3

25.32.2 rw_ctrl

Address	0x4
Default	0x00000000
Type	Read/Write
Description	Serial CRC control register.

Bit(s)	Name	Description	Value
0	dif_in_en	Enable or disable the serial data interface for incoming data. no: Disabled yes: Enabled	no=0 yes=1

25.32.3 r_stat

Address	0x8
Default	
Type	Read
Description	Status register.

Bit(s)	Name	Description	Value
0	err	Status bit from the serial CRC. Checks if rs_computed_crc is not equal rw_correct_crc. no: No CRC error yes: CRC error	no=0 yes=1

25.32.4 rw_init_crc

Address	0xc
Default	0x00000000
Type	Read/Write
Description	Initial shift register value for the CRC, e.g. X^31 + X^22 + X^16 + X^15 + X^11 + X^5 + X^4 + X^3 + 1 = 0x80418839 A write to this register reloads r_computed_crc with rw_init_crc.

25.32.5 rs_computed_crc/r_computed_crc

Address	0x10/0x14
Default	0x00000000
Type	Read with side effects/Read
Description	The contents of the CRC shift register. A read from rs_computed_crc reloads rs_computed_crc with rw_init_crc.

25.32.6 rw_crc

Address	0x18
Default	
Type	Read/Write
Description	The CRC generator polynomial. A term xˆn in the generator polynomial will result in a 1 at bit position n-1 in rw_crc, e.g. xˆ16 + xˆ12 + xˆ5 + 1 = 0x00008810 Note that the representation of the generator polynomial does not contain the last coefficient (+1). A write to this register reloads rw_crc with rw_init_crc.

25.32.7 rw_correct_crc

Address	0x1c
Default	
Type	Read/Write
Description	Set the correct CRC value. If the CRC is correct, r_computed_crc shall have the value of rw_correct_crc, after the data stream (including the CRC) has been received, otherwise it is a CRC error.

25.32.8 rw_wr1bit

Address	0x20
Default	
Type	Read/Write
Description	Generates a serial strobe to the serial CRC. Writes data and marks it with last if set in last.

Bit(s)	Name	Description	Value
3-2:2	last	Last mark to set on this data bit. no: No last is set yes: Last is set dif_in: Last is sampled from dif_in last signal	no=0 yes=1 dif_in=2
1-0:2	data	Value for this data bit. set0: Data bit is zero set1: Data bit is one dif_in: Data is sampled from dif_in data signal	set0=0 set1=1 dif_in=2

25.33 iop_scrc_out

Instance	Base Address
iop_scrc_out0	0xb0020980
iop_scrc_out1	0xb0020a00

25.33.1 rw_cfg

Address	0x0
Default	0x00000000
Type	Read/Write
Description	Configures the serial CRC. The serial CRC will be reset when writing to this register. I.e., the internal FIFO will then become empty and r_computed_crc is set to either rw_init_crc or to the inverted bits of rw_init_crc depending on the value of inv_crc.

Bit(s)	Name	Description	Value
2	inv_crc	Select if each bit of the computed CRC should be inverted or not. no: CRC bits are not inverted yes: CRC bits are inverted	no=0 yes=1
1-0:2	trig	Select if the strobe for serial output should be triggered when it is high or on positive or negative edge. hi: High strobe pos: Positive edge of strobe neg: Negative edge of strobe pos_neg: Both positive and negative edge of strobe	hi=0 pos=1 neg=2 pos_neg=3

25.33.2 rw_ctrl

Address	0x4
Default	0x00000000
Type	Read/Write
Description	Serial CRC control register.

Bit(s)	Name	Description	Value
1	out_src	Select the source of the DIF output. When the CRC is transmitted, the first CRC bit sent is the most significant bit of r_computed_crc (depends on the length of the generator polynomial). The last CRC bit sent is bit 0 of r_computed_crc. data: Output data from rw_data.val crc: Output CRC	data=0 crc=1
0	strb_src	Select the source of the CRC strobe. reg: Strobe CRC when data is written to rw_data.val dif: Use the strobe from DIF as CRC strobe	reg=0 dif=1

25.33.3 rw_init_crc

Address	0x8
Default	0x00000000
Type	Read/Write
Description	Initial shift register value for the CRC. When writing to this register, the CRC shift register for outgoing data will be reset. r_computed_crc is then set to either rw_init_crc or to the inverted bits of rw_init_crc depending on the value of rw_cfg.inv_crc.

25.33.4 rw_crc

Address	0xc
Default	0x00000000
Type	Read/Write
Description	The CRC generator polynomial. A term xˆn in the generator polynomial will result in a 1 at bit position n-1 in rw_crc, e.g. xˆ16 + xˆ12 + xˆ5 + 1 = 0x00008810 Note that the representation of the generator polynomial does not contain the last coefficient (+1). A write to this register reloads the CRC shift register. r_computed_crc is then set to either rw_init_crc or to the inverted bits of rw_init_crc depending on the value of rw_cfg.inv_crc.

25.33.5 rw_data

Address	0x10
Default	0x00000000
Type	Read/Write
Description	Write data to the serial CRC. Only the write strobe of rw_data is used (the value of rw_data.val is ignored), if rw_ctrl.out_src equals crc and rw_ctrl.strb_src equals reg.

Bit(s)	Name	Description	Value
0	val	Data bit.	

25.33.6 r_computed_crc

Address	0x14
Default	
Type	Read
Description	The contents of the CRC shift register.

25.34 iop_version

Instance	Base Address
iop_version	0xb0020000

25.34.1 r_version

Address	0x0
Default	
Type	Read
Description	I/O Processor version register.

Bit(s)	Name	Description	Value
7-0:8	nr	I/O Processor version number. v1_0: Version 1.0	v1_0=1

25.35 marb_bp

Instance	Base Address
marb_bp0	0xb003e240
marb_bp1	0xb003e280
marb_bp2	0xb003e2c0
marb_bp3	0xb003e300

25.35.1 rw_first_addr

Address	0x0
Default	
Type	Read/Write
Description	First address triggering the breakpoint. For the chosen access types and clients, the breakpoint triggers when addr+size > rw_first_addr and if wrap in the rw_options register is set to no, addr <= rw_last_addr must also be true.

25.35.2 rw_last_addr

Address	0x4
Default	
Type	Read/Write
Description	Last address triggering the breakpoint. For the chosen access types and clients, the breakpoint triggers when addr <= rw_last_addr and if wrap in the rw_options register is set to no, addr+size > rw_first_addr must also be true.

25.35.3 rw_op

Address	0x8
Default	0x00000000
Type	Read/Write
Description	Operations selection for the breakpoint.

Bit(s)	Name	Description	Value
7	us_pri_wr	Turn breaking on unsnooped priority writes on/off. yes: Break on unsnooped priority writes no: Do not break on unsnooped priority writes	yes=1 no=0
6	us_rd_excl	Turn breaking on unsnooped read exclusives on/off. yes: Break on unsnooped read exclusives no: Do not break on unsnooped read exclusives	yes=1 no=0
5	us_wr	Turn breaking on unsnooped plain writes on/off. yes: Break on unsnooped plain writes no: Do not break on unsnooped plain writes	yes=1 no=0
4	us_rd	Turn breaking on unsnooped plain reads on/off. yes: Break on unsnooped plain reads no: Do not break on unsnooped plain reads	yes=1 no=0
3	pri_wr	Turn breaking on priority writes on/off. yes: Break on priority writes no: Do not break on priority writes	yes=1 no=0
2	rd_excl	Turn breaking on read exclusives on/off. yes: Break on read exclusives no: Do not break on read exclusives	yes=1 no=0
1	wr	Turn breaking on plain writes on/off. yes: Break on plain writes no: Do not break on plain writes	yes=1 no=0
0	rd	Turn breaking on plain reads on/off. yes: Break on plain reads no: Do not break on plain reads	yes=1 no=0

25.35.4 rw_clients

Address	0xc
Default	
Type	Read/Write
Description	Chooses which clients the breakpoint listens to.

Bit(s)	Name	Description	Value
13	slave	Turn breaking on accesses from slave on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0
12	iop	Turn breaking on accesses from iop on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0
11	cpud	Turn breaking on accesses from cpud on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0
10	cpui	Turn breaking on accesses from cpui on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0
9	dma9	Turn breaking on accesses from dma9 on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0
8	dma8	Turn breaking on accesses from dma8 on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0
7	dma7	Turn breaking on accesses from dma7 on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0
6	dma6	Turn breaking on accesses from dma6 on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0
5	dma5	Turn breaking on accesses from dma5 on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0
4	dma4	Turn breaking on accesses from dma4 on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0
3	dma3	Turn breaking on accesses from dma3 on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0
2	dma2	Turn breaking on accesses from dma2 on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0

1	dma1	Turn breaking on accesses from dma1 on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0
0	dma0	Turn breaking on accesses from dma0 on/off. yes: Listen to client no: Do not listen to client	yes=1 no=0

25.35.5 rw_options

Address	0x10
Default	0x00000000
Type	Read/Write
Description	Options affecting the behavior of the breakpoint.

Bit(s)	Name	Description	Value
0	wrap	Selects if the breakpoint address range should be [rw_first_addr, rw_last_addr] (wrap == no) or [rw_first_addr, 0xffffffff, [0, rw_last_addr] (wrap == yes). yes: Address range wraps from 0xffffffff to 0 no: Address range does not wrap	yes=1 no=0

25.35.6 r_brk_addr

Address	0x14
Default	
Type	Read
Description	The address of the first access that triggered the breakpoint after it has been enabled or acknowledged through rw_ack.

25.35.7 r_brk_op

Address	0x18
Default	
Type	Read
Description	The operation of the first access that triggered the breakpoint after it has been enabled or acknowledge through rw_ack.

Bit(s)	Name	Description	Value
7	us_pri_wr	Breakpoint triggered by unsnooped priority writes. yes: Triggered on unsnooped priority writes no: Did not trigger on unsnooped priority writes	yes=1 no=0
6	us_rd_excl	Breakpoint triggered by unsnooped read exclusives. yes: Triggered on unsnooped read exclusives no: Did not trigger on unsnooped read exclusives	yes=1 no=0
5	us_wr	Breakpoint triggered by unsnooped plain writes. yes: Triggered on unsnooped plain writes no: Did not trigger on unsnooped plain writes	yes=1 no=0
4	us_rd	Breakpoint triggered by unsnooped plain reads. yes: Triggered on unsnooped plain reads no: Did not trigger on unsnooped plain reads	yes=1 no=0
3	pri_wr	Breakpoint triggered by priority writes. yes: Triggered on priority writes no: Did not trigger on priority writes	yes=1 no=0
2	rd_excl	Breakpoint triggered by read exclusives. yes: Triggered on read exclusives no: Did not trigger on read exclusives	yes=1 no=0
1	wr	Breakpoint triggered by plain writes. yes: Triggered on plain writes no: Did not trigger on plain writes	yes=1 no=0
0	rd	Breakpoint triggered by plain reads. yes: Triggered on plain reads no: Did not trigger on plain reads	yes=1 no=0

25.35.8 r_brk_clients

Address	0x1c
Default	
Type	Read
Description	All clients that have triggered the breakpoint since it was last acknowledged through rw_ack.

Bit(s)	Name	Description	Value
13	slave	Has slave triggered this breakpoint? yes: slave has triggered breakpoint no: slave has not triggered breakpoint	yes=1 no=0
12	iop	Has iop triggered this breakpoint? yes: iop has triggered breakpoint no: iop has not triggered breakpoint	yes=1 no=0
11	cpud	Has cpud triggered this breakpoint? yes: cpud has triggered breakpoint no: cpud has not triggered breakpoint	yes=1 no=0
10	cpui	Has cpui triggered this breakpoint? yes: cpui has triggered breakpoint no: cpui has not triggered breakpoint	yes=1 no=0
9	dma9	Has dma9 triggered this breakpoint? yes: dma9 has triggered breakpoint no: dma9 has not triggered breakpoint	yes=1 no=0
8	dma8	Has dma8 triggered this breakpoint? yes: dma8 has triggered breakpoint no: dma8 has not triggered breakpoint	yes=1 no=0
7	dma7	Has dma7 triggered this breakpoint? yes: dma7 has triggered breakpoint no: dma7 has not triggered breakpoint	yes=1 no=0
6	dma6	Has dma6 triggered this breakpoint? yes: dma6 has triggered breakpoint no: dma6 has not triggered breakpoint	yes=1 no=0
5	dma5	Has dma5 triggered this breakpoint? yes: dma5 has triggered breakpoint no: dma5 has not triggered breakpoint	yes=1 no=0
4	dma4	Has dma4 triggered this breakpoint? yes: dma4 has triggered breakpoint no: dma4 has not triggered breakpoint	yes=1 no=0
3	dma3	Has dma3 triggered this breakpoint? yes: dma3 has triggered breakpoint no: dma3 has not triggered breakpoint	yes=1 no=0
2	dma2	Has dma2 triggered this breakpoint? yes: dma2 has triggered breakpoint no: dma2 has not triggered breakpoint	yes=1 no=0

1	dma1	Has dma1 triggered this breakpoint? yes: dma1 has triggered breakpoint no: dma1 has not triggered breakpoint	yes=1 no=0
0	dma0	Has dma0 triggered this breakpoint? yes: dma0 has triggered breakpoint no: dma0 has not triggered breakpoint	yes=1 no=0

25.35.9 r_brk_first_client

Address	0x20
Default	
Type	Read
Description	The first client that triggered the breakpoint after it was last acknowledged through rw_ack.

Bit(s)	Name	Description	Value
13	slave	Did slave trigger this breakpoint first? yes: slave first triggered breakpoint no: slave first not triggered breakpoint	yes=1 no=0
12	iop	Did iop trigger this breakpoint first? yes: iop first triggered breakpoint no: iop first not triggered breakpoint	yes=1 no=0
11	cpud	Did cpud trigger this breakpoint first? yes: cpud first triggered breakpoint no: cpud first not triggered breakpoint	yes=1 no=0
10	cpui	Did cpui trigger this breakpoint first? yes: cpui first triggered breakpoint no: cpui first not triggered breakpoint	yes=1 no=0
9	dma9	Did dma9 trigger this breakpoint first? yes: dma9 first triggered breakpoint no: dma9 first not triggered breakpoint	yes=1 no=0
8	dma8	Did dma8 trigger this breakpoint first? yes: dma8 first triggered breakpoint no: dma8 first not triggered breakpoint	yes=1 no=0
7	dma7	Did dma7 trigger this breakpoint first? yes: dma7 first triggered breakpoint no: dma7 first not triggered breakpoint	yes=1 no=0
6	dma6	Did dma6 trigger this breakpoint first? yes: dma6 first triggered breakpoint no: dma6 first not triggered breakpoint	yes=1 no=0
5	dma5	Did dma5 trigger this breakpoint first? yes: dma5 first triggered breakpoint no: dma5 first not triggered breakpoint	yes=1 no=0
4	dma4	Did dma4 trigger this breakpoint first? yes: dma4 first triggered breakpoint no: dma4 first not triggered breakpoint	yes=1 no=0
3	dma3	Did dma3 trigger this breakpoint first? yes: dma3 first triggered breakpoint no: dma3 first not triggered breakpoint	yes=1 no=0
2	dma2	Did dma2 trigger this breakpoint first? yes: dma2 first triggered breakpoint no: dma2 first not triggered breakpoint	yes=1 no=0

1	dma1	Did dma1 trigger this breakpoint first? yes: dma1 first triggered breakpoint no: dma1 first not triggered breakpoint	yes=1 no=0
0	dma0	Did dma0 trigger this breakpoint first? yes: dma0 first triggered breakpoint no: dma0 first not triggered breakpoint	yes=1 no=0

25.35.10 r_brk_size

Address	0x24
Default	
Type	Read
Description	The size of the first access that triggered the breakpoint after it was last acknowledged through rw_ack.

25.35.11 rw_ack

Address	0x28
Default	
Type	Read/Write
Description	Writing anything to this register will acknowledge this breakpoint if it has triggered. I.e make it ”untriggered”. This has the same effect as writing yes to the field in marb.rw_ack_intr referring to this breakpoint.

25.36 marb

Instance	Base Address
marb	0xb003e000

25.36.1 rw_int_slots

Address	0x0, 0x4, 0x8, 0xc, 0x10, 0x14, 0x18, 0x1c, 0x20, 0x24, 0x28, 0x2c, 0x30, 0x34, 0x38, 0x3c, 0x40, 0x44, 0x48, 0x4c, 0x50, 0x54, 0x58, 0x5c, 0x60, 0x64, 0x68, 0x6c, 0x70, 0x74, 0x78, 0x7c, 0x80, 0x84, 0x88, 0x8c, 0x90, 0x94, 0x98, 0x9c, 0xa0, 0xa4, 0xa8, 0xac, 0xb0, 0xb4, 0xb8, 0xbc, 0xc0, 0xc4, 0xc8, 0xcc, 0xd0, 0xd4, 0xd8, 0xdc, 0xe0, 0xe4, 0xe8, 0xec, 0xf0, 0xf4, 0xf8, 0xfc
Default	0x00000000, 0x00000000
Type	Read/Write
Description	Slot allocation vector for Internal 128KB RAM and 8KB ROM.

Bit(s)	Name	Description	Value
3-0:4	owner	Slot owner. dma0: Client dma0 dma1: Client dma1 dma2: Client dma2 dma3: Client dma3 dma4: Client dma4 dma5: Client dma5 dma6: Client dma6 dma7: Client dma7 dma8: Client dma8 dma9: Client dma9 cpui: Client cpui cpud: Client cpud iop: Client iop slave: Client slave	dma0=0 dma1=1 dma2=2 dma3=3 dma4=4 dma5=5 dma6=6 dma7=7 dma8=8 dma9=9 cpui=10 cpud=11 iop=12 slave=13

25.36.2 rw_ext_slots

Address	0x100, 0x104, 0x108, 0x10c, 0x110, 0x114, 0x118, 0x11c, 0x120, 0x124, 0x128, 0x12c, 0x130, 0x134, 0x138, 0x13c, 0x140, 0x144, 0x148, 0x14c, 0x150, 0x154, 0x158, 0x15c, 0x160, 0x164, 0x168, 0x16c, 0x170, 0x174, 0x178, 0x17c, 0x180, 0x184, 0x188, 0x18c, 0x190, 0x194, 0x198, 0x19c, 0x1a0, 0x1a4, 0x1a8, 0x1ac, 0x1b0, 0x1b4, 0x1b8, 0x1bc, 0x1c0, 0x1c4, 0x1c8, 0x1cc, 0x1d0, 0x1d4, 0x1d8, 0x1dc, 0x1e0, 0x1e4, 0x1e8, 0x1ec, 0x1f0, 0x1f4, 0x1f8, 0x1fc
Default	0x00000000, 0x00000000
Type	Read/Write
Description	Slot allocation vector for External memory.

Bit(s)	Name	Description	Value
3-0:4	owner	Slot owner. dma0: Client dma0 dma1: Client dma1 dma2: Client dma2 dma3: Client dma3 dma4: Client dma4 dma5: Client dma5 dma6: Client dma6 dma7: Client dma7 dma8: Client dma8 dma9: Client dma9 cpui: Client cpui cpud: Client cpud iop: Client iop slave: Client slave	dma0=0 dma1=1 dma2=2 dma3=3 dma4=4 dma5=5 dma6=6 dma7=7 dma8=8 dma9=9 cpui=10 cpud=11 iop=12 slave=13

25.36.3 rw_regs_slots

Address	0x200, 0x204, 0x208, 0x20c
Default	0x00000000, 0x00000000, 0x00000000, 0x00000000
Type	Read/Write
Description	Slot allocation vector for Mode registers.

Bit(s)	Name	Description	Value
3-0:4	owner	Slot owner. cpud: Client cpud iop: Client iop slave: Client slave	cpud=11 iop=12 slave=13

25.36.4 rw_intr_mask

Address	0x210
Default	0x00000000
Type	Read/Write
Description	Interrupt mask.

Bit(s)	Name	Description	Value
3	bp3	Enable/disable breakpoint 3. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	bp2	Enable/disable breakpoint 2. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	bp1	Enable/disable breakpoint 1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	bp0	Enable/disable breakpoint 0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.36.5 rw_ack_intr

Address	0x214
Default	
Type	Read/Write
Description	Acknowledge interrupts.

Bit(s)	Name	Description	Value
3	bp3	Acknowledge breakpoint 3. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	bp2	Acknowledge breakpoint 2. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	bp1	Acknowledge breakpoint 1. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	bp0	Acknowledge breakpoint 0. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.36.6 r_intr

Address	0x218
Default	
Type	Read
Description	Interrupts before mask. Can be used to tell which breakpoints have triggered.

Bit(s)	Name	Description	Value
3	bp3	Status before mask of breakpoint 3. yes: Interrupt is active (bp has triggered) no: Interrupt is inactive (bp has not triggered)	yes=1 no=0
2	bp2	Status before mask of breakpoint 2. yes: Interrupt is active (bp has triggered) no: Interrupt is inactive (bp has not triggered)	yes=1 no=0
1	bp1	Status before mask of breakpoint 1. yes: Interrupt is active (bp has triggered) no: Interrupt is inactive (bp has not triggered)	yes=1 no=0
0	bp0	Status before mask of breakpoint 0. yes: Interrupt is active (bp has triggered) no: Interrupt is inactive (bp has not triggered)	yes=1 no=0

25.36.7 r_masked_intr

Address	0x21c
Default	
Type	Read
Description	Interrupts after mask.

Bit(s)	Name	Description	Value
3	bp3	Status after mask of breakpoint 3. yes: Interrupt is active (bp has triggered) no: Interrupt is inactive (bp has not triggered)	yes=1 no=0
2	bp2	Status after mask of breakpoint 2. yes: Interrupt is active (bp has triggered) no: Interrupt is inactive (bp has not triggered)	yes=1 no=0
1	bp1	Status after mask of breakpoint 1. yes: Interrupt is active (bp has triggered) no: Interrupt is inactive (bp has not triggered)	yes=1 no=0
0	bp0	Status after mask of breakpoint 0. yes: Interrupt is active (bp has triggered) no: Interrupt is inactive (bp has not triggered)	yes=1 no=0

25.36.8 rw_stop_mask

Address	0x220
Default	0x00000000
Type	Read/Write
Description	Clients to stop when they trigger a breakpoint.

Bit(s)	Name	Description	Value
13	slave	When on, accesses from client slave will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0
12	iop	When on, accesses from client iop will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0
11	cpud	When on, accesses from client cpud will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0
10	cpui	When on, accesses from client cpui will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0
9	dma9	When on, accesses from client dma9 will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0
8	dma8	When on, accesses from client dma8 will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0

7	dma7	When on, accesses from client dma7 will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0
6	dma6	When on, accesses from client dma6 will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0
5	dma5	When on, accesses from client dma5 will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0
4	dma4	When on, accesses from client dma4 will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0
3	dma3	When on, accesses from client dma3 will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0
2	dma2	When on, accesses from client dma2 will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0
1	dma1	When on, accesses from client dma1 will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0

0	dma0	When on, accesses from client dma0 will be stopped when it triggers a breakpoint. The client will be started again when the breakpoint(s) is/are acknowledged in rw_ack_intr or this field is set to no. yes: Stop client at breakpoint trigger no: Do not stop client at breakpoint trigger	yes=1 no=0

25.36.9 r_stopped

Address	0x224
Default	0x00000000
Type	Read
Description	Clients stopped by triggering breakpoints.

Bit(s)	Name	Description	Value
13	slave	Tells if client slave is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0
12	iop	Tells if client iop is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0
11	cpud	Tells if client cpud is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0
10	cpui	Tells if client cpui is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0
9	dma9	Tells if client dma9 is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0
8	dma8	Tells if client dma8 is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0
7	dma7	Tells if client dma7 is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0
6	dma6	Tells if client dma6 is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0
5	dma5	Tells if client dma5 is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0
4	dma4	Tells if client dma4 is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0
3	dma3	Tells if client dma3 is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0
2	dma2	Tells if client dma2 is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0

1	dma1	Tells if client dma1 is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0
0	dma0	Tells if client dma0 is stopped by a triggered breakpoint or not. yes: Client is stopped no: Client is running	yes=1 no=0

25.36.10 rw_no_snoop

Address	0x340
Default	0x00000000
Type	Read/Write
Description	Make clients use only unsnooped accesses.

Bit(s)	Name	Description	Value
13	slave	When on, accesses from client slave will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0
12	iop	When on, accesses from client iop will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0
11	cpud	When on, accesses from client cpud will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0
10	cpui	When on, accesses from client cpui will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0
9	dma9	When on, accesses from client dma9 will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0
8	dma8	When on, accesses from client dma8 will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0
7	dma7	When on, accesses from client dma7 will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0
6	dma6	When on, accesses from client dma6 will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0
5	dma5	When on, accesses from client dma5 will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0
4	dma4	When on, accesses from client dma4 will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0
3	dma3	When on, accesses from client dma3 will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0
2	dma2	When on, accesses from client dma2 will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0

1	dma1	When on, accesses from client dma1 will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0
0	dma0	When on, accesses from client dma0 will always be unsnooped. yes: Client uses only unsnooped accesses no: Client may uses any type of access	yes=1 no=0

25.36.11 rw_no_snoop_rq

Address	0x344
Default	0x00000000
Type	Read/Write
Description	Make client caches ignore snoop requests.

Bit(s)	Name	Description	Value
11	cpud	When on, snoop requests to client cpud will be ignored. yes: Clients cache ignores snoop requests no: Clients cache obeys snoop requests	yes=1 no=0
10	cpui	When on, snoop requests to client cpui will be ignored. yes: Clients cache ignores snoop requests no: Clients cache obeys snoop requests	yes=1 no=0

25.37 mmu

Bank
1
2

25.37.1 rw_mm_cfg

Support register	s0
Default	
Type	Read/Write
Description	General configuration register.

Bit(s)	Name	Description	Value
19	we	Write error exception enable. Enable this bit to turn on write protection of pages marked with the write enable bit in the TLB. When disabled, the write enable bit in the TLB entry is ignored and can not generate a write error exception. All pages are then write enabled. off: Disable write error exception on: Enable write error exception	off=0 on=1
18	acc	Access violation exception enable. Enable this bit to turn on protection of kernel pages. When disabled, the kernel bit in the TLB entry is ignored and can not generate an access violation exception. All kernel pages may then be referenced in user mode also. off: Disable access violation exception on: Enable access violation exception	off=0 on=1
17	ex	Execute violation exception enable. Enable this bit to turn on protection of pages from code execution. When disabled, the CPU is allowed to execute code from any page and no access violation exceptions are generated. off: Disable execute violation exception on: Enable execute violation exception	off=0 on=1
16	inv	Invalid page exception enable. Enable this bit to turn on the invalid page exception. When enabled, a matching entry in the TLB with the valid bit cleared will generate an invalid page exception. When disabled, an entry with the valid bit cleared will be treated as a miss and thus generate a refill fault. off: Disable invalid page exception on: Enable invalid page exception	off=0 on=1
15	seg_f	Segment seg_f map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1

14	seg_e	Segment seg_e map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
13	seg_d	Segment seg_d map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
12	seg_c	Segment seg_c map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
11	seg_b	Segment seg_b map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
10	seg_a	Segment seg_a map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
9	seg_9	Segment seg_9 map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
8	seg_8	Segment seg_8 map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
7	seg_7	Segment seg_7 map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
6	seg_6	Segment seg_6 map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
5	seg_5	Segment seg_5 map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
4	seg_4	Segment seg_4 map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
3	seg_3	Segment seg_3 map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
2	seg_2	Segment seg_2 map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
1	seg_1	Segment seg_1 map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1
0	seg_0	Segment seg_0 map type select. page: Select page mapping linear: Select linear segment mapping	page=0 linear=1

25.37.2 rw_mm_kbase_lo

Support register	s1
Default	
Type	Read/Write
Description	Kernel segment base low.

Bit(s)	Name	Description	Value
31-28:4	base_7	Kernel segment seg_7 base.	
27-24:4	base_6	Kernel segment seg_6 base.	
23-20:4	base_5	Kernel segment seg_5 base.	
19-16:4	base_4	Kernel segment seg_4 base.	
15-12:4	base_3	Kernel segment seg_3 base.	
11-8:4	base_2	Kernel segment seg_2 base.	
7-4:4	base_1	Kernel segment seg_1 base.	
3-0:4	base_0	Kernel segment seg_0 base.	

25.37.3 rw_mm_kbase_hi

Support register	s2
Default	
Type	Read/Write
Description	Kernel segment base high.

Bit(s)	Name	Description	Value
31-28:4	base_f	Kernel segment seg_f base.	
27-24:4	base_e	Kernel segment seg_e base.	
23-20:4	base_d	Kernel segment seg_d base.	
19-16:4	base_c	Kernel segment seg_c base.	
15-12:4	base_b	Kernel segment seg_b base.	
11-8:4	base_a	Kernel segment seg_a base.	
7-4:4	base_9	Kernel segment seg_9 base.	
3-0:4	base_8	Kernel segment seg_8 base.	

25.37.4 r_mm_cause

Support register	s3
Default	
Type	Read
Description	MMU fault cause. This register is also used for storage of rw_mm_tlb_hi data and the contents of the register will be destroyed when writing to rw_mm_tlb_hi. The r_mm_cause register is updated when a memory management exception occurs, and tells the system what triggered the exception. When rw_mm_tlb_lo is written the pid field and the upper 15 bits of the vpn field are written into the TLB using the idx field in rw_mm_tlb_sel.

Bit(s)	Name	Description	Value
31-13:19	vpn	Virtual Page Number. This field is updated when a memory management exception occurs and holds the vpn for the referenced address that generated the exception.	
9-8:2	op	Operation type. This field is updated when a memory management exception occurs and indicates if the exception was caused by an execute, read, write or flush operation. execute: Execute operation read: Read operation write: Write operation flush: Flush operation	execute=0 read=1 write=2 flush=3
7-0:8	pid	Page ID. This field is updated when a memory management exception occurs and holds the process ID value in the CPU when the exception occurred.	

25.37.5 rw_mm_tlb_sel

Support register	s4
Default	
Type	Read/Write
Description	TLB entry select.

Bit(s)	Name	Description	Value
5-4:2	set	TLB set select. This field selects which set in the TLB to use when rw_mm_tlb_lo and rw_mm_tlb_hi registers are used. At a refill exception the set field is loaded with a random value to select which set to replace. All other exceptions load the set field with a pointer to the set that triggered the exception.	
3-0:4	idx	TLB index select. This field selects which index in the TLB to use when rw_mm_tlb_lo and rw_mm_tlb_hi registers are used. At an exception the idx field is loaded with the four lower bits of the faulting vpn to be used as a pointer to the index that triggered the exception.	

25.37.6 rw_mm_tlb_lo

Support register	s5
Default	
Type	Read/Write
Description	TLB low data. This register is used for reading and writing the lower part of an entry in the TLB. When rw_mm_tlb_lo is read, the pfn, g, v, k, w and x fields in the TLB entry selected by the idx field in rw_mm_tlb_sel will be read. When rw_mm_tlb_lo is written, this value plus the pid field and the 15 most significant bits of the vpn field in the r_mm_cause register are written into the TLB entry selected by the idx field in rw_mm_tlb_sel. The four lower bits of the vpn field are not used but must always match the four lower bits in the idx field in rw_mm_tlb_sel.

Bit(s)	Name	Description	Value
31-13:19	pfn	Physical frame number.	
4	g	Global bit. no: Disable global access yes: Enable global access	no=0 yes=1
3	v	Valid bit. no: Mark as invalid yes: Mark as valid	no=0 yes=1
2	k	Kernel bit. no: Enable user mode permission yes: Disable user mode permission	no=0 yes=1
1	w	Write enable bit. no: Disable write permission yes: Enable write permission	no=0 yes=1
0	x	Execute bit. no: Disable execute permission yes: Enable execute permission	no=0 yes=1

25.37.7 rw_mm_tlb_hi

Support register	s6
Default	
Type	Read/Write
Description	TLB high data. This register is used for reading and writing the higher part of an entry in the TLB. When rw_mm_tlb_hi is read, the pid and vpn fields in the TLB entry selected by the idx field in rw_mm_tlb_sel will be read. When rw_mm_tlb_hi is written, the value is stored in the corresponding fields in r_mm_cause. When rw_mm_tlb_lo is written the fields in r_mm_cause will be written into the TLB. Note that the previous contents of r_mm_cause will be destroyed when writing to this register.

Bit(s)	Name	Description	Value
31-13:19	vpn	Virtual Page Number.	
7-0:8	pid	Page ID.	

25.38 pinmux

Instance	Base Address
pinmux	0xb0038000

25.38.1 rw_pa

Address	0x0
Default	0x00000000
Type	Read/Write
Description	Connection of configurable I/O port **pa**.

Bit(s)	Name	Description	Value
15	hsh7	Connects to external DMA/slave mode handshake signal **hsh7** to pin **pa7**. no: Not connected yes: Connected	no=0 yes=1
14	hsh6	Connects to external DMA/slave mode handshake signal **hsh6** to pin **pa6**. no: Not connected yes: Connected	no=0 yes=1
13	hsh5	Connects to external DMA/slave mode handshake signal **hsh5** to pin **pa5**. no: Not connected yes: Connected	no=0 yes=1
12	hsh4	Connects to external DMA/slave mode handshake signal **hsh4** to pin **pa4**. no: Not connected yes: Connected	no=0 yes=1
11	csp6_n	Connects chip select **csp6_n** to pin **pa3**. no: Not connected yes: Connected	no=0 yes=1
10	csp5_n	Connects chip select **csp5_n** to pin **pa2**. no: Not connected yes: Connected	no=0 yes=1
9	csp3_n	Connects chip select **csp3_n** to pin **pa1**. no: Not connected yes: Connected	no=0 yes=1
8	csp2_n	Connects chip select **csp2_n** to pin **pa0**. no: Not connected yes: Connected	no=0 yes=1

7	pa7	Connects general I/O signal **pa7** to pin **pa7**. no: Not connected yes: Connected	no=0 yes=1
6	pa6	Connects general I/O signal **pa6** to pin **pa6**. no: Not connected yes: Connected	no=0 yes=1
5	pa5	Connects general I/O signal **pa5** to pin **pa5**. no: Not connected yes: Connected	no=0 yes=1
4	pa4	Connects general I/O signal **pa4** to pin **pa4**. no: Not connected yes: Connected	no=0 yes=1
3	pa3	Connects general I/O signal **pa3** to pin **pa3**. no: Not connected yes: Connected	no=0 yes=1
2	pa2	Connects general I/O signal **pa2** to pin **pa2**. no: Not connected yes: Connected	no=0 yes=1
1	pa1	Connects general I/O signal **pa1** to pin **pa1**. no: Not connected yes: Connected	no=0 yes=1
0	pa0	Connects general I/O signal **pa0** to pin **pa0**. no: Not connected yes: Connected	no=0 yes=1

25.38.2 rw_hwprot

Address	0x4
Default	0x00000000
Type	Read/Write
Description	Connection of I/O protocol functions that are implemented as fixed blocks.

Bit(s)	Name	Description	Value
12	timer	Connects the timer output to pin **pc16**. no: Not connected yes: Connected	no=0 yes=1
11	eth1_mgm	Connects Ethernet port 1 transceiver management signals to pins **pe17** - **pe16**. no: Not connected yes: Connected	no=0 yes=1
10	eth1	Connects Ethernet port 1 communication signals to pins **pe15** - **pe0**. no: Not connected yes: Connected	no=0 yes=1
9	ata	Connects common ATA signals to pins **pb15** - **pb0** and **pd14** - **pd8**. no: Not connected yes: Connected	no=0 yes=1
8	ata3	Connects ATA bus 3 signals to pins **pc10** - **pc8** and **pc2** - **pc0**. no: Not connected yes: Connected	no=0 yes=1
7	ata2	Connects ATA bus 2 signals to pins **pc15** - **pc11** and **pc3**. no: Not connected yes: Connected	no=0 yes=1
6	ata1	Connects ATA bus 1 signals to pins **pd4** - **pd0** and **pe17**. no: Not connected yes: Connected	no=0 yes=1
5	ata0	Connects ATA bus 0 signals to pins **pd17** - **pd15** and **pd7** - **pd5**. no: Not connected yes: Connected	no=0 yes=1
4	sser1	Connects synchronous serial port 1 to pins **pd4** - **pd0**. no: Not connected yes: Connected	no=0 yes=1
3	sser0	Connects synchronous serial port 0 to pins **pc16** and **pc3** - **pc0**. no: Not connected yes: Connected	no=0 yes=1

2	ser3	Connects asynchronous serial port 3 to pins **pc15** - **pc12**. no: Not connected yes: Connected	no=0 yes=1
1	ser2	Connects asynchronous serial port 2 to pins **pc11** - **pc8**. no: Not connected yes: Connected	no=0 yes=1
0	ser1	Connects asynchronous serial port 1 to pins **pc7** - **pc4**. no: Not connected yes: Connected	no=0 yes=1

25.38.3 rw_pb_gio

Address	0x8
Default	0x00000000
Type	Read/Write
Description	Connects general I/O port pb to port **pb**.

Bit(s)	Name	Description	Value
17	pb17	Connects general I/O signal **pb17** to pin **pb17**. no: Not connected yes: Connected	no=0 yes=1
16	pb16	Connects general I/O signal **pb16** to pin **pb16**. no: Not connected yes: Connected	no=0 yes=1
15	pb15	Connects general I/O signal **pb15** to pin **pb15**. no: Not connected yes: Connected	no=0 yes=1
14	pb14	Connects general I/O signal **pb14** to pin **pb14**. no: Not connected yes: Connected	no=0 yes=1
13	pb13	Connects general I/O signal **pb13** to pin **pb13**. no: Not connected yes: Connected	no=0 yes=1
12	pb12	Connects general I/O signal **pb12** to pin **pb12**. no: Not connected yes: Connected	no=0 yes=1
11	pb11	Connects general I/O signal **pb11** to pin **pb11**. no: Not connected yes: Connected	no=0 yes=1
10	pb10	Connects general I/O signal **pb10** to pin **pb10**. no: Not connected yes: Connected	no=0 yes=1
9	pb9	Connects general I/O signal **pb9** to pin **pb9**. no: Not connected yes: Connected	no=0 yes=1
8	pb8	Connects general I/O signal **pb8** to pin **pb8**. no: Not connected yes: Connected	no=0 yes=1
7	pb7	Connects general I/O signal **pb7** to pin **pb7**. no: Not connected yes: Connected	no=0 yes=1
6	pb6	Connects general I/O signal **pb6** to pin **pb6**. no: Not connected yes: Connected	no=0 yes=1

5	pb5	Connects general I/O signal **pb5** to pin **pb5**. no: Not connected yes: Connected	no=0 yes=1
4	pb4	Connects general I/O signal **pb4** to pin **pb4**. no: Not connected yes: Connected	no=0 yes=1
3	pb3	Connects general I/O signal **pb3** to pin **pb3**. no: Not connected yes: Connected	no=0 yes=1
2	pb2	Connects general I/O signal **pb2** to pin **pb2**. no: Not connected yes: Connected	no=0 yes=1
1	pb1	Connects general I/O signal **pb1** to pin **pb1**. no: Not connected yes: Connected	no=0 yes=1
0	pb0	Connects general I/O signal **pb0** to pin **pb0**. no: Not connected yes: Connected	no=0 yes=1

25.38.4 rw_pb_iop

Address	0xc
Default	0x00000000
Type	Read/Write
Description	Connects I/O processor port pb to port **pb**.

Bit(s)	Name	Description	Value
17	pb17	Connects I/O processor signal **pb17** to pin **pb17**. no: Not connected yes: Connected	no=0 yes=1
16	pb16	Connects I/O processor signal **pb16** to pin **pb16**. no: Not connected yes: Connected	no=0 yes=1
15	pb15	Connects I/O processor signal **pb15** to pin **pb15**. no: Not connected yes: Connected	no=0 yes=1
14	pb14	Connects I/O processor signal **pb14** to pin **pb14**. no: Not connected yes: Connected	no=0 yes=1
13	pb13	Connects I/O processor signal **pb13** to pin **pb13**. no: Not connected yes: Connected	no=0 yes=1
12	pb12	Connects I/O processor signal **pb12** to pin **pb12**. no: Not connected yes: Connected	no=0 yes=1
11	pb11	Connects I/O processor signal **pb11** to pin **pb11**. no: Not connected yes: Connected	no=0 yes=1
10	pb10	Connects I/O processor signal **pb10** to pin **pb10**. no: Not connected yes: Connected	no=0 yes=1
9	pb9	Connects I/O processor signal **pb9** to pin **pb9**. no: Not connected yes: Connected	no=0 yes=1
8	pb8	Connects I/O processor signal **pb8** to pin **pb8**. no: Not connected yes: Connected	no=0 yes=1
7	pb7	Connects I/O processor signal **pb7** to pin **pb7**. no: Not connected yes: Connected	no=0 yes=1
6	pb6	Connects I/O processor signal **pb6** to pin **pb6**. no: Not connected yes: Connected	no=0 yes=1

5	pb5	Connects I/O processor signal **pb5** to pin **pb5**. no: Not connected yes: Connected	no=0 yes=1
4	pb4	Connects I/O processor signal **pb4** to pin **pb4**. no: Not connected yes: Connected	no=0 yes=1
3	pb3	Connects I/O processor signal **pb3** to pin **pb3**. no: Not connected yes: Connected	no=0 yes=1
2	pb2	Connects I/O processor signal **pb2** to pin **pb2**. no: Not connected yes: Connected	no=0 yes=1
1	pb1	Connects I/O processor signal **pb1** to pin **pb1**. no: Not connected yes: Connected	no=0 yes=1
0	pb0	Connects I/O processor signal **pb0** to pin **pb0**. no: Not connected yes: Connected	no=0 yes=1

25.38.5 rw_pc_gio

Address	0x10
Default	0x00000000
Type	Read/Write
Description	Connects general I/O port pc to port **pc**.

Bit(s)	Name	Description	Value
17	pc17	Connects general I/O signal **pc17** to pin **pc17**. no: Not connected yes: Connected	no=0 yes=1
16	pc16	Connects general I/O signal **pc16** to pin **pc16**. no: Not connected yes: Connected	no=0 yes=1
15	pc15	Connects general I/O signal **pc15** to pin **pc15**. no: Not connected yes: Connected	no=0 yes=1
14	pc14	Connects general I/O signal **pc14** to pin **pc14**. no: Not connected yes: Connected	no=0 yes=1
13	pc13	Connects general I/O signal **pc13** to pin **pc13**. no: Not connected yes: Connected	no=0 yes=1
12	pc12	Connects general I/O signal **pc12** to pin **pc12**. no: Not connected yes: Connected	no=0 yes=1
11	pc11	Connects general I/O signal **pc11** to pin **pc11**. no: Not connected yes: Connected	no=0 yes=1
10	pc10	Connects general I/O signal **pc10** to pin **pc10**. no: Not connected yes: Connected	no=0 yes=1
9	pc9	Connects general I/O signal **pc9** to pin **pc9**. no: Not connected yes: Connected	no=0 yes=1
8	pc8	Connects general I/O signal **pc8** to pin **pc8**. no: Not connected yes: Connected	no=0 yes=1
7	pc7	Connects general I/O signal **pc7** to pin **pc7**. no: Not connected yes: Connected	no=0 yes=1
6	pc6	Connects general I/O signal **pc6** to pin **pc6**. no: Not connected yes: Connected	no=0 yes=1

5	pc5	Connects general I/O signal **pc5** to pin **pc5**. no: Not connected yes: Connected	no=0 yes=1
4	pc4	Connects general I/O signal **pc4** to pin **pc4**. no: Not connected yes: Connected	no=0 yes=1
3	pc3	Connects general I/O signal **pc3** to pin **pc3**. no: Not connected yes: Connected	no=0 yes=1
2	pc2	Connects general I/O signal **pc2** to pin **pc2**. no: Not connected yes: Connected	no=0 yes=1
1	pc1	Connects general I/O signal **pc1** to pin **pc1**. no: Not connected yes: Connected	no=0 yes=1
0	pc0	Connects general I/O signal **pc0** to pin **pc0**. no: Not connected yes: Connected	no=0 yes=1

25.38.6 rw_pc_iop

Address	0x14
Default	0x00000000
Type	Read/Write
Description	Connects I/O processor port pc to port **pc**.

Bit(s)	Name	Description	Value
17	pc17	Connects I/O processor signal **pc17** to pin **pc17**. no: Not connected yes: Connected	no=0 yes=1
16	pc16	Connects I/O processor signal **pc16** to pin **pc16**. no: Not connected yes: Connected	no=0 yes=1
15	pc15	Connects I/O processor signal **pc15** to pin **pc15**. no: Not connected yes: Connected	no=0 yes=1
14	pc14	Connects I/O processor signal **pc14** to pin **pc14**. no: Not connected yes: Connected	no=0 yes=1
13	pc13	Connects I/O processor signal **pc13** to pin **pc13**. no: Not connected yes: Connected	no=0 yes=1
12	pc12	Connects I/O processor signal **pc12** to pin **pc12**. no: Not connected yes: Connected	no=0 yes=1
11	pc11	Connects I/O processor signal **pc11** to pin **pc11**. no: Not connected yes: Connected	no=0 yes=1
10	pc10	Connects I/O processor signal **pc10** to pin **pc10**. no: Not connected yes: Connected	no=0 yes=1
9	pc9	Connects I/O processor signal **pc9** to pin **pc9**. no: Not connected yes: Connected	no=0 yes=1
8	pc8	Connects I/O processor signal **pc8** to pin **pc8**. no: Not connected yes: Connected	no=0 yes=1
7	pc7	Connects I/O processor signal **pc7** to pin **pc7**. no: Not connected yes: Connected	no=0 yes=1
6	pc6	Connects I/O processor signal **pc6** to pin **pc6**. no: Not connected yes: Connected	no=0 yes=1

5	pc5	Connects I/O processor signal **pc5** to pin **pc5**. no: Not connected yes: Connected	no=0 yes=1
4	pc4	Connects I/O processor signal **pc4** to pin **pc4**. no: Not connected yes: Connected	no=0 yes=1
3	pc3	Connects I/O processor signal **pc3** to pin **pc3**. no: Not connected yes: Connected	no=0 yes=1
2	pc2	Connects I/O processor signal **pc2** to pin **pc2**. no: Not connected yes: Connected	no=0 yes=1
1	pc1	Connects I/O processor signal **pc1** to pin **pc1**. no: Not connected yes: Connected	no=0 yes=1
0	pc0	Connects I/O processor signal **pc0** to pin **pc0**. no: Not connected yes: Connected	no=0 yes=1

25.38.7 rw_pd_gio

Address	0x18
Default	0x00000000
Type	Read/Write
Description	Connects general I/O port pd to port **pd**.

Bit(s)	Name	Description	Value
17	pd17	Connects general I/O signal **pd17** to pin **pd17**. no: Not connected yes: Connected	no=0 yes=1
16	pd16	Connects general I/O signal **pd16** to pin **pd16**. no: Not connected yes: Connected	no=0 yes=1
15	pd15	Connects general I/O signal **pd15** to pin **pd15**. no: Not connected yes: Connected	no=0 yes=1
14	pd14	Connects general I/O signal **pd14** to pin **pd14**. no: Not connected yes: Connected	no=0 yes=1
13	pd13	Connects general I/O signal **pd13** to pin **pd13**. no: Not connected yes: Connected	no=0 yes=1
12	pd12	Connects general I/O signal **pd12** to pin **pd12**. no: Not connected yes: Connected	no=0 yes=1
11	pd11	Connects general I/O signal **pd11** to pin **pd11**. no: Not connected yes: Connected	no=0 yes=1
10	pd10	Connects general I/O signal **pd10** to pin **pd10**. no: Not connected yes: Connected	no=0 yes=1
9	pd9	Connects general I/O signal **pd9** to pin **pd9**. no: Not connected yes: Connected	no=0 yes=1
8	pd8	Connects general I/O signal **pd8** to pin **pd8**. no: Not connected yes: Connected	no=0 yes=1
7	pd7	Connects general I/O signal **pd7** to pin **pd7**. no: Not connected yes: Connected	no=0 yes=1
6	pd6	Connects general I/O signal **pd6** to pin **pd6**. no: Not connected yes: Connected	no=0 yes=1

5	pd5	Connects general I/O signal **pd5** to pin **pd5**. no: Not connected yes: Connected	no=0 yes=1
4	pd4	Connects general I/O signal **pd4** to pin **pd4**. no: Not connected yes: Connected	no=0 yes=1
3	pd3	Connects general I/O signal **pd3** to pin **pd3**. no: Not connected yes: Connected	no=0 yes=1
2	pd2	Connects general I/O signal **pd2** to pin **pd2**. no: Not connected yes: Connected	no=0 yes=1
1	pd1	Connects general I/O signal **pd1** to pin **pd1**. no: Not connected yes: Connected	no=0 yes=1
0	pd0	Connects general I/O signal **pd0** to pin **pd0**. no: Not connected yes: Connected	no=0 yes=1

25.38.8 rw_pd_iop

Address	0x1c
Default	0x00000000
Type	Read/Write
Description	Connects I/O processor port pd to port **pd**.

Bit(s)	Name	Description	Value
17	pd17	Connects I/O processor signal **pd17** to pin **pd17**. no: Not connected yes: Connected	no=0 yes=1
16	pd16	Connects I/O processor signal **pd16** to pin **pd16**. no: Not connected yes: Connected	no=0 yes=1
15	pd15	Connects I/O processor signal **pd15** to pin **pd15**. no: Not connected yes: Connected	no=0 yes=1
14	pd14	Connects I/O processor signal **pd14** to pin **pd14**. no: Not connected yes: Connected	no=0 yes=1
13	pd13	Connects I/O processor signal **pd13** to pin **pd13**. no: Not connected yes: Connected	no=0 yes=1
12	pd12	Connects I/O processor signal **pd12** to pin **pd12**. no: Not connected yes: Connected	no=0 yes=1
11	pd11	Connects I/O processor signal **pd11** to pin **pd11**. no: Not connected yes: Connected	no=0 yes=1
10	pd10	Connects I/O processor signal **pd10** to pin **pd10**. no: Not connected yes: Connected	no=0 yes=1
9	pd9	Connects I/O processor signal **pd9** to pin **pd9**. no: Not connected yes: Connected	no=0 yes=1
8	pd8	Connects I/O processor signal **pd8** to pin **pd8**. no: Not connected yes: Connected	no=0 yes=1
7	pd7	Connects I/O processor signal **pd7** to pin **pd7**. no: Not connected yes: Connected	no=0 yes=1
6	pd6	Connects I/O processor signal **pd6** to pin **pd6**. no: Not connected yes: Connected	no=0 yes=1

5	pd5	Connects I/O processor signal **pd5** to pin **pd5**. no: Not connected yes: Connected	no=0 yes=1
4	pd4	Connects I/O processor signal **pd4** to pin **pd4**. no: Not connected yes: Connected	no=0 yes=1
3	pd3	Connects I/O processor signal **pd3** to pin **pd3**. no: Not connected yes: Connected	no=0 yes=1
2	pd2	Connects I/O processor signal **pd2** to pin **pd2**. no: Not connected yes: Connected	no=0 yes=1
1	pd1	Connects I/O processor signal **pd1** to pin **pd1**. no: Not connected yes: Connected	no=0 yes=1
0	pd0	Connects I/O processor signal **pd0** to pin **pd0**. no: Not connected yes: Connected	no=0 yes=1

25.38.9 rw_pe_gio

Address	0x20
Default	0x00000000
Type	Read/Write
Description	Connects general I/O port pe to port **pe**.

Bit(s)	Name	Description	Value
17	pe17	Connects general I/O signal **pe17** to pin **pe17**. no: Not connected yes: Connected	no=0 yes=1
16	pe16	Connects general I/O signal **pe16** to pin **pe16**. no: Not connected yes: Connected	no=0 yes=1
15	pe15	Connects general I/O signal **pe15** to pin **pe15**. no: Not connected yes: Connected	no=0 yes=1
14	pe14	Connects general I/O signal **pe14** to pin **pe14**. no: Not connected yes: Connected	no=0 yes=1
13	pe13	Connects general I/O signal **pe13** to pin **pe13**. no: Not connected yes: Connected	no=0 yes=1
12	pe12	Connects general I/O signal **pe12** to pin **pe12**. no: Not connected yes: Connected	no=0 yes=1
11	pe11	Connects general I/O signal **pe11** to pin **pe11**. no: Not connected yes: Connected	no=0 yes=1
10	pe10	Connects general I/O signal **pe10** to pin **pe10**. no: Not connected yes: Connected	no=0 yes=1
9	pe9	Connects general I/O signal **pe9** to pin **pe9**. no: Not connected yes: Connected	no=0 yes=1
8	pe8	Connects general I/O signal **pe8** to pin **pe8**. no: Not connected yes: Connected	no=0 yes=1
7	pe7	Connects general I/O signal **pe7** to pin **pe7**. no: Not connected yes: Connected	no=0 yes=1
6	pe6	Connects general I/O signal **pe6** to pin **pe6**. no: Not connected yes: Connected	no=0 yes=1

5	pe5	Connects general I/O signal **pe5** to pin **pe5**. no: Not connected yes: Connected	no=0 yes=1
4	pe4	Connects general I/O signal **pe4** to pin **pe4**. no: Not connected yes: Connected	no=0 yes=1
3	pe3	Connects general I/O signal **pe3** to pin **pe3**. no: Not connected yes: Connected	no=0 yes=1
2	pe2	Connects general I/O signal **pe2** to pin **pe2**. no: Not connected yes: Connected	no=0 yes=1
1	pe1	Connects general I/O signal **pe1** to pin **pe1**. no: Not connected yes: Connected	no=0 yes=1
0	pe0	Connects general I/O signal **pe0** to pin **pe0**. no: Not connected yes: Connected	no=0 yes=1

25.38.10 rw_pe_iop

Address	0x24
Default	0x00000000
Type	Read/Write
Description	Connects I/O processor port pe to port **pe**.

Bit(s)	Name	Description	Value
17	pe17	Connects I/O processor signal **pe17** to pin **pe17**. no: Not connected yes: Connected	no=0 yes=1
16	pe16	Connects I/O processor signal **pe16** to pin **pe16**. no: Not connected yes: Connected	no=0 yes=1
15	pe15	Connects I/O processor signal **pe15** to pin **pe15**. no: Not connected yes: Connected	no=0 yes=1
14	pe14	Connects I/O processor signal **pe14** to pin **pe14**. no: Not connected yes: Connected	no=0 yes=1
13	pe13	Connects I/O processor signal **pe13** to pin **pe13**. no: Not connected yes: Connected	no=0 yes=1
12	pe12	Connects I/O processor signal **pe12** to pin **pe12**. no: Not connected yes: Connected	no=0 yes=1
11	pe11	Connects I/O processor signal **pe11** to pin **pe11**. no: Not connected yes: Connected	no=0 yes=1
10	pe10	Connects I/O processor signal **pe10** to pin **pe10**. no: Not connected yes: Connected	no=0 yes=1
9	pe9	Connects I/O processor signal **pe9** to pin **pe9**. no: Not connected yes: Connected	no=0 yes=1
8	pe8	Connects I/O processor signal **pe8** to pin **pe8**. no: Not connected yes: Connected	no=0 yes=1
7	pe7	Connects I/O processor signal **pe7** to pin **pe7**. no: Not connected yes: Connected	no=0 yes=1
6	pe6	Connects I/O processor signal **pe6** to pin **pe6**. no: Not connected yes: Connected	no=0 yes=1

5	pe5	Connects I/O processor signal **pe5** to pin **pe5**. no: Not connected yes: Connected	no=0 yes=1
4	pe4	Connects I/O processor signal **pe4** to pin **pe4**. no: Not connected yes: Connected	no=0 yes=1
3	pe3	Connects I/O processor signal **pe3** to pin **pe3**. no: Not connected yes: Connected	no=0 yes=1
2	pe2	Connects I/O processor signal **pe2** to pin **pe2**. no: Not connected yes: Connected	no=0 yes=1
1	pe1	Connects I/O processor signal **pe1** to pin **pe1**. no: Not connected yes: Connected	no=0 yes=1
0	pe0	Connects I/O processor signal **pe0** to pin **pe0**. no: Not connected yes: Connected	no=0 yes=1

25.38.11 rw_usb_phy

Address	0x28
Default	0x00000000
Type	Read/Write
Description	Connects the internal USB 1.1 phy to the I/O processor. Two different mappings are available.

Bit(s)	Name	Description	Value
1	en_usb1	Enables the I/O-processor to use the internal USB phy. This field connects I/O processor signals **pe14** - **pe8** to the USB phy. no: USB phy not connected yes: USB phy connected to the I/O processor	no=0 yes=1
0	en_usb0	Enables the I/O processor to use the internal USB phy. This field connects I/O processor signals **pc14** - **pc8** to the USB phy. no: USB phy not connected yes: USB phy connected to the I/O processor	no=0 yes=1

25.39 rt_trace

Instance	Base Address
trace	0xb0040000

25.39.1 rw_cfg

Address	0x0
Default	0x00000000
Type	Read/Write
Description	Configuration of real time trace. This register may be written by software and by the TAP controller. This enables the TAP controller to control the real time trace functionality.

Bit(s)	Name	Description	Value
22-16:7	wp_stop	Specifies which watchpoints stop the tracing. Requires that the mode field is set to wp, and that the corresponding watchpoints in the CPU are turned on. wp0: Watchpoint 0 stops tracing wp1: Watchpoint 1 stops tracing wp2: Watchpoint 2 stops tracing wp3: Watchpoint 3 stops tracing wp4: Watchpoint 4 stops tracing wp5: Watchpoint 5 stops tracing wp6: Watchpoint 6 stops tracing	wp0=1 wp1=2 wp2=4 wp3=8 wp4=16 wp5=32 wp6=64
14-8:7	wp_start	Specifies which watchpoints start the tracing. Requires that the mode field is set to wp, and that the corresponding watchpoints in the CPU are turned on. wp0: Watchpoint 0 starts tracing wp1: Watchpoint 1 starts tracing wp2: Watchpoint 2 starts tracing wp3: Watchpoint 3 starts tracing wp4: Watchpoint 4 starts tracing wp5: Watchpoint 5 starts tracing wp6: Watchpoint 6 starts tracing	wp0=1 wp1=2 wp2=4 wp3=8 wp4=16 wp5=32 wp6=64
4	stall	Decides if the tracer may stop the CPU when the trace buffer is full to prevent loss of trace data. When off, the tracer may temporarily loose track when there are a lot of jumps in a short time. no: Tracer will never stop the CPU yes: Tracer may stop the CPU	no=0 yes=1
3	wp	Turn watchpoint tracing on/off. no: Watchpoint tracing disabled yes: Watchpoint tracing enabled	no=0 yes=1

2	owner	Turn ownership tracing on/off. no: Ownership tracing disabled yes: Ownership tracing enabled	no=0 yes=1
1	mode	Decides how tracing is started and stopped. normal: Tracing always on wp: Tracing start/stop controlled by watchpoints	normal=0 wp=1
0	en	Turn real time tracing as a whole on/off. no: Disable real time tracing yes: Enable real time tracing	no=0 yes=1

25.39.2 rw_tap_ctrl

Address	0x4
Default	
Type	Read/Write
Description	JTAG TAP debug data register control.

Bit(s)	Name	Description	Value
1	ack_guru	Acknowledge Guru exception caused by the TAP. yes: Acknowledge Guru exception no: No operation	yes=1 no=0
0	ack_data	Acknowledge read data. I.e. clear the dav field of the r_tap_stat register. yes: Acknowledge read data no: No operation	yes=1 no=0

25.39.3 r_tap_stat

Address	0x8
Default	
Type	Read
Description	JTAG TAP debug data register status.

Bit(s)	Name	Description	Value
1	empty	Tells if data written to rw_tap_data has been read via the TAP controller or not. Cleared when rw_tap_data is written. Set when TAP has captured data from rw_tap_data and rw_tap_hdata. yes: Write register is empty no: Write register is full	yes=1 no=0
0	dav	New data from TAP available in rw_tap_data and rw_tap_hdata. Cleared by writing to the ack_data bit of rw_tap_ctrl. yes: New data is available no: No new data	yes=1 no=0

25.39.4 rw_tap_data

Address	0xc
Default	
Type	Read/Write
Description	Read/write debug data register in JTAG TAP controller, bit 0 - 31.

25.39.5 rw_tap_hdata

Address	0x10
Default	
Type	Read/Write
Description	Read/write debug data register in JTAG TAP controller, bit 32 - 39.

Bit(s)	Name	Description	Value
7-4:4	sub_op	Sub-operation code. gmode: Guru mode entered rdreg: Read general register rdpreg: Read special register rdsreg: Read support register wrreg: Write general register wrpreg: Write special register wrsreg: Write support register rdmem: Read dword from memory wrmem: Write dword to memory rdmemb: Read byte from memory wrmemb: Write byte to memory ret: Return from Guru mode brk: Break current command	gmode=0 rdreg=1 rdpreg=2 rdsreg=3 wrreg=4 wrpreg=5 wrsreg=6 rdmem=7 wrmem=8 rdmemb=9 wrmemb=10 ret=11 brk=12
3-0:4	op	Operation code. nop: No operation trcfg: Controls real time tracing by writing the rw_cfg register dbg: Make CPU enter Guru debug mode dbgdi: Data to Guru debug mode dbgdo: Data from Guru debug mode redir: Write debug redirection address register (r_redir)	nop=0 trcfg=1 dbg=3 dbgdi=4 dbgdo=5 redir=6

25.39.6 r_redir

Address	0x14
Default	
Type	Read
Description	If non-zero, the debug ROM will jump to the address read here when triggered instead of running the normal debug ROM code. This register can only be written through the TAP using the redir command as specified in rw_tap_hdata.

25.40 ser

Instance	Base Address
ser0	0xb0026000
ser1	0xb0028000
ser2	0xb002a000
ser3	0xb002c000

25.40.1 rw_tr_ctrl

Address	0x0
Default	0x00008000
Type	Read/Write
Description	Configuration for the serial interface transmitter.

Bit(s)	Name	Description	Value
16	auto_cts	Automatic handling of **cts_n**. When enabled, a high signal on **cts_n** stops transmission after the ongoing byte. no: No automatic **cts_n** handling yes: Automatic **cts_n** handling	no=0 yes=1
15	txd	Value on the **txd** pin when the transmitter is disabled or stopped.	
14	auto_rts	Automatically set **rts_n** for half-duplex operation. The **rts_n** pin is set to the value of the rts_n field of the rw_rec_ctrl register when the transmitter is paused and to the inverted value when transmitting. If this mode is not enabled, the **rts_n** will be set to the value of the rts_n field at all times. no: No automatic **rts_n** handling yes: Automatic **rts_n** handling	no=0 yes=1
13	rts_setup	When auto_rts is set to yes, this field sets the delay from **rts_n** is toggled until the start of the transmission. The delay is set relative to the start of the start bit. bits1: One bit delay bits2: Two bits delay	bits1=0 bits2=1

12-10:3	rts_delay	When auto_rts is set to yes, this field sets the delay from completion of transmission until **rts_n** is toggled. The delay is set relative to the start of the first stop bit. del0_5: Delay 0.5 bit times del1: Delay 1 bit time del1_5: Delay 1.5 bit times del2: Delay 2 bit times del2_5: Delay 2.5 bit times del3: Delay 3 bit times del3_5: Delay 3.5 bit times del4: Delay 4 bit times	del0_5=0 del1=1 del1_5=2 del2=3 del2_5=4 del3=5 del3_5=6 del4=7
9	stop	Stop transmitter after the ongoing byte. When this field is set, the ongoing transmission (if any) will be completed including the specified number of stop bits, and thereafter the tr_empty and tr_idle fields will be set to yes and the **txd** pin will be set to the value of the txd field. no: Do not stop yes: Stop	no=0 yes=1
8	stop_bits	Number of stop bits. bits1: One stop bit bits2: Two stop bits	bits1=0 bits2=1
7	data_bits	Number of data bits for the transmitter. bits8: 8 data bits bits7: 7 data bits	bits8=0 bits7=1
6	par_en	Enable parity generation for the transmitter. no: Parity disabled yes: Parity enabled	no=0 yes=1
5-4:2	par	Selects parity for the transmitter. even: Even parity odd: Odd parity mark: MARK parity (logic 1) space: SPACE parity (logic 0)	even=0 odd=1 mark=2 space=3
3	en	Enable the serial transmitter. no: Serial transmitter disabled yes: Serial transmitter enabled	no=0 yes=1
2-0:3	base_freq	Transmitter baud rate base frequency. The transmitter baud clock is base_freq / (rw_tr_baud_div.div * 8). off: No clock ext: External clock input f29_493: 29.493 MHz f32: 32.000 MHz f32_768: 32.768 MHz f100: 100 MHz	off=0 ext=1 f29_493=4 f32=5 f32_768=6 f100=7

25.40.2 rw_tr_dma_en

Address	0x4
Default	0x00000000
Type	Read/Write
Description	Enables DMA for the serial transmitter.

Bit(s)	Name	Description	Value
0	en	Enable DMA. This field will be set to no by the DMA when the DMA reaches end of packet. no: DMA disabled yes: DMA enabled	no=0 yes=1

25.40.3 rw_rec_ctrl

Address	0x8
Default	0x00010000
Type	Read/Write
Description	Configuration for the serial interface receiver.

Bit(s)	Name	Description	Value
17	loopback	Enables internal loop back from the serial transmitter. no: Normal operation yes: Internal loop back	no=0 yes=1
16	rts_n	Controls the **rts_n** pin. When read, this field returns the value written to it. To read the actual value on the **rts_n** pin, use the rts_n field of the r_stat_din register. active: The **rts_n** pin is set to active inactive: The **rts_n** pin is set to inactive	active=0 inactive=1
15	half_duplex	Automatically disables receiver while transmitting. This is useful in e.g. 2-wire RS-485 applications to avoid receiving your own data. no: Full duplex yes: Half duplex	no=0 yes=1
14	auto_eop	Generate eop to the DMA when no characters have been received for the time specified by the timeout field. no: Automatic eop generation disabled yes: Automatic eop generation enabled	no=0 yes=1
13-11:3	timeout	Receive timeout (in character times). A value of 0 will generate an eop to the DMA as soon as characters are not received back-to-back.	
10	sampling	Sampling mode for serial receiver. middle: One sample in the middle of the data bit majority: Majority of three samples in the middle of the data bit	middle=0 majority=1
9	dma_err	Controls the handling of receive errors when DMA is used. If stop is selected a receive error will cause an eop to the DMA channel and the DMA transfers will stop. The erroneous byte will not be forwarded to the DMA. stop: Receive error generates eop and stops the DMA ignore: Receive errors are ignored when DMA is used	stop=0 ignore=1
8	dma_mode	DMA is used to receive data. no: Register access mode yes: DMA mode	no=0 yes=1
7	data_bits	Number of data bits for the receiver. bits8: 8 data bits bits7: 7 data bits	bits8=0 bits7=1

6	par_en	Enable parity check for the receiver. no: Parity disabled yes: Parity enabled	no=0 yes=1
5-4:2	par	Selects parity for the receiver. even: Even parity odd: Odd parity mark: MARK parity (logic 1) space: SPACE parity (logic 0)	even=0 odd=1 mark=2 space=3
3	en	Enable the serial receiver. no: Serial receiver disabled yes: Serial receiver enabled	no=0 yes=1
2-0:3	base_freq	Receiver baud rate base frequency. The receiver baud clock is base_freq / (rw_rec_baud_div.div * 8). off: No clock ext: External clock input f29_493: 29.493 MHz f32: 32.000 MHz f32_768: 32.768 MHz f100: 100 MHz	off=0 ext=1 f29_493=4 f32=5 f32_768=6 f100=7

25.40.4 rw_tr_baud_div

Address	0xc
Default	0x00000000
Type	Read/Write
Description	Transmitter baud rate divisor. The baud clock is rw_tr_ctrl.base_freq / (div * 8). Note that writing to this register will restart the baud rate generator. Writing to the register while transmitting will therefore corrupt outgoing data even if the register contents is not changed.

Bit(s)	Name	Description	Value
15-0:16	div	Transmitter baud rate divisor.	

25.40.5 rw_rec_baud_div

Address	0x10
Default	0x00000000
Type	Read/Write
Description	Receiver baud rate divisor. The baud clock is rw_rec_ctrl.base_freq / (div * 8). Note that writing to this register will restart the baud rate generator. Writing to the register while receiving will therefore corrupt incoming data even if the register contents is not changed.

Bit(s)	Name	Description	Value
15-0:16	div	Receiver baud rate divisor.	

25.40.6 rw_xoff

Address	0x14
Default	0x00000000
Type	Read/Write
Description	Configuration of xoff for the serial interface.

Bit(s)	Name	Description	Value
8	automatic	Automatic xoff handling. When enabled, the transmitter is automatically stopped when an xoff character is detected. no: Disable automatic xoff handling yes: Enable automatic xoff handling	no=0 yes=1
7-0:8	chr	The xoff character code.	

25.40.7 rw_xoff_clr

Address	0x18
Default	
Type	Read/Write
Description	Clears the xoff_detect field in the r_stat_din and rs_stat_din registers.

Bit(s)	Name	Description	Value
0	clr	Clears xoff_detect in the r_stat_din and rs_stat_din registers. no: No operation yes: Clear	no=0 yes=1

25.40.8 rw_dout

Address	0x1c
Default	
Type	Read/Write
Description	Transmit data.

Bit(s)	Name	Description	Value
7-0:8	data	Byte to transmit.	

25.40.9 rs_stat_din/r_stat_din

Address	0x20/0x24
Default	
Type	Read with side effects/Read
Description	Serial port status and received data. Read with side effects will clear the orun, par_err, framing_err and dav fields.

Bit(s)	Name	Description	Value
28	txd	The value on the **txd** pin.	
27	rts_n	Value on the **rts_n** pin. active: **rts_n** is low (active) inactive: **rts_n** is high (inactive)	active=0 inactive=1
26	xoff_detect	An xoff character detected. Cleared by writing to the rw_xoff_clr register. An xoff is only detected if the automatic field of the rw_xoff register is set to yes. no: No xoff detected yes: xoff detected	no=0 yes=1
25	cts_n	Value on the **cts_n** pin. active: **cts_n** is low (active) inactive: **cts_n** is high (inactive)	active=0 inactive=1
24	tr_rdy	Serial transmitter ready (a character can be written to rw_dout). no: Transmitter not ready yes: Transmitter ready	no=0 yes=1
23	tr_empty	Serial transmitter empty. The tr_empty field is set when the transmitter is idle (see tr_idle below) or when there is no ongoing transmission and the DMA FIFO is empty. Note that the tr_empty field can be used as an indication of a completed DMA transfer only when the associated DMA channel has reached end of list. no: Transmitter not empty yes: Transmitter empty	no=0 yes=1
22	tr_idle	Serial transmitter is idle. The condition for tr_idle is: (no ongoing transmission) and ((dma disabled) or ((automatic xoff) and (xoff detected)) or (transmitter stopped) or (transmitter disabled)). no: Transmitter not idle yes: Transmitter idle	no=0 yes=1
21	rxd	Value on the **rxd** pin.	
20	rec_err	Receiver error. Indicates that an overrun error, a framing error or a parity error has occured. Cleared by reading rs_stat_din. This field is not cleared when a new correct character is received.	

19	orun	Receiver overrun. Indicates that a new character was received before the previous one was read out or transferred to the DMA, and that the previous character was lost. Cleared by reading rs_stat_din. no: No overrun yes: Overrun	no=0 yes=1
18	par_err	Parity error. This field shows the parity status of the last received character. Cleared by reading rs_stat_din. no: No parity error yes: Parity error	no=0 yes=1
17	framing_err	Framing error. This field shows the framing status of the last received character. Cleared by reading rs_stat_din. no: No framing error yes: Framing error	no=0 yes=1
16	dav	Data is available from the serial receiver. Cleared by reading rs_stat_din. no: No data available yes: Data available	no=0 yes=1
7-0:8	data	Received data.	

25.40.10 rw_rec_eop

Address	0x28
Default	
Type	Read/Write
Description	This register is used by the software to set an eop (end of packet) to the DMA channel of the serial receiver.

Bit(s)	Name	Description	Value
0	set	Set eop to the DMA. no: No operation yes: Set eop	no=0 yes=1

25.40.11 rw_intr_mask

Address	0x2c
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	dav	Enable/disable dav interrupt. Data available interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	tr_idle	Enable/disable tr_idle interrupt. Transmitter idle interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	tr_empty	Enable/disable tr_empty interrupt. Transmitter empty interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	tr_rdy	Enable/disable tr_rdy interrupt. Transmitter ready interrupt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.40.12 rw_ack_intr

Address	0x30
Default	
Type	Read/Write
Description	Acknowledge interrupts.

Bit(s)	Name	Description	Value
3	dav	Acknowledge dav interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	tr_idle	Acknowledge tr_idle interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	tr_empty	Acknowledge tr_empty interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	tr_rdy	Acknowledge tr_rdy interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.40.13 r_intr

Address	0x34
Default	
Type	Read
Description	Interrupts before the mask. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	dav	Interrupt dav active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	tr_idle	Interrupt tr_idle active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	tr_empty	Interrupt tr_empty active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	tr_rdy	Interrupt tr_rdy active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.40.14 r_masked_intr

Address	0x38
Default	
Type	Read
Description	Interrupts after the mask. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	dav	Interrupt dav active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	tr_idle	Interrupt tr_idle active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	tr_empty	Interrupt tr_empty active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	tr_rdy	Interrupt tr_rdy active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.41 sser

Instance	Base Address
sser0	0xb0022000
sser1	0xb0024000

25.41.1 rw_cfg

Address	0x0
Default	0x00000000
Type	Read/Write
Description	General control register for the synchronous serial port.

Bit(s)	Name	Description	Value
30	en	Turns the entire port on or off. The fields rw_tr_cfg.tr_en and rw_rec_cfg.rec_en have no effect if this field is disabled. yes: Enable port no: Disable port	yes=1 no=0
29	prepare	Prepare transmitter and frame for operation. This field resets frame circuit and keeps transmitter from starting. Only needed in some special cases. yes: Prepare no: Normal operation	yes=1 no=0
28	hold_pol	Hold signal polarity, if used, for both incoming and outgoing signals. pos: Active high polarity, 1 = hold neg: Active low polarity, 0 = hold	pos=0 neg=1
27	clk_in_sel	Select which clock input to use (if any). clk_in: Use clk_in (**clk**) ext_clk: Use ext_clk (shared **ext_clk**)	clk_in=0 ext_clk=1
26-25:2	out_clk_src	Selects the source for the output clock. In nojitter mode, ext_clk frequency must be no higher than 16.67 MHz, and clk_div must not be set to 0. clk100: The internal 100 MHz clock intern_clk: An internally generated clock nojitter: As intern_clk but output on the pos edge of ext_clk const0: Constant value, controlled by out_clk_pol	clk100=0 intern_clk=1 nojitter=2 const0=3
24	out_clk_pol	Selects polarity of output clock pos: Normal polarity neg: Inverted polarity	pos=0 neg=1

23	clk_od_mode	Turns **clk** pin into an open-drain output (if clk_dir is out). Gives expected results only when out_clk_src is set to intern_clk. yes: Open drain mode no: Normal mode	yes=1 no=0
22	clk_dir	Specifies the direction of the **clk** pin. out: The **clk** pin is an output in: The **clk** pin is an input	out=1 in=0
21	clkgate_in	Specifies whether the incoming clock is gated or continuous. Note: Must be set to no when using internal clocking. yes: The incoming external clock is gated no: The incoming external clock is continuous	yes=1 no=0
20	clkgate_ctrl	If gate_clk is yes, this field specifies which unit that controls clock gating. tr: Transmitter controls clock gating rec: Receiver controls clock gating	tr=1 rec=0
19	gate_clk	Selects if an output clock shall be gated or continuous. If gated, the level of the **clk** pin is controlled by out_clk_pol when inactive. yes: The outgoing internal clock is gated no: The outgoing internal clock is continuous	yes=1 no=0
18-16:3	base_freq	Frequency base for internal clock generator. The clock generator output frequency will be this frequency divided by clk_div. f100: 100 MHz ext: External clock input f29_493: 29.493 MHz f32: 32.000 MHz f32_768: 32.768 MHz no_clk: No clock	f100=0 ext=1 f29_493=4 f32=5 f32_768=6 no_clk=7
15-0:16	clk_div	Division factor, minus 1, for internal clock generator. See the 23.4.4.1 for details.	

25.41.2 rw_frm_cfg

Address	0x4
Default	0x00000000
Type	Read/Write
Description	Configures frame signalling.

Bit(s)	Name	Description	Value
30-29:2	status_pin_use	Selects how to use the **status** pin. Notes: 1. The field rw_tr_cfg.dual_i2s overrides this setting and uses the **status** pin as a second I2S output. 2. If rw_rec_cfg.slave2_en is set, set this field to gio0. frm: Frame signalling hold: Hold signalling gio1: General IO, with value 1 if output gio0: General IO, with value 0 if output	frm=3 hold=2 gio1=1 gio0=0
28	status_pin_dir	Selects the direction of the **status** pin. out: The **status** pin is an output in: The **status** pin is an input	out=1 in=0
27-26:2	frame_pin_use	Selects how to use the **frame** pin. Note: If rw_rec_cfg.slave3_en is set and rw_rec_cfg.mode equals wiresave, set this field to gio0. frm: Frame signalling hold: Hold signalling gio1: General IO, with value 1 if output gio0: General IO, with value 0 if output	frm=3 hold=2 gio1=1 gio0=0
25	frame_pin_dir	Selects the direction of the **frame** pin. out: The **frame** pin is an output in: The **frame** pin is an input	out=1 in=0
24	out_on	If a frame signal is output, this field specifies which frame event source that triggers it. tr: Transmitter controls frame output intern_tb: internal frame time base	tr=1 intern_tb=0
23	out_off	If a frame signal is output, and type is set to level, this field controls which unit that generates the end-of-word signal that turns off the frame output. tr: Transmitter turns off frame rec: Receiver turns off frame	tr=1 rec=0

22	clk_src	Selects internal or external clock for frame signal sampling and output. See also fr_in_rxclk. ext: External clock is used intern: Internal clock is used	ext=1 intern=0
21	fr_in_rxclk	Sample the incoming external frame signal (SSI in slave mode) with the receiver's clock instead of the frame clock. This field has no effect when not using an incoming external frame signal. yes: Use receiver clock for frame decoding no: Use frame clock for frame decoding	yes=1 no=0
20	clk_pol	Selects active edge of the frame clock. Note: When using internal clock with rw_cfg.clk_div = 0, positive edge is always used regardless of this setting. The output clock can still be inverted by rw_cfg.out_clk_pol though. pos: Positive edge is used neg: Negative edge is used	pos=0 neg=1
19	type	Selects if incoming and outgoing frame signal(s) shall be of edge or level type. Edge-sensitive frame input: A new active edge on the frame signal is needed for each new word. Level-sensitive frame input: The active level on the frame signal will trigger the frame circuit, and if the frame signal stays at the active level, the frame circuit will retrigger immediately after each word. Edge-type frame output: edge means that the frame signal is active for one bit time at each word start (except if level equals both, then the output toggles once for each new frame event). Level-type frame output: level means the frame output is active during the entire word. See also early_wend. edge: Edge-type frame level: Level-type frame	edge=0 level=1

18-17:2	level	Selects which edges or levels shall be interpreted as frame events, for both incoming and outgoing frame. Notes: 1. both in conjunction with type = level will give a constantly-active incoming frame, even if none of the **status** or **frame** pins are used as a frame input. 2. When using type.edge with level.both transmission will start with a negative edge (both for frame direction in and out). neg_lo: Negative edges or low level are frame events pos_hi: Positive edges or high level are frame events both: Both edges are frame events	neg_lo=0 pos_hi=1 both=2
16	early_wend	End level-type frame out at word end. yes: End level type frame at the end of the word no: End level type frame one cycle after word end	yes=1 no=0
15-13:3	tr_delay	Specifies a distance, in bit clock cycles, between a frame event and the first data bit for the transmitter. For details see 23.4.2.3. Note: If the value 0 is used together with an external frame input signal, 2 ns of additional setup time is required for the frame signal.	
12-10:3	rec_delay	Specifies a distance, in bit clock cycles, between a frame event and the first data bit for the receiver. For details see 23.4.2.3. Note: If the value 0 is used together with an external frame input signal, 2 ns of additional setup time is required for the frame signal.	
9-0:10	wordrate	Bit clock cycles between start of words, minus one, in isochronous modes (if a new word should start every 16:th cycle, write 15 here).	

25.41.3 rw_tr_cfg

Address	0x8
Default	0x01800000
Type	Read/Write
Description	Transmitter configuration

Bit(s)	Name	Description	Value
27-26:2	bulk_wspace	In bulk mode, minimum space between words. Note: Must be equal to or greater than rw_frm_cfg.tr_delay if bulk mode is used. Must also be equal to or greater than rw_frm_cfg.rec_delay if the receiver uses the transmitter as source of frame events in bulk mode.	
25	od_mode	Turns **data** pin into an open-drain output. yes: Open drain mode no: Normal mode	yes=1 no=0
24-23:2	data_pin_use	**data** pin usage. If general-IO-input mode or three data inputs in non-wiresave-mode are wanted, select ts_out and disable the transmitter, the data pin will then be tristated. Note: When use60958 is set, set this field to dout. ts_out: Data out, but tristated by frame input signal dout: Normal data output gio1: General output, with value 1 gio0: General output, with value 0	ts_out=3 dout=2 gio1=1 gio0=0
22	dual_i2s	Use both I2S transmitters. The **status** pin will be used as a slave output. Must be set to no in other modes than I2S. yes: Both I2S transmitters are used no: One I2S transmitter or other mode is used	yes=1 no=0

21	use_md	Send metadata from DMA data descriptors or mode register. This is only useful in wiresave mode. Notes: 1. The 'md_rec' interrupt is generated whenever metadata is received from DMA, regardless of this setting. 2. xon and xoff metadata (as requested by the receiver) will not be sent if this field is disabled. However, received xon and xoff metadata will still turn the transmitter on and off. Ie, flow control will be disabled in the receive direction. yes: metadata is sent no: metadata is discarded	yes=1 no=0
20	rate_ctrl	Selects isochronous or bulk modes iso: Data is sent at a fixed rate bulk: Data is sent when it is available	iso=0 bulk=1
19-18:2	iec60958_ckdiv	Divide factor, minus one, for the external clock used for IEC60958 transmission. For a value of zero, a 128*Fsamp clock shall be applied to the clock pin. See 23.4.1.4 for clock frequencies for standard IEC60958 sampling frequencies.	
17	use60958	Use the IEC60958 transmitter yes: IEC60958 mode no: other modes	yes=1 no=0
16	frm_src	Selects what triggers the transmission of each word. Don't care in bulk mode (see rate_ctrl). ext: External frame signal intern: Internal frame time base	ext=1 intern=0
15-14:2	mode	Selects transmitter main mode. See 23.4.1 for details. lospeed: lowspeed mode hispeed: highspeed mode wiresave: wiresave mode	lospeed=0 hispeed=1 wiresave=2
13	use_dma	Use DMA or register interface as transmit data source. yes: Source is DMA no: Source is mode register	yes=1 no=0
12	clk_src	Select internal or external clock for transmitter ext: External clock is used intern: Internal clock is used	ext=1 intern=0

11	clk_pol	Selects polarity of clock edge for data output. Note: When using internal clock with rw_cfg.clk_div = 0, positive edges are always used regardless of this setting. The output clock can still be inverted by rw_cfg.out_clk_pol though. pos: Positive edge is used neg: Negative edge is used	pos=0 neg=1
10	sh_dir	Selects shift direction of serial transmission. Must be set to lsbfirst when mode is wiresave. lsbfirst: LSB is sent first msbfirst: MSB is sent first	lsbfirst=0 msbfirst=1
9-4:6	sample_size	The size, minus one, of each sample. If mode is set to lospeed, the size is given in bits, otherwise it is given in nibbles (bits/4). In wiresave mode, write the value 3 here (ie 16 bits per sample).	
3	eop_stop	Stop transfer after next EOP. Note: Will only stop if metadata of last descriptor is other than tx_null. See also the tidle interrupt. This field has no effect if use_dma is set to no. yes: Stop transfer at EOP no: Continue unaffected by EOP	yes=1 no=0
2	urun_stop	Stop the transfer in case of underrun. If urun_stop is not set, one or more junk word(s) will be sent at underrun. yes: Stop transfer at underrun no: Continue as soon as possible after underrun	yes=1 no=0
1	stop	Start, stop or pause the transfer as soon as possible. Will wait until data already (partly) inside the transmitter has been transmitted. yes: Stop transfer no: Continue transfer	yes=1 no=0
0	tr_en	Turns the transmitter on or off. Disabling the transmitter resets its internal state. yes: Enable transmitter no: Disable transmitter	yes=1 no=0

25.41.4 rw_rec_cfg

Address	0xc
Default	0x00000000
Type	Read/Write
Description	Receiver configuration

Bit(s)	Name	Description	Value
28-27:2	fifo_thr	DMA receiver fifo threshold for flow control. When the fifo has less than the threshold number of bytes free, and flow control is used, the hold output is set or xoff is transmitted. When the fifo space is above the threshold again, the hold output is cleared or xon is transmitted. When no flow control is wanted, set to inf. In modereg mode, other values than inf gives hold whenever there is un-acked data in the register r_rec_data. inf: No flow control thr32: Threshold is 32 bytes thr16: Threshold is 16 bytes thr8: Threshold is 8 bytes	inf=3 thr32=2 thr16=1 thr8=0
26	slave3_en	Use slave receiver connected to the **frame** (in wiresave mode) or **data** (in other modes) pins. For other modes than wiresave and I2S, set to no. yes: Third slave receiver is used no: Third slave receiver not used	yes=1 no=0
25	slave2_en	Use slave receiver connected to **status** pin. For other modes than wiresave and I2S, set to no. yes: Second slave receiver is used no: Second slave receiver not used	yes=1 no=0
24-20:5	iec60958_ui_len	Length of a Unit Interval (half a bit time), in 100 MHz samples. Set this field to 0 in other than IEC60958 modes.	
19	use60958	Use the IEC60958 receiver yes: IEC60958 mode no: Other modes	yes=1 no=0
18-17:2	frm_src	Selects what starts reception of each word. tx_bulk: transmitter bulk-mode frame event ext: External frame signal intern: Internal frame time base	tx_bulk=2 ext=1 intern=0
16-15:2	mode	Selects receiver main mode. See 23.4.1 for details. lospeed: lowspeed mode hispeed: highspeed mode wiresave: wiresave mode	lospeed=0 hispeed=1 wiresave=2

14	use_dma	Use DMA or register interface as receive data destination. yes: Destination is DMA no: Destination is mode register	yes=1 no=0
13	clk_src	Select internal or external clock for receiver ext: External clock is used intern: Internal clock is used	ext=1 intern=0
12	clk_pol	Selects polarity of clock edge for data sampling. Note: When using internal clock with rw_cfg.clk_div = 0, positive edges are always used regardless of this setting. The output clock can still be inverted by rw_cfg.out_clk_pol though. pos: Positive edge is used neg: Negative edge is used	pos=0 neg=1
11	sh_dir	Selects shift direction of serial transmission. Must be set to lsbfirst when using wiresave mode. lsbfirst: LSB is received first msbfirst: MSB is received first	lsbfirst=0 msbfirst=1
10-5:6	sample_size	The size, minus one, of each sample. If mode is set to lospeed, the size is given in bits, otherwise it is given in nibbles (bits/4). In wiresave mode, write the value 3 here (ie 16 bits per sample).	
4	eop_stop	Stop transfer after next received EOP (metadata). Reception will continue as soon as the rstop interrupt is ack:ed. yes: Stop transfer at next EOP no: Do not stop transfer	yes=1 no=0
3	orun_stop	Stop the transfer in case of overrun. yes: Stop transfer at overrun no: Continue as soon as possible after overrun	yes=1 no=0

2	stop	Start, stop or pause the transfer. After this bit is written high, partially received data is written to the DMA and 'eop' and 'last' are signalled. Then the rstop interrupt is issued. Notes: 1. DMA flush: It takes a while for the DMA to flush its fifo after EOP, so software has to wait for the in_eop interrupt from DMA after that stop is written, before accessing the data. 2. Stop point: All data words that have been completely clocked in from the serial line 60 ns or more before stop are guaranteed to be flushed to the DMA. When using slave receivers, the last word written from each channel was received in the same clock cycle. When the word length is greater than 16 bits, the sync serial receiver starts writing data to DMA before the reception of a word is finished. If such a word is half-written to DMA when stop is set, the rest of the word is received and flushed to DMA before stopping. 3. Metadata: The metadata stored in the data descriptor when stop or force_eop is used, is 0xX001, where X is 0byz10, where y is one if 'stop' was the reason for writing EOP, and z is one if force_eop was the reason. Both can be set if both should happen simultaneously, in that case reception stops after EOP. yes: Stop transfer no: Continue transfer	yes=1 no=0

1	force_eop	Force eop to the DMA channel. Unlike the stop field, force_eop doesn't interrupt the transmission, only sends EOP to DMA. Notes: The notes 1 and 3 for stop applies also here, plus: 1. EOP point: All data words that have been completely clocked in from the serial line 60 ns or more before stop are guaranteed to be flushed to the DMA before the EOP. Unlike the stop field, force_eop takes action immediately. This means that there can be half-written words in the DMA fifo (if wordlength $>$ 16 bits and (the word wasn't completely received 60 ns before force_eop or the DMA is full)), and if slave receivers are used, the last word from each receiver before EOP need not have arrived at the same time (if the words weren't completely received 60 ns before force_eop, or if the DMA has become full before all words were flushed). 2. After this bit has been set, wait for the DMA in_eop interrupt, before writing this bit low again. yes: Signal EOP as soon as possible no: No operation	yes=1 no=0
0	rec_en	Turns the receiver on or off yes: Enable receiver no: Disable receiver	yes=1 no=0

25.41.5 rw_tr_data

Address	0x10
Default	0x00000000
Type	Read/Write
Description	Transmit data. Each write to this register will cause one data word to be sent. Notes: 1. The transmitter expects that the first word has been written here before the transmitter is enabled. 2. The trdy interrupt must be awaited before writing this register (except when writing the first word before enabling the transmitter). 3. Only use this register when rw_tr_cfg.use_dma is disabled.

Bit(s)	Name	Description	Value
16	md	Type of data in the data field. Must always be no when rw_tr_cfg.use_md is set to no. Note: Make sure this field is not set before disabling the transmitter in modereg driven mode, otherwise infinite false md_sent interrupts can occur. yes: Metadata no: Ordinary data	yes=1 no=0
15-0:16	data	Data to transmit	

25.41.6 r_rec_data

Address	0x14
Default	
Type	Read
Description	Receive data and external pin read values.

Bit(s)	Name	Description	Value
22	clk_in	Level of the **clk** input pin	
21	data_in	Level of the **data** input pin	
20	din	Level of the **din** input pin	
19	frame_in	Level of the **frame** input pin	
18	status_in	Level of the **status** input pin	
17	ext_clk	Level of the **ext_clk** input pin	
16	md	Type of data in the data field. yes: Metadata no: Ordinary data	yes=1 no=0
15-0:16	data	Received data. An 'rdav' interrupt will be issued when data is available. An acknowledge of this interrupt also acknowledges that this field, and the md field, have been read (so they can be re-used).	

25.41.7 rw_extra

Address	0x18
Default	0x00000000
Type	Read/Write
Description	This register contains fields for clock gating in highspeed mode, and a delay option for output data.

Bit(s)	Name	Description	Value
26-22:5	dout_delay	This field specifies a delay for the data output at `dout`. If greater than zero, the data will be delayed the specified number of 100 MHz clock cycles before being output, thus adding extra hold time after the clock edge and shortening the setup time by the same amount. The hold time applied must not be longer than the clock period used, otherwise output data pulses might be missed. If using dout_delay when the transmitter is clocked by an external clock, note that the data at `dout` will be resampled by the internal clock before being output. This leads to that the delay will vary between dout_delay-1 and dout_delay cycles. Further, a dout_delay value of 1 must not be used with external clocking.	
21	clkon_en	If this field is enabled, the clock output at the **clk** pin will be gated away until the field rw_cfg.prepare is written low. The clock will then be enabled in a clean way. The first visible clock edge will be positive if rw_cfg.out_clk_pol is pos, negative otherwise. yes: Special output-clock-on feature enabled no: normal behavior	yes=1 no=0
20	clkoff_en	If this field is enabled, the output clock will be automatically turned off after a counter, started by the special clock turn-on feature, counting down from the clkoff_cycles value, reaches zero. The edges counted, and the last visible edge, are of the opposite polarity compared to the first visible edge. yes: Special output-clock-off feature enabled no: normal behavior	yes=1 no=0
19-0:20	clkoff_cycles	Number of output clock edges minus one, of the type specified by rw_cfg.out_clk_pol, to output before clock is turned off.	

25.41.8 rw_intr_mask

Address	0x1c
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Interrupts from sync serial port. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
8	r958err	Enable/disable r958err interrupt. IEC60958 receiver error. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
7	md_sent	Enable/disable md_sent interrupt. Metadata sent. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
6	md_rec	Enable/disable md_rec interrupt. Metadata received. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
5	orun	Enable/disable orun interrupt. Receiver overrun error. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
4	urun	Enable/disable urun interrupt. Transmitter underrun error. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
3	rstop	Enable/disable rstop interrupt. Receiver stopped. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	tidle	Enable/disable tidle interrupt. Transmitter idle. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	rdav	Enable/disable rdav interrupt. Receiver data available. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	trdy	Enable/disable trdy interrupt. Transmitter ready for new data. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.41.9 rw_ack_intr

Address	0x20
Default	
Type	Read/Write
Description	Acknowledge interrupts. Interrupts from sync serial port.

Bit(s)	Name	Description	Value
8	r958err	Acknowledge r958err interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
7	md_sent	Acknowledge md_sent interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
6	md_rec	Acknowledge md_rec interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
5	orun	Acknowledge orun interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
4	urun	Acknowledge urun interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
3	rstop	Acknowledge rstop interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	tidle	Acknowledge tidle interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	rdav	Acknowledge rdav interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	trdy	Acknowledge trdy interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.41.10 r_intr

Address	0x24
Default	
Type	Read
Description	Interrupts before the mask. Interrupts from sync serial port. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
8	r958err	Interrupt r958err active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	md_sent	Interrupt md_sent active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	md_rec	Interrupt md_rec active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	orun	Interrupt orun active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
4	urun	Interrupt urun active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	rstop	Interrupt rstop active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	tidle	Interrupt tidle active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	rdav	Interrupt rdav active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	trdy	Interrupt trdy active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.41.11 r_masked_intr

Address	0x28
Default	
Type	Read
Description	Interrupts after the mask. Interrupts from sync serial port. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
8	r958err	Interrupt r958err active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
7	md_sent	Interrupt md_sent active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
6	md_rec	Interrupt md_rec active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
5	orun	Interrupt orun active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
4	urun	Interrupt urun active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
3	rstop	Interrupt rstop active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	tidle	Interrupt tidle active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	rdav	Interrupt rdav active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	trdy	Interrupt trdy active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.42 strcop

Instance	Base Address
strcop	0xb0030000

25.42.1 rw_cfg

Address	0x0
Default	0x00000002
Type	Read/Write
Description	Crypto accelerator configuration register.

Bit(s)	Name	Description	Value
5	en	Enable/disable the crypto accelerator. yes: Enable no: Disable	yes=1 no=0
4	ignore_sync	If this field is set, the sync field in the DMA in channel descriptors will be ignored. Data at the out channel will not be synchronized with data at the in channel. yes: Ignore the sync field no: Use the sync field	yes=1 no=0
3	ipend	IP checksum result endianness. big: Big endian (network order) little: Little endian	big=1 little=0
2	td1	3DES 1st round mode during encryption. Default is encrypt. d: Decrypt e: Encrypt	d=1 e=0
1	td2	3DES 2nd round mode during encryption. Default is decrypt. d: Decrypt e: Encrypt	d=1 e=0
0	td3	3DES 3rd round mode during encryption. Default is encrypt. d: Decrypt e: Encrypt	d=1 e=0

25.43 strmux

Instance	Base Address
strmux	0xb003a000

25.43.1 rw_cfg

Address	0x0
Default	0x00000000
Type	Read/Write
Description	Configuration register for the connection between the on chip I/O blocks and the DMA channels.

Bit(s)	Name	Description	Value
29-27:3	dma9	Connections for DMA channel 9 (in). off: Not connected ext3: External DMA channel 3 strcop: Crypto accelerator ser3: Asynchronous serial port 3	off=0 ext3=1 strcop=2 ser3=3
26-24:3	dma8	Connections for DMA channel 8 (out). off: Not connected ext2: External DMA channel 2 strcop: Crypto accelerator ser3: Asynchronous serial port 3	off=0 ext2=1 strcop=2 ser3=3
23-21:3	dma7	Connections for DMA channel 7 (in). off: Not connected ext1: External DMA channel 1 ser0: Asynchronous serial port 0 sser1: Synchronous serial port 1 eth1: Ethernet interface 1	off=0 ext1=1 ser0=2 sser1=3 eth1=4
20-18:3	dma6	Connections for DMA channel 6 (out). off: Not connected ext0: External DMA channel 0 ser0: Asynchronous serial port 0 sser1: Synchronous serial port 1 eth1: Ethernet interface 1	off=0 ext0=1 ser0=2 sser1=3 eth1=4
17-15:3	dma5	Connections for DMA channel 5 (in). off: Not connected iop1: I/O processor DMA Communicator block 1 ser1: Asynchronous serial port 1 sser0: Synchronous serial port 0	off=0 iop1=1 ser1=2 sser0=3

14-12:3	dma4	Connections for DMA channel 4 (out). off: Not connected iop1: I/O processor DMA Communicator block 1 ser1: Asynchronous serial port 1 sser0: Synchronous serial port 0	off=0 iop1=1 ser1=2 sser0=3
11-9:3	dma3	Connections for DMA channel 3 (in). off: Not connected ext3: External DMA channel 3 iop0: I/O processor DMA Communicator block 0 ata: ATA interface ser2: Asynchronous serial port 2	off=0 ext3=1 iop0=2 ata=3 ser2=4
8-6:3	dma2	Connections for DMA channel 2 (out). off: Not connected ext2: External DMA channel 2 iop0: I/O processor DMA Communicator block 0 ata: ATA interface ser2: Asynchronous serial port 2	off=0 ext2=1 iop0=2 ata=3 ser2=4
5-3:3	dma1	Connections for DMA channel 1 (in). off: Not connected eth0: Ethernet interface 0	off=0 eth0=1
2-0:3	dma0	Connections for DMA channel 0 (out). off: Not connected eth0: Ethernet interface 0	off=0 eth0=1

25.44 timer

Instance	Base Address
timer	0xb001e000

25.44.1 rw_tmr0_div

Address	0x0
Default	
Type	Read/Write
Description	Timer tmr0 divide factor. The timer counts down from this value to 1, then reloads the divide factor, generates the tmr0 interrupt and continues. The register value 0 results in a divide factor of 2^{32}.

25.44.2 r_tmr0_data

Address	0x4
Default	
Type	Read
Description	Current value of timer tmr0.

25.44.3 rw_tmr0_ctrl

Address	0x8
Default	0x00000000
Type	Read/Write
Description	Timer tmr0 control register.

Bit(s)	Name	Description	Value
4-2:3	freq	Selects the input clock frequency to the timer. off: No clock ext: External clock input f29_493: 29.493 MHz f32: 32.000 MHz f32_768: 32.768 MHz f100: 100 MHz	off=0 ext=1 f29_493=4 f32=5 f32_768=6 f100=7
1-0:2	op	Starts/stops the timer. ld: The timer loads the start value hold: The timer stops counting, and holds its value run: The timer counts downwards	ld=0 hold=1 run=2

25.44.4 rw_tmr1_div

Address	0x10
Default	
Type	Read/Write
Description	Timer tmr1 divide factor. The timer counts down from this value to 1, then reloads the divide factor, generates the tmr1 interrupt and continues. The register value 0 results in a divide factor of 2^{32}.

25.44.5 r_tmr1_data

Address	0x14
Default	
Type	Read
Description	Current value of timer tmr1.

25.44.6 rw_tmr1_ctrl

Address	0x18
Default	0x00000000
Type	Read/Write
Description	Timer tmr1 control register.

Bit(s)	Name	Description	Value
4-2:3	freq	Selects the input clock frequency to the timer. off: No clock ext: External clock input f29_493: 29.493 MHz f32: 32.000 MHz f32_768: 32.768 MHz f100: 100 MHz	off=0 ext=1 f29_493=4 f32=5 f32_768=6 f100=7
1-0:2	op	Starts/stops the timer. ld: The timer loads the start value hold: The timer stops counting, and holds its value run: The timer counts downwards	ld=0 hold=1 run=2

25.44.7 rs_cnt_data/r_cnt_data

Address	0x20/0x24
Default	
Type	Read with side effects/Read
Description	Counter cnt data. Read with side effects will clear the cnt field in the r_cnt_data register.

Bit(s)	Name	Description	Value
31-24:8	cnt	Counter value. The counter is incremented by 1 for each input clock pulse, as selected in the rw_cnt_cfg register. The counter saturates at 255. The cnt interrupt is generated whenever the counter is non-zero. A read with side effects will clear the counter.	
23-0:24	tmr	Timer value. When the rw_cnt_cfg register selects a timer output as the input clock, this field contains the lower 24 bits of that timer. When the external clock is selected, or the counter clock is off, this field contains the lower 24 bits of the r_time register.	

25.44.8 rw_cnt_cfg

Address	0x28
Default	0x00000000
Type	Read/Write
Description	Configuration of input clock for counter cnt. This configuration also affects which value that will be read from the tmr field of the r_cnt_data register.

Bit(s)	Name	Description	Value
1-0:2	clk	Selects clock input. off: Clock off, counter holds its value ext: External clock input tmr0: Take clock from tmr0 tmr1: Take clock from tmr1	off=0 ext=1 tmr0=2 tmr1=3

25.44.9 rw_trig

Address	0x30
Default	
Type	Read/Write
Description	This register contains a trig point value, which is compared with one of the timers tmr0 or tmr1, or with the reg:r_time register. The trig interrupt is generated when the values match and the selected timer (tmr0, tmr1 or r_time) is clocked and running.

25.44.10 rw_trig_cfg

Address	0x34
Default	0x00000000
Type	Read/Write
Description	Configuration register for trig point.

Bit(s)	Name	Description	Value
1-0:2	tmr	Selects which timer to compare the trig point value with. off: Trig point disabled time: Trig point compared with reg:r_time tmr0: Trig point compared with tmr0 tmr1: Trig point compared with tmr1	off=0 time=1 tmr0=2 tmr1=3

25.44.11 r_time

Address	0x38
Default	
Type	Read
Description	This register contains a 32-bit binary counter that runs continuously with a 100 MHz input clock. It starts from 0 at system reset, and counts upwards. Before the reg:r_time register has reached 65536, it may be initialized to 0xffffff00 by the en field of the rw_test register.

25.44.12 rw_out

Address	0x3c
Default	0x00000000
Type	Read/Write
Description	Selects which timer that controls the timer output. The timer output toggles each time the selected timer wraps.

Bit(s)	Name	Description	Value
1-0:2	tmr	Timer output action. off: Sets the timer output to 0 hold: Timer output holds its value tmr0: Toggle when tmr0 wraps tmr1: Toggle when tmr1 wraps	off=0 hold=1 tmr0=2 tmr1=3

25.44.13 rw_wd_ctrl

Address	0x40
Default	
Type	Read/Write
Description	Watchdog control register. While the watchdog is running, further accesses to this register will only take effect if the key value matches the bitwise inverse of the previously given key value.

Bit(s)	Name	Description	Value
15-9:7	key	This field contains a key value for the watchdog timer. The key value is stored (not readable) when the watchdog is started.	
8	cmd	Start/stop command to the watchdog timer. stop: Stop the watchdog timer start: Start/restart the watchdog timer	stop=0 start=1
7-0:8	cnt	Start value for the watchdog timer. The watchdog counts downwards from the selected start value, with a frequency of 763 Hz. When it reaches 1, it generates a non-maskable interrupt, and when it counts down to 0 it resets the chip. Setting a start value of 0 will give a count value of 256.	

25.44.14 r_wd_stat

Address	0x44
Default	
Type	Read
Description	Watchdog status register.

Bit(s)	Name	Description	Value
8	cmd	Shows the last valid command given to the watchdog. stop: Stop the watchdog timer start: Start/restart the watchdog timer	stop=0 start=1
7-0:8	cnt	The current counter value of the watchdog.	

25.44.15 rw_intr_mask

Address	0x48
Default	0x00000000
Type	Read/Write
Description	Interrupt mask. Timer interrupts. Specifies which interrupts are enabled in this subsystem. Only enabled interrupts will propagate to the central interrupt handler. In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	trig	Enable/disable trig interrupt. Interrupt from trig point. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
2	cnt	Enable/disable cnt interrupt. Interrupt from counter cnt. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
1	tmr1	Enable/disable tmr1 interrupt. Interrupt from timer tmr1. yes: Enable interrupt no: Disable interrupt	yes=1 no=0
0	tmr0	Enable/disable tmr0 interrupt. Interrupt from timer tmr0. yes: Enable interrupt no: Disable interrupt	yes=1 no=0

25.44.16 rw_ack_intr

Address	0x4c
Default	
Type	Read/Write
Description	Acknowledge interrupts. Timer interrupts.

Bit(s)	Name	Description	Value
3	trig	Acknowledge trig interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
2	cnt	Acknowledge cnt interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
1	tmr1	Acknowledge tmr1 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0
0	tmr0	Acknowledge tmr0 interrupt. yes: Acknowledge interrupt no: No operation	yes=1 no=0

25.44.17 r_intr

Address	0x50
Default	
Type	Read
Description	Interrupts before the mask. Timer interrupts. Makes it possible to determine if an interrupt is active even though it is not enabled in the mask (rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	trig	Interrupt trig active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	cnt	Interrupt cnt active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	tmr1	Interrupt tmr1 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	tmr0	Interrupt tmr0 active. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.44.18 r_masked_intr

Address	0x54
Default	
Type	Read
Description	Interrupts after the mask. Timer interrupts. Tells which interrupts are active and enabled (in rw_intr_mask). In C code the relationship between rw_intr_mask, r_intr and r_masked_intr can be expressed as: r_masked_intr = r_intr & rw_intr_mask

Bit(s)	Name	Description	Value
3	trig	Interrupt trig active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
2	cnt	Interrupt cnt active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
1	tmr1	Interrupt tmr1 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0
0	tmr0	Interrupt tmr0 active and enabled. yes: Interrupt is active no: Interrupt is inactive	yes=1 no=0

25.44.19 rw_test

Address	0x58
Default	0x00000000
Type	Read/Write
Description	Test mode register. Used to speed up testing of the read-only timer and the watchdog timer. The test mode will be permanently disabled if the dis bit is set, or when r_time reaches 65536.

Bit(s)	Name	Description	Value
1	en	Enable timer test mode. no: Normal operation mode yes: Timer test mode	no=0 yes=1
0	dis	Permanently disable test mode. Once set, this field can only be cleared by system reset. no: Test mode allowed yes: Test mode permanently disabled	no=0 yes=1

25.45 Register addresses

Address	Scope	Instance	Register
0xac000000	bif_slave_ext	bif_slave_ext	r_ch0_seq_data
0xac000004	bif_slave_ext	bif_slave_ext	r_ch0_data
0xac000008	bif_slave_ext	bif_slave_ext	rw_ch0_addr
0xac00000c	bif_slave_ext	bif_slave_ext	r_ch0_stat
0xac00000c	bif_slave_ext	bif_slave_ext	rw_ch0_ctrl
0xac000010	bif_slave_ext	bif_slave_ext	rw_ch1_seq_data
0xac000014	bif_slave_ext	bif_slave_ext	rw_ch1_data
0xac000018	bif_slave_ext	bif_slave_ext	rw_ch1_addr
0xac00001c	bif_slave_ext	bif_slave_ext	r_ch1_stat
0xac00001c	bif_slave_ext	bif_slave_ext	rw_ch1_ctrl
0xac000020	bif_slave_ext	bif_slave_ext	r_ch2_seq_data
0xac000024	bif_slave_ext	bif_slave_ext	r_ch2_data
0xac000028	bif_slave_ext	bif_slave_ext	rw_ch2_addr
0xac00002c	bif_slave_ext	bif_slave_ext	r_ch2_stat
0xac00002c	bif_slave_ext	bif_slave_ext	rw_ch2_ctrl
0xac000030	bif_slave_ext	bif_slave_ext	rw_ch3_seq_data
0xac000034	bif_slave_ext	bif_slave_ext	rw_ch3_data
0xac000038	bif_slave_ext	bif_slave_ext	rw_ch3_addr
0xac00003c	bif_slave_ext	bif_slave_ext	r_ch3_stat
0xac00003c	bif_slave_ext	bif_slave_ext	rw_ch3_ctrl
0xb0000000	dma	dma0	rw_data
0xb0000004	dma	dma0	rw_data_next
0xb0000008	dma	dma0	rw_data_buf
0xb000000c	dma	dma0	rw_data_ctrl
0xb0000010	dma	dma0	rw_data_stat
0xb0000014	dma	dma0	rw_data_md
0xb0000018	dma	dma0	rw_data_md_s
0xb000001c	dma	dma0	rw_data_after
0xb0000020	dma	dma0	rw_ctxt
0xb0000024	dma	dma0	rw_ctxt_next
0xb0000028	dma	dma0	rw_ctxt_ctrl
0xb000002c	dma	dma0	rw_ctxt_stat
0xb0000030	dma	dma0	rw_ctxt_md0
0xb0000034	dma	dma0	rw_ctxt_md0_s
0xb0000038	dma	dma0	rw_ctxt_md1
0xb000003c	dma	dma0	rw_ctxt_md1_s
0xb0000040	dma	dma0	rw_ctxt_md2
0xb0000044	dma	dma0	rw_ctxt_md2_s

0xb0000048	dma	dma0	rw_ctxt_md3
0xb000004c	dma	dma0	rw_ctxt_md3_s
0xb0000050	dma	dma0	rw_ctxt_md4
0xb0000054	dma	dma0	rw_ctxt_md4_s
0xb0000058	dma	dma0	rw_saved_data
0xb000005c	dma	dma0	rw_saved_data_buf
0xb0000060	dma	dma0	rw_group
0xb0000064	dma	dma0	rw_group_next
0xb0000068	dma	dma0	rw_group_ctrl
0xb000006c	dma	dma0	rw_group_stat
0xb0000070	dma	dma0	rw_group_md
0xb0000074	dma	dma0	rw_group_md_s
0xb0000078	dma	dma0	rw_group_up
0xb000007c	dma	dma0	rw_group_down
0xb0000080	dma	dma0	rw_cmd
0xb0000084	dma	dma0	rw_cfg
0xb0000088	dma	dma0	rw_stat
0xb000008c	dma	dma0	rw_intr_mask
0xb0000090	dma	dma0	rw_ack_intr
0xb0000094	dma	dma0	r_intr
0xb0000098	dma	dma0	r_masked_intr
0xb000009c	dma	dma0	rw_stream_cmd
0xb0002000	dma	dma1	rw_data
0xb0002004	dma	dma1	rw_data_next
0xb0002008	dma	dma1	rw_data_buf
0xb000200c	dma	dma1	rw_data_ctrl
0xb0002010	dma	dma1	rw_data_stat
0xb0002014	dma	dma1	rw_data_md
0xb0002018	dma	dma1	rw_data_md_s
0xb000201c	dma	dma1	rw_data_after
0xb0002020	dma	dma1	rw_ctxt
0xb0002024	dma	dma1	rw_ctxt_next
0xb0002028	dma	dma1	rw_ctxt_ctrl
0xb000202c	dma	dma1	rw_ctxt_stat
0xb0002030	dma	dma1	rw_ctxt_md0
0xb0002034	dma	dma1	rw_ctxt_md0_s
0xb0002038	dma	dma1	rw_ctxt_md1
0xb000203c	dma	dma1	rw_ctxt_md1_s
0xb0002040	dma	dma1	rw_ctxt_md2
0xb0002044	dma	dma1	rw_ctxt_md2_s
0xb0002048	dma	dma1	rw_ctxt_md3

0xb000204c	dma	dma1	rw_ctxt_md3_s
0xb0002050	dma	dma1	rw_ctxt_md4
0xb0002054	dma	dma1	rw_ctxt_md4_s
0xb0002058	dma	dma1	rw_saved_data
0xb000205c	dma	dma1	rw_saved_data_buf
0xb0002060	dma	dma1	rw_group
0xb0002064	dma	dma1	rw_group_next
0xb0002068	dma	dma1	rw_group_ctrl
0xb000206c	dma	dma1	rw_group_stat
0xb0002070	dma	dma1	rw_group_md
0xb0002074	dma	dma1	rw_group_md_s
0xb0002078	dma	dma1	rw_group_up
0xb000207c	dma	dma1	rw_group_down
0xb0002080	dma	dma1	rw_cmd
0xb0002084	dma	dma1	rw_cfg
0xb0002088	dma	dma1	rw_stat
0xb000208c	dma	dma1	rw_intr_mask
0xb0002090	dma	dma1	rw_ack_intr
0xb0002094	dma	dma1	r_intr
0xb0002098	dma	dma1	r_masked_intr
0xb000209c	dma	dma1	rw_stream_cmd
0xb0004000	dma	dma2	rw_data
0xb0004004	dma	dma2	rw_data_next
0xb0004008	dma	dma2	rw_data_buf
0xb000400c	dma	dma2	rw_data_ctrl
0xb0004010	dma	dma2	rw_data_stat
0xb0004014	dma	dma2	rw_data_md
0xb0004018	dma	dma2	rw_data_md_s
0xb000401c	dma	dma2	rw_data_after
0xb0004020	dma	dma2	rw_ctxt
0xb0004024	dma	dma2	rw_ctxt_next
0xb0004028	dma	dma2	rw_ctxt_ctrl
0xb000402c	dma	dma2	rw_ctxt_stat
0xb0004030	dma	dma2	rw_ctxt_md0
0xb0004034	dma	dma2	rw_ctxt_md0_s
0xb0004038	dma	dma2	rw_ctxt_md1
0xb000403c	dma	dma2	rw_ctxt_md1_s
0xb0004040	dma	dma2	rw_ctxt_md2
0xb0004044	dma	dma2	rw_ctxt_md2_s
0xb0004048	dma	dma2	rw_ctxt_md3
0xb000404c	dma	dma2	rw_ctxt_md3_s

0xb0004050	dma	dma2	rw_ctxt_md4
0xb0004054	dma	dma2	rw_ctxt_md4_s
0xb0004058	dma	dma2	rw_saved_data
0xb000405c	dma	dma2	rw_saved_data_buf
0xb0004060	dma	dma2	rw_group
0xb0004064	dma	dma2	rw_group_next
0xb0004068	dma	dma2	rw_group_ctrl
0xb000406c	dma	dma2	rw_group_stat
0xb0004070	dma	dma2	rw_group_md
0xb0004074	dma	dma2	rw_group_md_s
0xb0004078	dma	dma2	rw_group_up
0xb000407c	dma	dma2	rw_group_down
0xb0004080	dma	dma2	rw_cmd
0xb0004084	dma	dma2	rw_cfg
0xb0004088	dma	dma2	rw_stat
0xb000408c	dma	dma2	rw_intr_mask
0xb0004090	dma	dma2	rw_ack_intr
0xb0004094	dma	dma2	r_intr
0xb0004098	dma	dma2	r_masked_intr
0xb000409c	dma	dma2	rw_stream_cmd
0xb0006000	dma	dma3	rw_data
0xb0006004	dma	dma3	rw_data_next
0xb0006008	dma	dma3	rw_data_buf
0xb000600c	dma	dma3	rw_data_ctrl
0xb0006010	dma	dma3	rw_data_stat
0xb0006014	dma	dma3	rw_data_md
0xb0006018	dma	dma3	rw_data_md_s
0xb000601c	dma	dma3	rw_data_after
0xb0006020	dma	dma3	rw_ctxt
0xb0006024	dma	dma3	rw_ctxt_next
0xb0006028	dma	dma3	rw_ctxt_ctrl
0xb000602c	dma	dma3	rw_ctxt_stat
0xb0006030	dma	dma3	rw_ctxt_md0
0xb0006034	dma	dma3	rw_ctxt_md0_s
0xb0006038	dma	dma3	rw_ctxt_md1
0xb000603c	dma	dma3	rw_ctxt_md1_s
0xb0006040	dma	dma3	rw_ctxt_md2
0xb0006044	dma	dma3	rw_ctxt_md2_s
0xb0006048	dma	dma3	rw_ctxt_md3
0xb000604c	dma	dma3	rw_ctxt_md3_s
0xb0006050	dma	dma3	rw_ctxt_md4

0xb0006054	dma	dma3	rw_ctxt_md4_s
0xb0006058	dma	dma3	rw_saved_data
0xb000605c	dma	dma3	rw_saved_data_buf
0xb0006060	dma	dma3	rw_group
0xb0006064	dma	dma3	rw_group_next
0xb0006068	dma	dma3	rw_group_ctrl
0xb000606c	dma	dma3	rw_group_stat
0xb0006070	dma	dma3	rw_group_md
0xb0006074	dma	dma3	rw_group_md_s
0xb0006078	dma	dma3	rw_group_up
0xb000607c	dma	dma3	rw_group_down
0xb0006080	dma	dma3	rw_cmd
0xb0006084	dma	dma3	rw_cfg
0xb0006088	dma	dma3	rw_stat
0xb000608c	dma	dma3	rw_intr_mask
0xb0006090	dma	dma3	rw_ack_intr
0xb0006094	dma	dma3	r_intr
0xb0006098	dma	dma3	r_masked_intr
0xb000609c	dma	dma3	rw_stream_cmd
0xb0008000	dma	dma4	rw_data
0xb0008004	dma	dma4	rw_data_next
0xb0008008	dma	dma4	rw_data_buf
0xb000800c	dma	dma4	rw_data_ctrl
0xb0008010	dma	dma4	rw_data_stat
0xb0008014	dma	dma4	rw_data_md
0xb0008018	dma	dma4	rw_data_md_s
0xb000801c	dma	dma4	rw_data_after
0xb0008020	dma	dma4	rw_ctxt
0xb0008024	dma	dma4	rw_ctxt_next
0xb0008028	dma	dma4	rw_ctxt_ctrl
0xb000802c	dma	dma4	rw_ctxt_stat
0xb0008030	dma	dma4	rw_ctxt_md0
0xb0008034	dma	dma4	rw_ctxt_md0_s
0xb0008038	dma	dma4	rw_ctxt_md1
0xb000803c	dma	dma4	rw_ctxt_md1_s
0xb0008040	dma	dma4	rw_ctxt_md2
0xb0008044	dma	dma4	rw_ctxt_md2_s
0xb0008048	dma	dma4	rw_ctxt_md3
0xb000804c	dma	dma4	rw_ctxt_md3_s
0xb0008050	dma	dma4	rw_ctxt_md4
0xb0008054	dma	dma4	rw_ctxt_md4_s

0xb0008058	dma	dma4	rw_saved_data
0xb000805c	dma	dma4	rw_saved_data_buf
0xb0008060	dma	dma4	rw_group
0xb0008064	dma	dma4	rw_group_next
0xb0008068	dma	dma4	rw_group_ctrl
0xb000806c	dma	dma4	rw_group_stat
0xb0008070	dma	dma4	rw_group_md
0xb0008074	dma	dma4	rw_group_md_s
0xb0008078	dma	dma4	rw_group_up
0xb000807c	dma	dma4	rw_group_down
0xb0008080	dma	dma4	rw_cmd
0xb0008084	dma	dma4	rw_cfg
0xb0008088	dma	dma4	rw_stat
0xb000808c	dma	dma4	rw_intr_mask
0xb0008090	dma	dma4	rw_ack_intr
0xb0008094	dma	dma4	r_intr
0xb0008098	dma	dma4	r_masked_intr
0xb000809c	dma	dma4	rw_stream_cmd
0xb000a000	dma	dma5	rw_data
0xb000a004	dma	dma5	rw_data_next
0xb000a008	dma	dma5	rw_data_buf
0xb000a00c	dma	dma5	rw_data_ctrl
0xb000a010	dma	dma5	rw_data_stat
0xb000a014	dma	dma5	rw_data_md
0xb000a018	dma	dma5	rw_data_md_s
0xb000a01c	dma	dma5	rw_data_after
0xb000a020	dma	dma5	rw_ctxt
0xb000a024	dma	dma5	rw_ctxt_next
0xb000a028	dma	dma5	rw_ctxt_ctrl
0xb000a02c	dma	dma5	rw_ctxt_stat
0xb000a030	dma	dma5	rw_ctxt_md0
0xb000a034	dma	dma5	rw_ctxt_md0_s
0xb000a038	dma	dma5	rw_ctxt_md1
0xb000a03c	dma	dma5	rw_ctxt_md1_s
0xb000a040	dma	dma5	rw_ctxt_md2
0xb000a044	dma	dma5	rw_ctxt_md2_s
0xb000a048	dma	dma5	rw_ctxt_md3
0xb000a04c	dma	dma5	rw_ctxt_md3_s
0xb000a050	dma	dma5	rw_ctxt_md4
0xb000a054	dma	dma5	rw_ctxt_md4_s
0xb000a058	dma	dma5	rw_saved_data

0xb000a05c	dma	dma5	rw_saved_data_buf
0xb000a060	dma	dma5	rw_group
0xb000a064	dma	dma5	rw_group_next
0xb000a068	dma	dma5	rw_group_ctrl
0xb000a06c	dma	dma5	rw_group_stat
0xb000a070	dma	dma5	rw_group_md
0xb000a074	dma	dma5	rw_group_md_s
0xb000a078	dma	dma5	rw_group_up
0xb000a07c	dma	dma5	rw_group_down
0xb000a080	dma	dma5	rw_cmd
0xb000a084	dma	dma5	rw_cfg
0xb000a088	dma	dma5	rw_stat
0xb000a08c	dma	dma5	rw_intr_mask
0xb000a090	dma	dma5	rw_ack_intr
0xb000a094	dma	dma5	r_intr
0xb000a098	dma	dma5	r_masked_intr
0xb000a09c	dma	dma5	rw_stream_cmd
0xb000c000	dma	dma6	rw_data
0xb000c004	dma	dma6	rw_data_next
0xb000c008	dma	dma6	rw_data_buf
0xb000c00c	dma	dma6	rw_data_ctrl
0xb000c010	dma	dma6	rw_data_stat
0xb000c014	dma	dma6	rw_data_md
0xb000c018	dma	dma6	rw_data_md_s
0xb000c01c	dma	dma6	rw_data_after
0xb000c020	dma	dma6	rw_ctxt
0xb000c024	dma	dma6	rw_ctxt_next
0xb000c028	dma	dma6	rw_ctxt_ctrl
0xb000c02c	dma	dma6	rw_ctxt_stat
0xb000c030	dma	dma6	rw_ctxt_md0
0xb000c034	dma	dma6	rw_ctxt_md0_s
0xb000c038	dma	dma6	rw_ctxt_md1
0xb000c03c	dma	dma6	rw_ctxt_md1_s
0xb000c040	dma	dma6	rw_ctxt_md2
0xb000c044	dma	dma6	rw_ctxt_md2_s
0xb000c048	dma	dma6	rw_ctxt_md3
0xb000c04c	dma	dma6	rw_ctxt_md3_s
0xb000c050	dma	dma6	rw_ctxt_md4
0xb000c054	dma	dma6	rw_ctxt_md4_s
0xb000c058	dma	dma6	rw_saved_data
0xb000c05c	dma	dma6	rw_saved_data_buf

0xb000c060	dma	dma6	rw_group
0xb000c064	dma	dma6	rw_group_next
0xb000c068	dma	dma6	rw_group_ctrl
0xb000c06c	dma	dma6	rw_group_stat
0xb000c070	dma	dma6	rw_group_md
0xb000c074	dma	dma6	rw_group_md_s
0xb000c078	dma	dma6	rw_group_up
0xb000c07c	dma	dma6	rw_group_down
0xb000c080	dma	dma6	rw_cmd
0xb000c084	dma	dma6	rw_cfg
0xb000c088	dma	dma6	rw_stat
0xb000c08c	dma	dma6	rw_intr_mask
0xb000c090	dma	dma6	rw_ack_intr
0xb000c094	dma	dma6	r_intr
0xb000c098	dma	dma6	r_masked_intr
0xb000c09c	dma	dma6	rw_stream_cmd
0xb000e000	dma	dma7	rw_data
0xb000e004	dma	dma7	rw_data_next
0xb000e008	dma	dma7	rw_data_buf
0xb000e00c	dma	dma7	rw_data_ctrl
0xb000e010	dma	dma7	rw_data_stat
0xb000e014	dma	dma7	rw_data_md
0xb000e018	dma	dma7	rw_data_md_s
0xb000e01c	dma	dma7	rw_data_after
0xb000e020	dma	dma7	rw_ctxt
0xb000e024	dma	dma7	rw_ctxt_next
0xb000e028	dma	dma7	rw_ctxt_ctrl
0xb000e02c	dma	dma7	rw_ctxt_stat
0xb000e030	dma	dma7	rw_ctxt_md0
0xb000e034	dma	dma7	rw_ctxt_md0_s
0xb000e038	dma	dma7	rw_ctxt_md1
0xb000e03c	dma	dma7	rw_ctxt_md1_s
0xb000e040	dma	dma7	rw_ctxt_md2
0xb000e044	dma	dma7	rw_ctxt_md2_s
0xb000e048	dma	dma7	rw_ctxt_md3
0xb000e04c	dma	dma7	rw_ctxt_md3_s
0xb000e050	dma	dma7	rw_ctxt_md4
0xb000e054	dma	dma7	rw_ctxt_md4_s
0xb000e058	dma	dma7	rw_saved_data
0xb000e05c	dma	dma7	rw_saved_data_buf
0xb000e060	dma	dma7	rw_group

0xb000e064	dma	dma7	rw_group_next
0xb000e068	dma	dma7	rw_group_ctrl
0xb000e06c	dma	dma7	rw_group_stat
0xb000e070	dma	dma7	rw_group_md
0xb000e074	dma	dma7	rw_group_md_s
0xb000e078	dma	dma7	rw_group_up
0xb000e07c	dma	dma7	rw_group_down
0xb000e080	dma	dma7	rw_cmd
0xb000e084	dma	dma7	rw_cfg
0xb000e088	dma	dma7	rw_stat
0xb000e08c	dma	dma7	rw_intr_mask
0xb000e090	dma	dma7	rw_ack_intr
0xb000e094	dma	dma7	r_intr
0xb000e098	dma	dma7	r_masked_intr
0xb000e09c	dma	dma7	rw_stream_cmd
0xb0010000	dma	dma8	rw_data
0xb0010004	dma	dma8	rw_data_next
0xb0010008	dma	dma8	rw_data_buf
0xb001000c	dma	dma8	rw_data_ctrl
0xb0010010	dma	dma8	rw_data_stat
0xb0010014	dma	dma8	rw_data_md
0xb0010018	dma	dma8	rw_data_md_s
0xb001001c	dma	dma8	rw_data_after
0xb0010020	dma	dma8	rw_ctxt
0xb0010024	dma	dma8	rw_ctxt_next
0xb0010028	dma	dma8	rw_ctxt_ctrl
0xb001002c	dma	dma8	rw_ctxt_stat
0xb0010030	dma	dma8	rw_ctxt_md0
0xb0010034	dma	dma8	rw_ctxt_md0_s
0xb0010038	dma	dma8	rw_ctxt_md1
0xb001003c	dma	dma8	rw_ctxt_md1_s
0xb0010040	dma	dma8	rw_ctxt_md2
0xb0010044	dma	dma8	rw_ctxt_md2_s
0xb0010048	dma	dma8	rw_ctxt_md3
0xb001004c	dma	dma8	rw_ctxt_md3_s
0xb0010050	dma	dma8	rw_ctxt_md4
0xb0010054	dma	dma8	rw_ctxt_md4_s
0xb0010058	dma	dma8	rw_saved_data
0xb001005c	dma	dma8	rw_saved_data_buf
0xb0010060	dma	dma8	rw_group
0xb0010064	dma	dma8	rw_group_next

0xb0010068	dma	dma8	rw_group_ctrl
0xb001006c	dma	dma8	rw_group_stat
0xb0010070	dma	dma8	rw_group_md
0xb0010074	dma	dma8	rw_group_md_s
0xb0010078	dma	dma8	rw_group_up
0xb001007c	dma	dma8	rw_group_down
0xb0010080	dma	dma8	rw_cmd
0xb0010084	dma	dma8	rw_cfg
0xb0010088	dma	dma8	rw_stat
0xb001008c	dma	dma8	rw_intr_mask
0xb0010090	dma	dma8	rw_ack_intr
0xb0010094	dma	dma8	r_intr
0xb0010098	dma	dma8	r_masked_intr
0xb001009c	dma	dma8	rw_stream_cmd
0xb0012000	dma	dma9	rw_data
0xb0012004	dma	dma9	rw_data_next
0xb0012008	dma	dma9	rw_data_buf
0xb001200c	dma	dma9	rw_data_ctrl
0xb0012010	dma	dma9	rw_data_stat
0xb0012014	dma	dma9	rw_data_md
0xb0012018	dma	dma9	rw_data_md_s
0xb001201c	dma	dma9	rw_data_after
0xb0012020	dma	dma9	rw_ctxt
0xb0012024	dma	dma9	rw_ctxt_next
0xb0012028	dma	dma9	rw_ctxt_ctrl
0xb001202c	dma	dma9	rw_ctxt_stat
0xb0012030	dma	dma9	rw_ctxt_md0
0xb0012034	dma	dma9	rw_ctxt_md0_s
0xb0012038	dma	dma9	rw_ctxt_md1
0xb001203c	dma	dma9	rw_ctxt_md1_s
0xb0012040	dma	dma9	rw_ctxt_md2
0xb0012044	dma	dma9	rw_ctxt_md2_s
0xb0012048	dma	dma9	rw_ctxt_md3
0xb001204c	dma	dma9	rw_ctxt_md3_s
0xb0012050	dma	dma9	rw_ctxt_md4
0xb0012054	dma	dma9	rw_ctxt_md4_s
0xb0012058	dma	dma9	rw_saved_data
0xb001205c	dma	dma9	rw_saved_data_buf
0xb0012060	dma	dma9	rw_group
0xb0012064	dma	dma9	rw_group_next
0xb0012068	dma	dma9	rw_group_ctrl

0xb001206c	dma	dma9	rw_group_stat
0xb0012070	dma	dma9	rw_group_md
0xb0012074	dma	dma9	rw_group_md_s
0xb0012078	dma	dma9	rw_group_up
0xb001207c	dma	dma9	rw_group_down
0xb0012080	dma	dma9	rw_cmd
0xb0012084	dma	dma9	rw_cfg
0xb0012088	dma	dma9	rw_stat
0xb001208c	dma	dma9	rw_intr_mask
0xb0012090	dma	dma9	rw_ack_intr
0xb0012094	dma	dma9	r_intr
0xb0012098	dma	dma9	r_masked_intr
0xb001209c	dma	dma9	rw_stream_cmd
0xb0014000	bif_core	bif_core	rw_grp1_cfg
0xb0014004	bif_core	bif_core	rw_grp2_cfg
0xb0014008	bif_core	bif_core	rw_grp3_cfg
0xb001400c	bif_core	bif_core	rw_grp4_cfg
0xb0014010	bif_core	bif_core	rw_sdram_cfg_grp0
0xb0014014	bif_core	bif_core	rw_sdram_cfg_grp1
0xb0014018	bif_core	bif_core	rw_sdram_timing
0xb001401c	bif_core	bif_core	rw_sdram_cmd
0xb0014020	bif_core	bif_core	rs_sdram_ref_stat
0xb0014024	bif_core	bif_core	r_sdram_ref_stat
0xb0016000	bif_dma	bif_dma	rw_ch0_ctrl
0xb0016004	bif_dma	bif_dma	rw_ch0_addr
0xb0016008	bif_dma	bif_dma	rw_ch0_start
0xb001600c	bif_dma	bif_dma	rw_ch0_cnt
0xb0016010	bif_dma	bif_dma	r_ch0_stat
0xb0016020	bif_dma	bif_dma	rw_ch1_ctrl
0xb0016024	bif_dma	bif_dma	rw_ch1_addr
0xb0016028	bif_dma	bif_dma	rw_ch1_start
0xb001602c	bif_dma	bif_dma	rw_ch1_cnt
0xb0016030	bif_dma	bif_dma	r_ch1_stat
0xb0016040	bif_dma	bif_dma	rw_ch2_ctrl
0xb0016044	bif_dma	bif_dma	rw_ch2_addr
0xb0016048	bif_dma	bif_dma	rw_ch2_start
0xb001604c	bif_dma	bif_dma	rw_ch2_cnt
0xb0016050	bif_dma	bif_dma	r_ch2_stat
0xb0016060	bif_dma	bif_dma	rw_ch3_ctrl
0xb0016064	bif_dma	bif_dma	rw_ch3_addr
0xb0016068	bif_dma	bif_dma	rw_ch3_start

0xb001606c	bif_dma	bif_dma	rw_ch3_cnt
0xb0016070	bif_dma	bif_dma	r_ch3_stat
0xb0016080	bif_dma	bif_dma	rw_intr_mask
0xb0016084	bif_dma	bif_dma	rw_ack_intr
0xb0016088	bif_dma	bif_dma	r_intr
0xb001608c	bif_dma	bif_dma	r_masked_intr
0xb00160a0	bif_dma	bif_dma	rw_pin0_cfg
0xb00160a4	bif_dma	bif_dma	rw_pin1_cfg
0xb00160a8	bif_dma	bif_dma	rw_pin2_cfg
0xb00160ac	bif_dma	bif_dma	rw_pin3_cfg
0xb00160b0	bif_dma	bif_dma	rw_pin4_cfg
0xb00160b4	bif_dma	bif_dma	rw_pin5_cfg
0xb00160b8	bif_dma	bif_dma	rw_pin6_cfg
0xb00160bc	bif_dma	bif_dma	rw_pin7_cfg
0xb00160c0	bif_dma	bif_dma	r_pin_stat
0xb0018000	bif_slave	bif_slave	rw_slave_cfg
0xb0018004	bif_slave	bif_slave	r_slave_mode
0xb0018010	bif_slave	bif_slave	rw_ch0_cfg
0xb0018014	bif_slave	bif_slave	rw_ch1_cfg
0xb0018018	bif_slave	bif_slave	rw_ch2_cfg
0xb001801c	bif_slave	bif_slave	rw_ch3_cfg
0xb0018020	bif_slave	bif_slave	rw_arb_cfg
0xb0018024	bif_slave	bif_slave	r_arb_stat
0xb0018040	bif_slave	bif_slave	rw_intr_mask
0xb0018044	bif_slave	bif_slave	rw_ack_intr
0xb0018048	bif_slave	bif_slave	r_intr
0xb001804c	bif_slave	bif_slave	r_masked_intr
0xb001a000	gio	gio	rw_pa_dout
0xb001a004	gio	gio	r_pa_din
0xb001a008	gio	gio	rw_pa_oe
0xb001a00c	gio	gio	rw_intr_cfg
0xb001a010	gio	gio	rw_intr_mask
0xb001a014	gio	gio	rw_ack_intr
0xb001a018	gio	gio	r_intr
0xb001a01c	gio	gio	r_masked_intr
0xb001a020	gio	gio	rw_pb_dout
0xb001a024	gio	gio	r_pb_din
0xb001a028	gio	gio	rw_pb_oe
0xb001a030	gio	gio	rw_pc_dout
0xb001a034	gio	gio	r_pc_din
0xb001a038	gio	gio	rw_pc_oe

0xb001a040	gio	gio	rw_pd_dout
0xb001a044	gio	gio	r_pd_din
0xb001a048	gio	gio	rw_pd_oe
0xb001a050	gio	gio	rw_pe_dout
0xb001a054	gio	gio	r_pe_din
0xb001a058	gio	gio	rw_pe_oe
0xb001c000	intr_vect	irq	rw_mask
0xb001c004	intr_vect	irq	r_vect
0xb001c008	intr_vect	irq	r_masked_vect
0xb001c00c	intr_vect	irq	r_nmi
0xb001c010	intr_vect	irq	r_guru
0xb001e000	timer	timer	rw_tmr0_div
0xb001e004	timer	timer	r_tmr0_data
0xb001e008	timer	timer	rw_tmr0_ctrl
0xb001e010	timer	timer	rw_tmr1_div
0xb001e014	timer	timer	r_tmr1_data
0xb001e018	timer	timer	rw_tmr1_ctrl
0xb001e020	timer	timer	rs_cnt_data
0xb001e024	timer	timer	r_cnt_data
0xb001e028	timer	timer	rw_cnt_cfg
0xb001e030	timer	timer	rw_trig
0xb001e034	timer	timer	rw_trig_cfg
0xb001e038	timer	timer	r_time
0xb001e03c	timer	timer	rw_out
0xb001e040	timer	timer	rw_wd_ctrl
0xb001e044	timer	timer	r_wd_stat
0xb001e048	timer	timer	rw_intr_mask
0xb001e04c	timer	timer	rw_ack_intr
0xb001e050	timer	timer	r_intr
0xb001e054	timer	timer	r_masked_intr
0xb001e058	timer	timer	rw_test
0xb0020000	iop_version	iop_version	r_version
0xb0020040	iop_fifo_in_extra	iop_fifo_in0_extra	rw_wr_data
0xb0020044	iop_fifo_in_extra	iop_fifo_in0_extra	r_stat
0xb0020048	iop_fifo_in_extra	iop_fifo_in0_extra	rw_strb_dif_in
0xb002004c	iop_fifo_in_extra	iop_fifo_in0_extra	rw_intr_mask
0xb0020050	iop_fifo_in_extra	iop_fifo_in0_extra	rw_ack_intr
0xb0020054	iop_fifo_in_extra	iop_fifo_in0_extra	r_intr
0xb0020058	iop_fifo_in_extra	iop_fifo_in0_extra	r_masked_intr
0xb0020080	iop_fifo_in_extra	iop_fifo_in1_extra	rw_wr_data
0xb0020084	iop_fifo_in_extra	iop_fifo_in1_extra	r_stat

0xb0020088	iop_fifo_in_extra	iop_fifo_in1_extra	rw_strb_dif_in
0xb002008c	iop_fifo_in_extra	iop_fifo_in1_extra	rw_intr_mask
0xb0020090	iop_fifo_in_extra	iop_fifo_in1_extra	rw_ack_intr
0xb0020094	iop_fifo_in_extra	iop_fifo_in1_extra	r_intr
0xb0020098	iop_fifo_in_extra	iop_fifo_in1_extra	r_masked_intr
0xb00200c0	iop_fifo_out_extra	iop_fifo_out0_extra	rs_rd_data
0xb00200c4	iop_fifo_out_extra	iop_fifo_out0_extra	r_rd_data
0xb00200c8	iop_fifo_out_extra	iop_fifo_out0_extra	r_stat
0xb00200cc	iop_fifo_out_extra	iop_fifo_out0_extra	rw_strb_dif_out
0xb00200d0	iop_fifo_out_extra	iop_fifo_out0_extra	rw_intr_mask
0xb00200d4	iop_fifo_out_extra	iop_fifo_out0_extra	rw_ack_intr
0xb00200d8	iop_fifo_out_extra	iop_fifo_out0_extra	r_intr
0xb00200dc	iop_fifo_out_extra	iop_fifo_out0_extra	r_masked_intr
0xb0020100	iop_fifo_out_extra	iop_fifo_out1_extra	rs_rd_data
0xb0020104	iop_fifo_out_extra	iop_fifo_out1_extra	r_rd_data
0xb0020108	iop_fifo_out_extra	iop_fifo_out1_extra	r_stat
0xb002010c	iop_fifo_out_extra	iop_fifo_out1_extra	rw_strb_dif_out
0xb0020110	iop_fifo_out_extra	iop_fifo_out1_extra	rw_intr_mask
0xb0020114	iop_fifo_out_extra	iop_fifo_out1_extra	rw_ack_intr
0xb0020118	iop_fifo_out_extra	iop_fifo_out1_extra	r_intr
0xb002011c	iop_fifo_out_extra	iop_fifo_out1_extra	r_masked_intr
0xb0020140	iop_trigger_grp	iop_trigger_grp0	rw_cfg
0xb0020144	iop_trigger_grp	iop_trigger_grp0	rw_cfg
0xb0020148	iop_trigger_grp	iop_trigger_grp0	rw_cfg
0xb002014c	iop_trigger_grp	iop_trigger_grp0	rw_cfg
0xb0020150	iop_trigger_grp	iop_trigger_grp0	rw_cmd
0xb0020154	iop_trigger_grp	iop_trigger_grp0	rw_intr_mask
0xb0020158	iop_trigger_grp	iop_trigger_grp0	rw_ack_intr
0xb002015c	iop_trigger_grp	iop_trigger_grp0	r_intr
0xb0020160	iop_trigger_grp	iop_trigger_grp0	r_masked_intr
0xb0020180	iop_trigger_grp	iop_trigger_grp1	rw_cfg
0xb0020184	iop_trigger_grp	iop_trigger_grp1	rw_cfg
0xb0020188	iop_trigger_grp	iop_trigger_grp1	rw_cfg
0xb002018c	iop_trigger_grp	iop_trigger_grp1	rw_cfg
0xb0020190	iop_trigger_grp	iop_trigger_grp1	rw_cmd
0xb0020194	iop_trigger_grp	iop_trigger_grp1	rw_intr_mask
0xb0020198	iop_trigger_grp	iop_trigger_grp1	rw_ack_intr
0xb002019c	iop_trigger_grp	iop_trigger_grp1	r_intr
0xb00201a0	iop_trigger_grp	iop_trigger_grp1	r_masked_intr
0xb00201c0	iop_trigger_grp	iop_trigger_grp2	rw_cfg
0xb00201c4	iop_trigger_grp	iop_trigger_grp2	rw_cfg

0xb00201c8	iop_trigger_grp	iop_trigger_grp2	rw_cfg
0xb00201cc	iop_trigger_grp	iop_trigger_grp2	rw_cfg
0xb00201d0	iop_trigger_grp	iop_trigger_grp2	rw_cmd
0xb00201d4	iop_trigger_grp	iop_trigger_grp2	rw_intr_mask
0xb00201d8	iop_trigger_grp	iop_trigger_grp2	rw_ack_intr
0xb00201dc	iop_trigger_grp	iop_trigger_grp2	r_intr
0xb00201e0	iop_trigger_grp	iop_trigger_grp2	r_masked_intr
0xb0020200	iop_trigger_grp	iop_trigger_grp3	rw_cfg
0xb0020204	iop_trigger_grp	iop_trigger_grp3	rw_cfg
0xb0020208	iop_trigger_grp	iop_trigger_grp3	rw_cfg
0xb002020c	iop_trigger_grp	iop_trigger_grp3	rw_cfg
0xb0020210	iop_trigger_grp	iop_trigger_grp3	rw_cmd
0xb0020214	iop_trigger_grp	iop_trigger_grp3	rw_intr_mask
0xb0020218	iop_trigger_grp	iop_trigger_grp3	rw_ack_intr
0xb002021c	iop_trigger_grp	iop_trigger_grp3	r_intr
0xb0020220	iop_trigger_grp	iop_trigger_grp3	r_masked_intr
0xb0020240	iop_trigger_grp	iop_trigger_grp4	rw_cfg
0xb0020244	iop_trigger_grp	iop_trigger_grp4	rw_cfg
0xb0020248	iop_trigger_grp	iop_trigger_grp4	rw_cfg
0xb002024c	iop_trigger_grp	iop_trigger_grp4	rw_cfg
0xb0020250	iop_trigger_grp	iop_trigger_grp4	rw_cmd
0xb0020254	iop_trigger_grp	iop_trigger_grp4	rw_intr_mask
0xb0020258	iop_trigger_grp	iop_trigger_grp4	rw_ack_intr
0xb002025c	iop_trigger_grp	iop_trigger_grp4	r_intr
0xb0020260	iop_trigger_grp	iop_trigger_grp4	r_masked_intr
0xb0020280	iop_trigger_grp	iop_trigger_grp5	rw_cfg
0xb0020284	iop_trigger_grp	iop_trigger_grp5	rw_cfg
0xb0020288	iop_trigger_grp	iop_trigger_grp5	rw_cfg
0xb002028c	iop_trigger_grp	iop_trigger_grp5	rw_cfg
0xb0020290	iop_trigger_grp	iop_trigger_grp5	rw_cmd
0xb0020294	iop_trigger_grp	iop_trigger_grp5	rw_intr_mask
0xb0020298	iop_trigger_grp	iop_trigger_grp5	rw_ack_intr
0xb002029c	iop_trigger_grp	iop_trigger_grp5	r_intr
0xb00202a0	iop_trigger_grp	iop_trigger_grp5	r_masked_intr
0xb00202c0	iop_trigger_grp	iop_trigger_grp6	rw_cfg
0xb00202c4	iop_trigger_grp	iop_trigger_grp6	rw_cfg
0xb00202c8	iop_trigger_grp	iop_trigger_grp6	rw_cfg
0xb00202cc	iop_trigger_grp	iop_trigger_grp6	rw_cfg
0xb00202d0	iop_trigger_grp	iop_trigger_grp6	rw_cmd
0xb00202d4	iop_trigger_grp	iop_trigger_grp6	rw_intr_mask
0xb00202d8	iop_trigger_grp	iop_trigger_grp6	rw_ack_intr

0xb00202dc	iop_trigger_grp	iop_trigger_grp6	r_intr
0xb00202e0	iop_trigger_grp	iop_trigger_grp6	r_masked_intr
0xb0020300	iop_trigger_grp	iop_trigger_grp7	rw_cfg
0xb0020304	iop_trigger_grp	iop_trigger_grp7	rw_cfg
0xb0020308	iop_trigger_grp	iop_trigger_grp7	rw_cfg
0xb002030c	iop_trigger_grp	iop_trigger_grp7	rw_cfg
0xb0020310	iop_trigger_grp	iop_trigger_grp7	rw_cmd
0xb0020314	iop_trigger_grp	iop_trigger_grp7	rw_intr_mask
0xb0020318	iop_trigger_grp	iop_trigger_grp7	rw_ack_intr
0xb002031c	iop_trigger_grp	iop_trigger_grp7	r_intr
0xb0020320	iop_trigger_grp	iop_trigger_grp7	r_masked_intr
0xb0020380	iop_crc_par	iop_crc_par0	rw_cfg
0xb0020384	iop_crc_par	iop_crc_par0	rw_init_crc
0xb0020388	iop_crc_par	iop_crc_par0	rw_correct_crc
0xb002038c	iop_crc_par	iop_crc_par0	rw_ctrl
0xb0020390	iop_crc_par	iop_crc_par0	rw_set_last
0xb0020394	iop_crc_par	iop_crc_par0	rw_wr1byte
0xb0020398	iop_crc_par	iop_crc_par0	rw_wr2byte
0xb002039c	iop_crc_par	iop_crc_par0	rw_wr3byte
0xb00203a0	iop_crc_par	iop_crc_par0	rw_wr4byte
0xb00203a4	iop_crc_par	iop_crc_par0	rw_wr1byte_last
0xb00203a8	iop_crc_par	iop_crc_par0	rw_wr2byte_last
0xb00203ac	iop_crc_par	iop_crc_par0	rw_wr3byte_last
0xb00203b0	iop_crc_par	iop_crc_par0	rw_wr4byte_last
0xb00203b4	iop_crc_par	iop_crc_par0	r_stat
0xb00203b8	iop_crc_par	iop_crc_par0	r_sh_reg
0xb00203bc	iop_crc_par	iop_crc_par0	r_crc
0xb00203c0	iop_crc_par	iop_crc_par0	rw_strb_rec_dif_in
0xb0020400	iop_crc_par	iop_crc_par1	rw_cfg
0xb0020404	iop_crc_par	iop_crc_par1	rw_init_crc
0xb0020408	iop_crc_par	iop_crc_par1	rw_correct_crc
0xb002040c	iop_crc_par	iop_crc_par1	rw_ctrl
0xb0020410	iop_crc_par	iop_crc_par1	rw_set_last
0xb0020414	iop_crc_par	iop_crc_par1	rw_wr1byte
0xb0020418	iop_crc_par	iop_crc_par1	rw_wr2byte
0xb002041c	iop_crc_par	iop_crc_par1	rw_wr3byte
0xb0020420	iop_crc_par	iop_crc_par1	rw_wr4byte
0xb0020424	iop_crc_par	iop_crc_par1	rw_wr1byte_last
0xb0020428	iop_crc_par	iop_crc_par1	rw_wr2byte_last
0xb002042c	iop_crc_par	iop_crc_par1	rw_wr3byte_last
0xb0020430	iop_crc_par	iop_crc_par1	rw_wr4byte_last

0xb0020434	iop_crc_par	iop_crc_par1	r_stat
0xb0020438	iop_crc_par	iop_crc_par1	r_sh_reg
0xb002043c	iop_crc_par	iop_crc_par1	r_crc
0xb0020440	iop_crc_par	iop_crc_par1	rw_strb_rec_dif_in
0xb0020480	iop_dmc_in	iop_dmc_in0	rw_cfg
0xb0020484	iop_dmc_in	iop_dmc_in0	rw_ctrl
0xb0020488	iop_dmc_in	iop_dmc_in0	r_stat
0xb002048c	iop_dmc_in	iop_dmc_in0	rw_stream_cmd
0xb0020490	iop_dmc_in	iop_dmc_in0	rw_stream_wr_data
0xb0020494	iop_dmc_in	iop_dmc_in0	rw_stream_wr_data_last
0xb0020498	iop_dmc_in	iop_dmc_in0	rw_stream_ctrl
0xb002049c	iop_dmc_in	iop_dmc_in0	r_stream_stat
0xb00204a0	iop_dmc_in	iop_dmc_in0	r_data_descr
0xb00204a4	iop_dmc_in	iop_dmc_in0	r_ctxt_descr
0xb00204a8	iop_dmc_in	iop_dmc_in0	r_ctxt_descr_md1
0xb00204ac	iop_dmc_in	iop_dmc_in0	r_ctxt_descr_md2
0xb00204b8	iop_dmc_in	iop_dmc_in0	r_group_descr
0xb00204bc	iop_dmc_in	iop_dmc_in0	rw_data_descr
0xb00204c0	iop_dmc_in	iop_dmc_in0	rw_ctxt_descr
0xb00204c4	iop_dmc_in	iop_dmc_in0	rw_ctxt_descr_md1
0xb00204c8	iop_dmc_in	iop_dmc_in0	rw_ctxt_descr_md2
0xb00204d4	iop_dmc_in	iop_dmc_in0	rw_group_descr
0xb00204d8	iop_dmc_in	iop_dmc_in0	rw_intr_mask
0xb00204dc	iop_dmc_in	iop_dmc_in0	rw_ack_intr
0xb00204e0	iop_dmc_in	iop_dmc_in0	r_intr
0xb00204e4	iop_dmc_in	iop_dmc_in0	r_masked_intr
0xb0020500	iop_dmc_in	iop_dmc_in1	rw_cfg
0xb0020504	iop_dmc_in	iop_dmc_in1	rw_ctrl
0xb0020508	iop_dmc_in	iop_dmc_in1	r_stat
0xb002050c	iop_dmc_in	iop_dmc_in1	rw_stream_cmd
0xb0020510	iop_dmc_in	iop_dmc_in1	rw_stream_wr_data
0xb0020514	iop_dmc_in	iop_dmc_in1	rw_stream_wr_data_last
0xb0020518	iop_dmc_in	iop_dmc_in1	rw_stream_ctrl
0xb002051c	iop_dmc_in	iop_dmc_in1	r_stream_stat
0xb0020520	iop_dmc_in	iop_dmc_in1	r_data_descr
0xb0020524	iop_dmc_in	iop_dmc_in1	r_ctxt_descr
0xb0020528	iop_dmc_in	iop_dmc_in1	r_ctxt_descr_md1
0xb002052c	iop_dmc_in	iop_dmc_in1	r_ctxt_descr_md2
0xb0020538	iop_dmc_in	iop_dmc_in1	r_group_descr
0xb002053c	iop_dmc_in	iop_dmc_in1	rw_data_descr
0xb0020540	iop_dmc_in	iop_dmc_in1	rw_ctxt_descr

0xb0020544	iop_dmc_in	iop_dmc_in1	rw_ctxt_descr_md1
0xb0020548	iop_dmc_in	iop_dmc_in1	rw_ctxt_descr_md2
0xb0020554	iop_dmc_in	iop_dmc_in1	rw_group_descr
0xb0020558	iop_dmc_in	iop_dmc_in1	rw_intr_mask
0xb002055c	iop_dmc_in	iop_dmc_in1	rw_ack_intr
0xb0020560	iop_dmc_in	iop_dmc_in1	r_intr
0xb0020564	iop_dmc_in	iop_dmc_in1	r_masked_intr
0xb0020580	iop_dmc_out	iop_dmc_out0	rw_cfg
0xb0020584	iop_dmc_out	iop_dmc_out0	rw_ctrl
0xb0020588	iop_dmc_out	iop_dmc_out0	r_stat
0xb002058c	iop_dmc_out	iop_dmc_out0	rw_stream_cmd
0xb0020590	iop_dmc_out	iop_dmc_out0	rs_stream_data
0xb0020594	iop_dmc_out	iop_dmc_out0	r_stream_data
0xb0020598	iop_dmc_out	iop_dmc_out0	r_stream_stat
0xb002059c	iop_dmc_out	iop_dmc_out0	r_data_descr
0xb00205a0	iop_dmc_out	iop_dmc_out0	r_ctxt_descr
0xb00205a4	iop_dmc_out	iop_dmc_out0	r_ctxt_descr_md1
0xb00205a8	iop_dmc_out	iop_dmc_out0	r_ctxt_descr_md2
0xb00205b4	iop_dmc_out	iop_dmc_out0	r_group_descr
0xb00205b8	iop_dmc_out	iop_dmc_out0	rw_data_descr
0xb00205bc	iop_dmc_out	iop_dmc_out0	rw_ctxt_descr
0xb00205c0	iop_dmc_out	iop_dmc_out0	rw_ctxt_descr_md1
0xb00205c4	iop_dmc_out	iop_dmc_out0	rw_ctxt_descr_md2
0xb00205d0	iop_dmc_out	iop_dmc_out0	rw_group_descr
0xb00205d4	iop_dmc_out	iop_dmc_out0	rw_intr_mask
0xb00205d8	iop_dmc_out	iop_dmc_out0	rw_ack_intr
0xb00205dc	iop_dmc_out	iop_dmc_out0	r_intr
0xb00205e0	iop_dmc_out	iop_dmc_out0	r_masked_intr
0xb0020600	iop_dmc_out	iop_dmc_out1	rw_cfg
0xb0020604	iop_dmc_out	iop_dmc_out1	rw_ctrl
0xb0020608	iop_dmc_out	iop_dmc_out1	r_stat
0xb002060c	iop_dmc_out	iop_dmc_out1	rw_stream_cmd
0xb0020610	iop_dmc_out	iop_dmc_out1	rs_stream_data
0xb0020614	iop_dmc_out	iop_dmc_out1	r_stream_data
0xb0020618	iop_dmc_out	iop_dmc_out1	r_stream_stat
0xb002061c	iop_dmc_out	iop_dmc_out1	r_data_descr
0xb0020620	iop_dmc_out	iop_dmc_out1	r_ctxt_descr
0xb0020624	iop_dmc_out	iop_dmc_out1	r_ctxt_descr_md1
0xb0020628	iop_dmc_out	iop_dmc_out1	r_ctxt_descr_md2
0xb0020634	iop_dmc_out	iop_dmc_out1	r_group_descr
0xb0020638	iop_dmc_out	iop_dmc_out1	rw_data_descr

0xb002063c	iop_dmc_out	iop_dmc_out1	rw_ctxt_descr
0xb0020640	iop_dmc_out	iop_dmc_out1	rw_ctxt_descr_md1
0xb0020644	iop_dmc_out	iop_dmc_out1	rw_ctxt_descr_md2
0xb0020650	iop_dmc_out	iop_dmc_out1	rw_group_descr
0xb0020654	iop_dmc_out	iop_dmc_out1	rw_intr_mask
0xb0020658	iop_dmc_out	iop_dmc_out1	rw_ack_intr
0xb002065c	iop_dmc_out	iop_dmc_out1	r_intr
0xb0020660	iop_dmc_out	iop_dmc_out1	r_masked_intr
0xb0020680	iop_fifo_in	iop_fifo_in0	rw_cfg
0xb0020684	iop_fifo_in	iop_fifo_in0	rw_ctrl
0xb0020688	iop_fifo_in	iop_fifo_in0	r_stat
0xb002068c	iop_fifo_in	iop_fifo_in0	rs_rd1byte
0xb0020690	iop_fifo_in	iop_fifo_in0	r_rd1byte
0xb0020694	iop_fifo_in	iop_fifo_in0	rs_rd2byte
0xb0020698	iop_fifo_in	iop_fifo_in0	r_rd2byte
0xb002069c	iop_fifo_in	iop_fifo_in0	rs_rd3byte
0xb00206a0	iop_fifo_in	iop_fifo_in0	r_rd3byte
0xb00206a4	iop_fifo_in	iop_fifo_in0	rs_rd4byte
0xb00206a8	iop_fifo_in	iop_fifo_in0	r_rd4byte
0xb00206ac	iop_fifo_in	iop_fifo_in0	rw_set_last
0xb00206b0	iop_fifo_in	iop_fifo_in0	rw_strb_dif_in
0xb00206b4	iop_fifo_in	iop_fifo_in0	rw_intr_mask
0xb00206b8	iop_fifo_in	iop_fifo_in0	rw_ack_intr
0xb00206bc	iop_fifo_in	iop_fifo_in0	r_intr
0xb00206c0	iop_fifo_in	iop_fifo_in0	r_masked_intr
0xb0020700	iop_fifo_in	iop_fifo_in1	rw_cfg
0xb0020704	iop_fifo_in	iop_fifo_in1	rw_ctrl
0xb0020708	iop_fifo_in	iop_fifo_in1	r_stat
0xb002070c	iop_fifo_in	iop_fifo_in1	rs_rd1byte
0xb0020710	iop_fifo_in	iop_fifo_in1	r_rd1byte
0xb0020714	iop_fifo_in	iop_fifo_in1	rs_rd2byte
0xb0020718	iop_fifo_in	iop_fifo_in1	r_rd2byte
0xb002071c	iop_fifo_in	iop_fifo_in1	rs_rd3byte
0xb0020720	iop_fifo_in	iop_fifo_in1	r_rd3byte
0xb0020724	iop_fifo_in	iop_fifo_in1	rs_rd4byte
0xb0020728	iop_fifo_in	iop_fifo_in1	r_rd4byte
0xb002072c	iop_fifo_in	iop_fifo_in1	rw_set_last
0xb0020730	iop_fifo_in	iop_fifo_in1	rw_strb_dif_in
0xb0020734	iop_fifo_in	iop_fifo_in1	rw_intr_mask
0xb0020738	iop_fifo_in	iop_fifo_in1	rw_ack_intr
0xb002073c	iop_fifo_in	iop_fifo_in1	r_intr

0xb0020740	iop_fifo_in	iop_fifo_in1	r_masked_intr
0xb0020780	iop_fifo_out	iop_fifo_out0	rw_cfg
0xb0020784	iop_fifo_out	iop_fifo_out0	rw_ctrl
0xb0020788	iop_fifo_out	iop_fifo_out0	r_stat
0xb002078c	iop_fifo_out	iop_fifo_out0	rw_wr1byte
0xb0020790	iop_fifo_out	iop_fifo_out0	rw_wr2byte
0xb0020794	iop_fifo_out	iop_fifo_out0	rw_wr3byte
0xb0020798	iop_fifo_out	iop_fifo_out0	rw_wr4byte
0xb002079c	iop_fifo_out	iop_fifo_out0	rw_wr1byte_last
0xb00207a0	iop_fifo_out	iop_fifo_out0	rw_wr2byte_last
0xb00207a4	iop_fifo_out	iop_fifo_out0	rw_wr3byte_last
0xb00207a8	iop_fifo_out	iop_fifo_out0	rw_wr4byte_last
0xb00207ac	iop_fifo_out	iop_fifo_out0	rw_set_last
0xb00207b0	iop_fifo_out	iop_fifo_out0	rs_rd_data
0xb00207b4	iop_fifo_out	iop_fifo_out0	r_rd_data
0xb00207b8	iop_fifo_out	iop_fifo_out0	rw_strb_dif_out
0xb00207bc	iop_fifo_out	iop_fifo_out0	rw_intr_mask
0xb00207c0	iop_fifo_out	iop_fifo_out0	rw_ack_intr
0xb00207c4	iop_fifo_out	iop_fifo_out0	r_intr
0xb00207c8	iop_fifo_out	iop_fifo_out0	r_masked_intr
0xb0020800	iop_fifo_out	iop_fifo_out1	rw_cfg
0xb0020804	iop_fifo_out	iop_fifo_out1	rw_ctrl
0xb0020808	iop_fifo_out	iop_fifo_out1	r_stat
0xb002080c	iop_fifo_out	iop_fifo_out1	rw_wr1byte
0xb0020810	iop_fifo_out	iop_fifo_out1	rw_wr2byte
0xb0020814	iop_fifo_out	iop_fifo_out1	rw_wr3byte
0xb0020818	iop_fifo_out	iop_fifo_out1	rw_wr4byte
0xb002081c	iop_fifo_out	iop_fifo_out1	rw_wr1byte_last
0xb0020820	iop_fifo_out	iop_fifo_out1	rw_wr2byte_last
0xb0020824	iop_fifo_out	iop_fifo_out1	rw_wr3byte_last
0xb0020828	iop_fifo_out	iop_fifo_out1	rw_wr4byte_last
0xb002082c	iop_fifo_out	iop_fifo_out1	rw_set_last
0xb0020830	iop_fifo_out	iop_fifo_out1	rs_rd_data
0xb0020834	iop_fifo_out	iop_fifo_out1	r_rd_data
0xb0020838	iop_fifo_out	iop_fifo_out1	rw_strb_dif_out
0xb002083c	iop_fifo_out	iop_fifo_out1	rw_intr_mask
0xb0020840	iop_fifo_out	iop_fifo_out1	rw_ack_intr
0xb0020844	iop_fifo_out	iop_fifo_out1	r_intr
0xb0020848	iop_fifo_out	iop_fifo_out1	r_masked_intr
0xb0020880	iop_scrc_in	iop_scrc_in0	rw_cfg
0xb0020884	iop_scrc_in	iop_scrc_in0	rw_ctrl

0xb0020888	iop_scrc_in	iop_scrc_in0	r_stat
0xb002088c	iop_scrc_in	iop_scrc_in0	rw_init_crc
0xb0020890	iop_scrc_in	iop_scrc_in0	rs_computed_crc
0xb0020894	iop_scrc_in	iop_scrc_in0	r_computed_crc
0xb0020898	iop_scrc_in	iop_scrc_in0	rw_crc
0xb002089c	iop_scrc_in	iop_scrc_in0	rw_correct_crc
0xb00208a0	iop_scrc_in	iop_scrc_in0	rw_wr1bit
0xb0020900	iop_scrc_in	iop_scrc_in1	rw_cfg
0xb0020904	iop_scrc_in	iop_scrc_in1	rw_ctrl
0xb0020908	iop_scrc_in	iop_scrc_in1	r_stat
0xb002090c	iop_scrc_in	iop_scrc_in1	rw_init_crc
0xb0020910	iop_scrc_in	iop_scrc_in1	rs_computed_crc
0xb0020914	iop_scrc_in	iop_scrc_in1	r_computed_crc
0xb0020918	iop_scrc_in	iop_scrc_in1	rw_crc
0xb002091c	iop_scrc_in	iop_scrc_in1	rw_correct_crc
0xb0020920	iop_scrc_in	iop_scrc_in1	rw_wr1bit
0xb0020980	iop_scrc_out	iop_scrc_out0	rw_cfg
0xb0020984	iop_scrc_out	iop_scrc_out0	rw_ctrl
0xb0020988	iop_scrc_out	iop_scrc_out0	rw_init_crc
0xb002098c	iop_scrc_out	iop_scrc_out0	rw_crc
0xb0020990	iop_scrc_out	iop_scrc_out0	rw_data
0xb0020994	iop_scrc_out	iop_scrc_out0	r_computed_crc
0xb0020a00	iop_scrc_out	iop_scrc_out1	rw_cfg
0xb0020a04	iop_scrc_out	iop_scrc_out1	rw_ctrl
0xb0020a08	iop_scrc_out	iop_scrc_out1	rw_init_crc
0xb0020a0c	iop_scrc_out	iop_scrc_out1	rw_crc
0xb0020a10	iop_scrc_out	iop_scrc_out1	rw_data
0xb0020a14	iop_scrc_out	iop_scrc_out1	r_computed_crc
0xb0020a80	iop_timer_grp	iop_timer_grp0	rw_cfg
0xb0020a84	iop_timer_grp	iop_timer_grp0	rw_half_period
0xb0020a88	iop_timer_grp	iop_timer_grp0	rw_half_period_len
0xb0020a8c	iop_timer_grp	iop_timer_grp0	rw_tmr_cfg
0xb0020a90	iop_timer_grp	iop_timer_grp0	rw_tmr_cfg
0xb0020a94	iop_timer_grp	iop_timer_grp0	rw_tmr_cfg
0xb0020a98	iop_timer_grp	iop_timer_grp0	rw_tmr_cfg
0xb0020aac	iop_timer_grp	iop_timer_grp0	rw_tmr_len
0xb0020ab0	iop_timer_grp	iop_timer_grp0	rw_tmr_len
0xb0020ab4	iop_timer_grp	iop_timer_grp0	rw_tmr_len
0xb0020ab8	iop_timer_grp	iop_timer_grp0	rw_tmr_len
0xb0020abc	iop_timer_grp	iop_timer_grp0	rw_cmd
0xb0020ac0	iop_timer_grp	iop_timer_grp0	r_clk_gen_cnt

0xb0020ac4	iop_timer_grp	iop_timer_grp0	rs_tmr_cnt
0xb0020acc	iop_timer_grp	iop_timer_grp0	rs_tmr_cnt
0xb0020ad4	iop_timer_grp	iop_timer_grp0	rs_tmr_cnt
0xb0020adc	iop_timer_grp	iop_timer_grp0	rs_tmr_cnt
0xb0020ae0	iop_timer_grp	iop_timer_grp0	r_tmr_cnt
0xb0020ae4	iop_timer_grp	iop_timer_grp0	rw_intr_mask
0xb0020ae8	iop_timer_grp	iop_timer_grp0	rw_ack_intr
0xb0020aec	iop_timer_grp	iop_timer_grp0	r_intr
0xb0020af0	iop_timer_grp	iop_timer_grp0	r_masked_intr
0xb0020b00	iop_timer_grp	iop_timer_grp1	rw_cfg
0xb0020b04	iop_timer_grp	iop_timer_grp1	rw_half_period
0xb0020b08	iop_timer_grp	iop_timer_grp1	rw_half_period_len
0xb0020b0c	iop_timer_grp	iop_timer_grp1	rw_tmr_cfg
0xb0020b10	iop_timer_grp	iop_timer_grp1	rw_tmr_cfg
0xb0020b14	iop_timer_grp	iop_timer_grp1	rw_tmr_cfg
0xb0020b18	iop_timer_grp	iop_timer_grp1	rw_tmr_cfg
0xb0020b2c	iop_timer_grp	iop_timer_grp1	rw_tmr_len
0xb0020b30	iop_timer_grp	iop_timer_grp1	rw_tmr_len
0xb0020b34	iop_timer_grp	iop_timer_grp1	rw_tmr_len
0xb0020b38	iop_timer_grp	iop_timer_grp1	rw_tmr_len
0xb0020b3c	iop_timer_grp	iop_timer_grp1	rw_cmd
0xb0020b40	iop_timer_grp	iop_timer_grp1	r_clk_gen_cnt
0xb0020b44	iop_timer_grp	iop_timer_grp1	rs_tmr_cnt
0xb0020b4c	iop_timer_grp	iop_timer_grp1	rs_tmr_cnt
0xb0020b54	iop_timer_grp	iop_timer_grp1	rs_tmr_cnt
0xb0020b5c	iop_timer_grp	iop_timer_grp1	rs_tmr_cnt
0xb0020b60	iop_timer_grp	iop_timer_grp1	r_tmr_cnt
0xb0020b64	iop_timer_grp	iop_timer_grp1	rw_intr_mask
0xb0020b68	iop_timer_grp	iop_timer_grp1	rw_ack_intr
0xb0020b6c	iop_timer_grp	iop_timer_grp1	r_intr
0xb0020b70	iop_timer_grp	iop_timer_grp1	r_masked_intr
0xb0020b80	iop_timer_grp	iop_timer_grp2	rw_cfg
0xb0020b84	iop_timer_grp	iop_timer_grp2	rw_half_period
0xb0020b88	iop_timer_grp	iop_timer_grp2	rw_half_period_len
0xb0020b8c	iop_timer_grp	iop_timer_grp2	rw_tmr_cfg
0xb0020b90	iop_timer_grp	iop_timer_grp2	rw_tmr_cfg
0xb0020b94	iop_timer_grp	iop_timer_grp2	rw_tmr_cfg
0xb0020b98	iop_timer_grp	iop_timer_grp2	rw_tmr_cfg
0xb0020bac	iop_timer_grp	iop_timer_grp2	rw_tmr_len
0xb0020bb0	iop_timer_grp	iop_timer_grp2	rw_tmr_len
0xb0020bb4	iop_timer_grp	iop_timer_grp2	rw_tmr_len

0xb0020bb8	iop_timer_grp	iop_timer_grp2	rw_tmr_len
0xb0020bbc	iop_timer_grp	iop_timer_grp2	rw_cmd
0xb0020bc0	iop_timer_grp	iop_timer_grp2	r_clk_gen_cnt
0xb0020bc4	iop_timer_grp	iop_timer_grp2	rs_tmr_cnt
0xb0020bcc	iop_timer_grp	iop_timer_grp2	rs_tmr_cnt
0xb0020bd4	iop_timer_grp	iop_timer_grp2	rs_tmr_cnt
0xb0020bdc	iop_timer_grp	iop_timer_grp2	rs_tmr_cnt
0xb0020be0	iop_timer_grp	iop_timer_grp2	r_tmr_cnt
0xb0020be4	iop_timer_grp	iop_timer_grp2	rw_intr_mask
0xb0020be8	iop_timer_grp	iop_timer_grp2	rw_ack_intr
0xb0020bec	iop_timer_grp	iop_timer_grp2	r_intr
0xb0020bf0	iop_timer_grp	iop_timer_grp2	r_masked_intr
0xb0020c00	iop_timer_grp	iop_timer_grp3	rw_cfg
0xb0020c04	iop_timer_grp	iop_timer_grp3	rw_half_period
0xb0020c08	iop_timer_grp	iop_timer_grp3	rw_half_period_len
0xb0020c0c	iop_timer_grp	iop_timer_grp3	rw_tmr_cfg
0xb0020c10	iop_timer_grp	iop_timer_grp3	rw_tmr_cfg
0xb0020c14	iop_timer_grp	iop_timer_grp3	rw_tmr_cfg
0xb0020c18	iop_timer_grp	iop_timer_grp3	rw_tmr_cfg
0xb0020c2c	iop_timer_grp	iop_timer_grp3	rw_tmr_len
0xb0020c30	iop_timer_grp	iop_timer_grp3	rw_tmr_len
0xb0020c34	iop_timer_grp	iop_timer_grp3	rw_tmr_len
0xb0020c38	iop_timer_grp	iop_timer_grp3	rw_tmr_len
0xb0020c3c	iop_timer_grp	iop_timer_grp3	rw_cmd
0xb0020c40	iop_timer_grp	iop_timer_grp3	r_clk_gen_cnt
0xb0020c44	iop_timer_grp	iop_timer_grp3	rs_tmr_cnt
0xb0020c4c	iop_timer_grp	iop_timer_grp3	rs_tmr_cnt
0xb0020c54	iop_timer_grp	iop_timer_grp3	rs_tmr_cnt
0xb0020c5c	iop_timer_grp	iop_timer_grp3	rs_tmr_cnt
0xb0020c60	iop_timer_grp	iop_timer_grp3	r_tmr_cnt
0xb0020c64	iop_timer_grp	iop_timer_grp3	rw_intr_mask
0xb0020c68	iop_timer_grp	iop_timer_grp3	rw_ack_intr
0xb0020c6c	iop_timer_grp	iop_timer_grp3	r_intr
0xb0020c70	iop_timer_grp	iop_timer_grp3	r_masked_intr
0xb0020d00	iop_sap_in	iop_sap_in	rw_bus0_sync
0xb0020d04	iop_sap_in	iop_sap_in	rw_bus1_sync
0xb0020d08	iop_sap_in	iop_sap_in	rw_gio
0xb0020d0c	iop_sap_in	iop_sap_in	rw_gio
0xb0020d10	iop_sap_in	iop_sap_in	rw_gio
0xb0020d14	iop_sap_in	iop_sap_in	rw_gio
0xb0020d18	iop_sap_in	iop_sap_in	rw_gio

0xb0020d1c	iop_sap_in	iop_sap_in	rw_gio
0xb0020d20	iop_sap_in	iop_sap_in	rw_gio
0xb0020d24	iop_sap_in	iop_sap_in	rw_gio
0xb0020d28	iop_sap_in	iop_sap_in	rw_gio
0xb0020d2c	iop_sap_in	iop_sap_in	rw_gio
0xb0020d30	iop_sap_in	iop_sap_in	rw_gio
0xb0020d34	iop_sap_in	iop_sap_in	rw_gio
0xb0020d38	iop_sap_in	iop_sap_in	rw_gio
0xb0020d3c	iop_sap_in	iop_sap_in	rw_gio
0xb0020d40	iop_sap_in	iop_sap_in	rw_gio
0xb0020d44	iop_sap_in	iop_sap_in	rw_gio
0xb0020d48	iop_sap_in	iop_sap_in	rw_gio
0xb0020d4c	iop_sap_in	iop_sap_in	rw_gio
0xb0020d50	iop_sap_in	iop_sap_in	rw_gio
0xb0020d54	iop_sap_in	iop_sap_in	rw_gio
0xb0020d58	iop_sap_in	iop_sap_in	rw_gio
0xb0020d5c	iop_sap_in	iop_sap_in	rw_gio
0xb0020d60	iop_sap_in	iop_sap_in	rw_gio
0xb0020d64	iop_sap_in	iop_sap_in	rw_gio
0xb0020d68	iop_sap_in	iop_sap_in	rw_gio
0xb0020d6c	iop_sap_in	iop_sap_in	rw_gio
0xb0020d70	iop_sap_in	iop_sap_in	rw_gio
0xb0020d74	iop_sap_in	iop_sap_in	rw_gio
0xb0020d78	iop_sap_in	iop_sap_in	rw_gio
0xb0020d7c	iop_sap_in	iop_sap_in	rw_gio
0xb0020d80	iop_sap_in	iop_sap_in	rw_gio
0xb0020d84	iop_sap_in	iop_sap_in	rw_gio
0xb0020e00	iop_sap_out	iop_sap_out	rw_gen_gated
0xb0020e04	iop_sap_out	iop_sap_out	rw_bus0
0xb0020e08	iop_sap_out	iop_sap_out	rw_bus1
0xb0020e0c	iop_sap_out	iop_sap_out	rw_bus0_lo_oe
0xb0020e10	iop_sap_out	iop_sap_out	rw_bus0_hi_oe
0xb0020e14	iop_sap_out	iop_sap_out	rw_bus1_lo_oe
0xb0020e18	iop_sap_out	iop_sap_out	rw_bus1_hi_oe
0xb0020e1c	iop_sap_out	iop_sap_out	rw_gio
0xb0020e20	iop_sap_out	iop_sap_out	rw_gio
0xb0020e24	iop_sap_out	iop_sap_out	rw_gio
0xb0020e28	iop_sap_out	iop_sap_out	rw_gio
0xb0020e2c	iop_sap_out	iop_sap_out	rw_gio
0xb0020e30	iop_sap_out	iop_sap_out	rw_gio
0xb0020e34	iop_sap_out	iop_sap_out	rw_gio

0xb0020e38	iop_sap_out	iop_sap_out	rw_gio
0xb0020e3c	iop_sap_out	iop_sap_out	rw_gio
0xb0020e40	iop_sap_out	iop_sap_out	rw_gio
0xb0020e44	iop_sap_out	iop_sap_out	rw_gio
0xb0020e48	iop_sap_out	iop_sap_out	rw_gio
0xb0020e4c	iop_sap_out	iop_sap_out	rw_gio
0xb0020e50	iop_sap_out	iop_sap_out	rw_gio
0xb0020e54	iop_sap_out	iop_sap_out	rw_gio
0xb0020e58	iop_sap_out	iop_sap_out	rw_gio
0xb0020e5c	iop_sap_out	iop_sap_out	rw_gio
0xb0020e60	iop_sap_out	iop_sap_out	rw_gio
0xb0020e64	iop_sap_out	iop_sap_out	rw_gio
0xb0020e68	iop_sap_out	iop_sap_out	rw_gio
0xb0020e6c	iop_sap_out	iop_sap_out	rw_gio
0xb0020e70	iop_sap_out	iop_sap_out	rw_gio
0xb0020e74	iop_sap_out	iop_sap_out	rw_gio
0xb0020e78	iop_sap_out	iop_sap_out	rw_gio
0xb0020e7c	iop_sap_out	iop_sap_out	rw_gio
0xb0020e80	iop_sap_out	iop_sap_out	rw_gio
0xb0020e84	iop_sap_out	iop_sap_out	rw_gio
0xb0020e88	iop_sap_out	iop_sap_out	rw_gio
0xb0020e8c	iop_sap_out	iop_sap_out	rw_gio
0xb0020e90	iop_sap_out	iop_sap_out	rw_gio
0xb0020e94	iop_sap_out	iop_sap_out	rw_gio
0xb0020e98	iop_sap_out	iop_sap_out	rw_gio
0xb0020f00	iop_spu	iop_spu0	rw_r
0xb0020f04	iop_spu	iop_spu0	rw_r
0xb0020f08	iop_spu	iop_spu0	rw_r
0xb0020f0c	iop_spu	iop_spu0	rw_r
0xb0020f10	iop_spu	iop_spu0	rw_r
0xb0020f14	iop_spu	iop_spu0	rw_r
0xb0020f18	iop_spu	iop_spu0	rw_r
0xb0020f1c	iop_spu	iop_spu0	rw_r
0xb0020f20	iop_spu	iop_spu0	rw_r
0xb0020f24	iop_spu	iop_spu0	rw_r
0xb0020f28	iop_spu	iop_spu0	rw_r
0xb0020f2c	iop_spu	iop_spu0	rw_r
0xb0020f30	iop_spu	iop_spu0	rw_r
0xb0020f34	iop_spu	iop_spu0	rw_r
0xb0020f38	iop_spu	iop_spu0	rw_r
0xb0020f3c	iop_spu	iop_spu0	rw_r

0xb0020f40	iop_spu	iop_spu0	rw_seq_pc
0xb0020f44	iop_spu	iop_spu0	rw_fsm_pc
0xb0020f48	iop_spu	iop_spu0	rw_ctrl
0xb0020f4c	iop_spu	iop_spu0	rw_fsm_inputs3_0
0xb0020f50	iop_spu	iop_spu0	rw_fsm_inputs7_4
0xb0020f54	iop_spu	iop_spu0	rw_gio_out
0xb0020f58	iop_spu	iop_spu0	rw_bus0_out
0xb0020f5c	iop_spu	iop_spu0	rw_bus1_out
0xb0020f60	iop_spu	iop_spu0	r_gio_in
0xb0020f64	iop_spu	iop_spu0	r_bus0_in
0xb0020f68	iop_spu	iop_spu0	r_bus1_in
0xb0020f6c	iop_spu	iop_spu0	rw_gio_out_set
0xb0020f70	iop_spu	iop_spu0	rw_gio_out_clr
0xb0020f74	iop_spu	iop_spu0	rs_wr_stat
0xb0020f78	iop_spu	iop_spu0	r_wr_stat
0xb0020f7c	iop_spu	iop_spu0	r_reg_indexed_by_bus0_in
0xb0020f80	iop_spu	iop_spu0	r_stat_in
0xb0020f84	iop_spu	iop_spu0	r_trigger_in
0xb0020f88	iop_spu	iop_spu0	r_special_stat
0xb0020f8c	iop_spu	iop_spu0	rw_reg_access
0xb0020f90	iop_spu	iop_spu0	rw_event_cfg
0xb0020f94	iop_spu	iop_spu0	rw_event_cfg
0xb0020f98	iop_spu	iop_spu0	rw_event_cfg
0xb0020f9c	iop_spu	iop_spu0	rw_event_cfg
0xb0020fa0	iop_spu	iop_spu0	rw_event_mask
0xb0020fa4	iop_spu	iop_spu0	rw_event_mask
0xb0020fa8	iop_spu	iop_spu0	rw_event_mask
0xb0020fac	iop_spu	iop_spu0	rw_event_mask
0xb0020fb0	iop_spu	iop_spu0	rw_event_val
0xb0020fb4	iop_spu	iop_spu0	rw_event_val
0xb0020fb8	iop_spu	iop_spu0	rw_event_val
0xb0020fbc	iop_spu	iop_spu0	rw_event_val
0xb0020fc0	iop_spu	iop_spu0	rw_event_ret
0xb0020fc4	iop_spu	iop_spu0	r_trace
0xb0020fc8	iop_spu	iop_spu0	r_fsm_trace
0xb0020fcc	iop_spu	iop_spu0	rw_brp
0xb0020fd0	iop_spu	iop_spu0	rw_brp
0xb0020fd4	iop_spu	iop_spu0	rw_brp
0xb0020fd8	iop_spu	iop_spu0	rw_brp
0xb0021000	iop_spu	iop_spu1	rw_r
0xb0021004	iop_spu	iop_spu1	rw_r

0xb0021008	iop_spu	iop_spu1	rw_r
0xb002100c	iop_spu	iop_spu1	rw_r
0xb0021010	iop_spu	iop_spu1	rw_r
0xb0021014	iop_spu	iop_spu1	rw_r
0xb0021018	iop_spu	iop_spu1	rw_r
0xb002101c	iop_spu	iop_spu1	rw_r
0xb0021020	iop_spu	iop_spu1	rw_r
0xb0021024	iop_spu	iop_spu1	rw_r
0xb0021028	iop_spu	iop_spu1	rw_r
0xb002102c	iop_spu	iop_spu1	rw_r
0xb0021030	iop_spu	iop_spu1	rw_r
0xb0021034	iop_spu	iop_spu1	rw_r
0xb0021038	iop_spu	iop_spu1	rw_r
0xb002103c	iop_spu	iop_spu1	rw_r
0xb0021040	iop_spu	iop_spu1	rw_seq_pc
0xb0021044	iop_spu	iop_spu1	rw_fsm_pc
0xb0021048	iop_spu	iop_spu1	rw_ctrl
0xb002104c	iop_spu	iop_spu1	rw_fsm_inputs3_0
0xb0021050	iop_spu	iop_spu1	rw_fsm_inputs7_4
0xb0021054	iop_spu	iop_spu1	rw_gio_out
0xb0021058	iop_spu	iop_spu1	rw_bus0_out
0xb002105c	iop_spu	iop_spu1	rw_bus1_out
0xb0021060	iop_spu	iop_spu1	r_gio_in
0xb0021064	iop_spu	iop_spu1	r_bus0_in
0xb0021068	iop_spu	iop_spu1	r_bus1_in
0xb002106c	iop_spu	iop_spu1	rw_gio_out_set
0xb0021070	iop_spu	iop_spu1	rw_gio_out_clr
0xb0021074	iop_spu	iop_spu1	rs_wr_stat
0xb0021078	iop_spu	iop_spu1	r_wr_stat
0xb002107c	iop_spu	iop_spu1	r_reg_indexed_by_bus0_in
0xb0021080	iop_spu	iop_spu1	r_stat_in
0xb0021084	iop_spu	iop_spu1	r_trigger_in
0xb0021088	iop_spu	iop_spu1	r_special_stat
0xb002108c	iop_spu	iop_spu1	rw_reg_access
0xb0021090	iop_spu	iop_spu1	rw_event_cfg
0xb0021094	iop_spu	iop_spu1	rw_event_cfg
0xb0021098	iop_spu	iop_spu1	rw_event_cfg
0xb002109c	iop_spu	iop_spu1	rw_event_cfg
0xb00210a0	iop_spu	iop_spu1	rw_event_mask
0xb00210a4	iop_spu	iop_spu1	rw_event_mask
0xb00210a8	iop_spu	iop_spu1	rw_event_mask

0xb00210ac	iop_spu	iop_spu1	rw_event_mask
0xb00210b0	iop_spu	iop_spu1	rw_event_val
0xb00210b4	iop_spu	iop_spu1	rw_event_val
0xb00210b8	iop_spu	iop_spu1	rw_event_val
0xb00210bc	iop_spu	iop_spu1	rw_event_val
0xb00210c0	iop_spu	iop_spu1	rw_event_ret
0xb00210c4	iop_spu	iop_spu1	r_trace
0xb00210c8	iop_spu	iop_spu1	r_fsm_trace
0xb00210cc	iop_spu	iop_spu1	rw_brp
0xb00210d0	iop_spu	iop_spu1	rw_brp
0xb00210d4	iop_spu	iop_spu1	rw_brp
0xb00210d8	iop_spu	iop_spu1	rw_brp
0xb0021100	iop_sw_cfg	iop_sw_cfg	rw_crc_par0_owner
0xb0021104	iop_sw_cfg	iop_sw_cfg	rw_crc_par1_owner
0xb0021108	iop_sw_cfg	iop_sw_cfg	rw_dmc_in0_owner
0xb002110c	iop_sw_cfg	iop_sw_cfg	rw_dmc_in1_owner
0xb0021110	iop_sw_cfg	iop_sw_cfg	rw_dmc_out0_owner
0xb0021114	iop_sw_cfg	iop_sw_cfg	rw_dmc_out1_owner
0xb0021118	iop_sw_cfg	iop_sw_cfg	rw_fifo_in0_owner
0xb002111c	iop_sw_cfg	iop_sw_cfg	rw_fifo_in0_extra_owner
0xb0021120	iop_sw_cfg	iop_sw_cfg	rw_fifo_in1_owner
0xb0021124	iop_sw_cfg	iop_sw_cfg	rw_fifo_in1_extra_owner
0xb0021128	iop_sw_cfg	iop_sw_cfg	rw_fifo_out0_owner
0xb002112c	iop_sw_cfg	iop_sw_cfg	rw_fifo_out0_extra_owner
0xb0021130	iop_sw_cfg	iop_sw_cfg	rw_fifo_out1_owner
0xb0021134	iop_sw_cfg	iop_sw_cfg	rw_fifo_out1_extra_owner
0xb0021138	iop_sw_cfg	iop_sw_cfg	rw_sap_in_owner
0xb002113c	iop_sw_cfg	iop_sw_cfg	rw_sap_out_owner
0xb0021140	iop_sw_cfg	iop_sw_cfg	rw_scrc_in0_owner
0xb0021144	iop_sw_cfg	iop_sw_cfg	rw_scrc_in1_owner
0xb0021148	iop_sw_cfg	iop_sw_cfg	rw_scrc_out0_owner
0xb002114c	iop_sw_cfg	iop_sw_cfg	rw_scrc_out1_owner
0xb0021150	iop_sw_cfg	iop_sw_cfg	rw_spu0_owner
0xb0021154	iop_sw_cfg	iop_sw_cfg	rw_spu1_owner
0xb0021158	iop_sw_cfg	iop_sw_cfg	rw_timer_grp0_owner
0xb002115c	iop_sw_cfg	iop_sw_cfg	rw_timer_grp1_owner
0xb0021160	iop_sw_cfg	iop_sw_cfg	rw_timer_grp2_owner
0xb0021164	iop_sw_cfg	iop_sw_cfg	rw_timer_grp3_owner
0xb0021168	iop_sw_cfg	iop_sw_cfg	rw_trigger_grp0_owner
0xb002116c	iop_sw_cfg	iop_sw_cfg	rw_trigger_grp1_owner
0xb0021170	iop_sw_cfg	iop_sw_cfg	rw_trigger_grp2_owner

0xb0021174	iop_sw_cfg	iop_sw_cfg	rw_trigger_grp3_owner
0xb0021178	iop_sw_cfg	iop_sw_cfg	rw_trigger_grp4_owner
0xb002117c	iop_sw_cfg	iop_sw_cfg	rw_trigger_grp5_owner
0xb0021180	iop_sw_cfg	iop_sw_cfg	rw_trigger_grp6_owner
0xb0021184	iop_sw_cfg	iop_sw_cfg	rw_trigger_grp7_owner
0xb0021188	iop_sw_cfg	iop_sw_cfg	rw_bus0_mask
0xb002118c	iop_sw_cfg	iop_sw_cfg	rw_bus0_oe_mask
0xb0021190	iop_sw_cfg	iop_sw_cfg	rw_bus1_mask
0xb0021194	iop_sw_cfg	iop_sw_cfg	rw_bus1_oe_mask
0xb0021198	iop_sw_cfg	iop_sw_cfg	rw_gio_mask
0xb002119c	iop_sw_cfg	iop_sw_cfg	rw_gio_oe_mask
0xb00211a0	iop_sw_cfg	iop_sw_cfg	rw_pinmapping
0xb00211a4	iop_sw_cfg	iop_sw_cfg	rw_bus_out_cfg
0xb00211a8	iop_sw_cfg	iop_sw_cfg	rw_gio_out_grp0_cfg
0xb00211ac	iop_sw_cfg	iop_sw_cfg	rw_gio_out_grp1_cfg
0xb00211b0	iop_sw_cfg	iop_sw_cfg	rw_gio_out_grp2_cfg
0xb00211b4	iop_sw_cfg	iop_sw_cfg	rw_gio_out_grp3_cfg
0xb00211b8	iop_sw_cfg	iop_sw_cfg	rw_gio_out_grp4_cfg
0xb00211bc	iop_sw_cfg	iop_sw_cfg	rw_gio_out_grp5_cfg
0xb00211c0	iop_sw_cfg	iop_sw_cfg	rw_gio_out_grp6_cfg
0xb00211c4	iop_sw_cfg	iop_sw_cfg	rw_gio_out_grp7_cfg
0xb00211c8	iop_sw_cfg	iop_sw_cfg	rw_spu0_cfg
0xb00211cc	iop_sw_cfg	iop_sw_cfg	rw_spu1_cfg
0xb00211d0	iop_sw_cfg	iop_sw_cfg	rw_timer_grp0_cfg
0xb00211d4	iop_sw_cfg	iop_sw_cfg	rw_timer_grp1_cfg
0xb00211d8	iop_sw_cfg	iop_sw_cfg	rw_timer_grp2_cfg
0xb00211dc	iop_sw_cfg	iop_sw_cfg	rw_timer_grp3_cfg
0xb00211e0	iop_sw_cfg	iop_sw_cfg	rw_trigger_grps_cfg
0xb00211e4	iop_sw_cfg	iop_sw_cfg	rw_pdp0_cfg
0xb00211e8	iop_sw_cfg	iop_sw_cfg	rw_pdp1_cfg
0xb00211ec	iop_sw_cfg	iop_sw_cfg	rw_sdp_cfg
0xb0021200	iop_sw_cpu	iop_sw_cpu	rw_mc_ctrl
0xb0021204	iop_sw_cpu	iop_sw_cpu	rw_mc_data
0xb0021208	iop_sw_cpu	iop_sw_cpu	rw_mc_addr
0xb002120c	iop_sw_cpu	iop_sw_cpu	rs_mc_data
0xb0021210	iop_sw_cpu	iop_sw_cpu	r_mc_data
0xb0021214	iop_sw_cpu	iop_sw_cpu	r_mc_stat
0xb0021218	iop_sw_cpu	iop_sw_cpu	rw_bus0_clr_mask
0xb002121c	iop_sw_cpu	iop_sw_cpu	rw_bus0_set_mask
0xb0021220	iop_sw_cpu	iop_sw_cpu	rw_bus0_oe_clr_mask
0xb0021224	iop_sw_cpu	iop_sw_cpu	rw_bus0_oe_set_mask

0xb0021228	iop_sw_cpu	iop_sw_cpu	r_bus0_in
0xb002122c	iop_sw_cpu	iop_sw_cpu	rw_bus1_clr_mask
0xb0021230	iop_sw_cpu	iop_sw_cpu	rw_bus1_set_mask
0xb0021234	iop_sw_cpu	iop_sw_cpu	rw_bus1_oe_clr_mask
0xb0021238	iop_sw_cpu	iop_sw_cpu	rw_bus1_oe_set_mask
0xb002123c	iop_sw_cpu	iop_sw_cpu	r_bus1_in
0xb0021240	iop_sw_cpu	iop_sw_cpu	rw_gio_clr_mask
0xb0021244	iop_sw_cpu	iop_sw_cpu	rw_gio_set_mask
0xb0021248	iop_sw_cpu	iop_sw_cpu	rw_gio_oe_clr_mask
0xb002124c	iop_sw_cpu	iop_sw_cpu	rw_gio_oe_set_mask
0xb0021250	iop_sw_cpu	iop_sw_cpu	r_gio_in
0xb0021254	iop_sw_cpu	iop_sw_cpu	rw_intr0_mask
0xb0021258	iop_sw_cpu	iop_sw_cpu	rw_ack_intr0
0xb002125c	iop_sw_cpu	iop_sw_cpu	r_intr0
0xb0021260	iop_sw_cpu	iop_sw_cpu	r_masked_intr0
0xb0021264	iop_sw_cpu	iop_sw_cpu	rw_intr1_mask
0xb0021268	iop_sw_cpu	iop_sw_cpu	rw_ack_intr1
0xb002126c	iop_sw_cpu	iop_sw_cpu	r_intr1
0xb0021270	iop_sw_cpu	iop_sw_cpu	r_masked_intr1
0xb0021274	iop_sw_cpu	iop_sw_cpu	rw_intr2_mask
0xb0021278	iop_sw_cpu	iop_sw_cpu	rw_ack_intr2
0xb002127c	iop_sw_cpu	iop_sw_cpu	r_intr2
0xb0021280	iop_sw_cpu	iop_sw_cpu	r_masked_intr2
0xb0021284	iop_sw_cpu	iop_sw_cpu	rw_intr3_mask
0xb0021288	iop_sw_cpu	iop_sw_cpu	rw_ack_intr3
0xb002128c	iop_sw_cpu	iop_sw_cpu	r_intr3
0xb0021290	iop_sw_cpu	iop_sw_cpu	r_masked_intr3
0xb0021300	iop_sw_mpu	iop_sw_mpu	rw_sw_cfg_owner
0xb0021304	iop_sw_mpu	iop_sw_mpu	rw_mc_ctrl
0xb0021308	iop_sw_mpu	iop_sw_mpu	rw_mc_data
0xb002130c	iop_sw_mpu	iop_sw_mpu	rw_mc_addr
0xb0021310	iop_sw_mpu	iop_sw_mpu	rs_mc_data
0xb0021314	iop_sw_mpu	iop_sw_mpu	r_mc_data
0xb0021318	iop_sw_mpu	iop_sw_mpu	r_mc_stat
0xb002131c	iop_sw_mpu	iop_sw_mpu	rw_bus0_clr_mask
0xb0021320	iop_sw_mpu	iop_sw_mpu	rw_bus0_set_mask
0xb0021324	iop_sw_mpu	iop_sw_mpu	rw_bus0_oe_clr_mask
0xb0021328	iop_sw_mpu	iop_sw_mpu	rw_bus0_oe_set_mask
0xb002132c	iop_sw_mpu	iop_sw_mpu	r_bus0_in
0xb0021330	iop_sw_mpu	iop_sw_mpu	rw_bus1_clr_mask
0xb0021334	iop_sw_mpu	iop_sw_mpu	rw_bus1_set_mask

0xb0021338	iop_sw_mpu	iop_sw_mpu	rw_bus1_oe_clr_mask
0xb002133c	iop_sw_mpu	iop_sw_mpu	rw_bus1_oe_set_mask
0xb0021340	iop_sw_mpu	iop_sw_mpu	r_bus1_in
0xb0021344	iop_sw_mpu	iop_sw_mpu	rw_gio_clr_mask
0xb0021348	iop_sw_mpu	iop_sw_mpu	rw_gio_set_mask
0xb002134c	iop_sw_mpu	iop_sw_mpu	rw_gio_oe_clr_mask
0xb0021350	iop_sw_mpu	iop_sw_mpu	rw_gio_oe_set_mask
0xb0021354	iop_sw_mpu	iop_sw_mpu	r_gio_in
0xb0021358	iop_sw_mpu	iop_sw_mpu	rw_cpu_intr
0xb002135c	iop_sw_mpu	iop_sw_mpu	r_cpu_intr
0xb0021360	iop_sw_mpu	iop_sw_mpu	rw_intr_grp0_mask
0xb0021364	iop_sw_mpu	iop_sw_mpu	rw_ack_intr_grp0
0xb0021368	iop_sw_mpu	iop_sw_mpu	r_intr_grp0
0xb002136c	iop_sw_mpu	iop_sw_mpu	r_masked_intr_grp0
0xb0021370	iop_sw_mpu	iop_sw_mpu	rw_intr_grp1_mask
0xb0021374	iop_sw_mpu	iop_sw_mpu	rw_ack_intr_grp1
0xb0021378	iop_sw_mpu	iop_sw_mpu	r_intr_grp1
0xb002137c	iop_sw_mpu	iop_sw_mpu	r_masked_intr_grp1
0xb0021380	iop_sw_mpu	iop_sw_mpu	rw_intr_grp2_mask
0xb0021384	iop_sw_mpu	iop_sw_mpu	rw_ack_intr_grp2
0xb0021388	iop_sw_mpu	iop_sw_mpu	r_intr_grp2
0xb002138c	iop_sw_mpu	iop_sw_mpu	r_masked_intr_grp2
0xb0021390	iop_sw_mpu	iop_sw_mpu	rw_intr_grp3_mask
0xb0021394	iop_sw_mpu	iop_sw_mpu	rw_ack_intr_grp3
0xb0021398	iop_sw_mpu	iop_sw_mpu	r_intr_grp3
0xb002139c	iop_sw_mpu	iop_sw_mpu	r_masked_intr_grp3
0xb0021400	iop_sw_spu	iop_sw_spu0	rw_mc_ctrl
0xb0021404	iop_sw_spu	iop_sw_spu0	rw_mc_data
0xb0021408	iop_sw_spu	iop_sw_spu0	rw_mc_addr
0xb002140c	iop_sw_spu	iop_sw_spu0	rs_mc_data
0xb0021410	iop_sw_spu	iop_sw_spu0	r_mc_data
0xb0021414	iop_sw_spu	iop_sw_spu0	r_mc_stat
0xb0021418	iop_sw_spu	iop_sw_spu0	rw_bus0_clr_mask
0xb002141c	iop_sw_spu	iop_sw_spu0	rw_bus0_set_mask
0xb0021420	iop_sw_spu	iop_sw_spu0	rw_bus0_oe_clr_mask
0xb0021424	iop_sw_spu	iop_sw_spu0	rw_bus0_oe_set_mask
0xb0021428	iop_sw_spu	iop_sw_spu0	r_bus0_in
0xb002142c	iop_sw_spu	iop_sw_spu0	rw_bus1_clr_mask
0xb0021430	iop_sw_spu	iop_sw_spu0	rw_bus1_set_mask
0xb0021434	iop_sw_spu	iop_sw_spu0	rw_bus1_oe_clr_mask
0xb0021438	iop_sw_spu	iop_sw_spu0	rw_bus1_oe_set_mask

0xb002143c	iop_sw_spu	iop_sw_spu0	r_bus1_in
0xb0021440	iop_sw_spu	iop_sw_spu0	rw_gio_clr_mask
0xb0021444	iop_sw_spu	iop_sw_spu0	rw_gio_set_mask
0xb0021448	iop_sw_spu	iop_sw_spu0	rw_gio_oe_clr_mask
0xb002144c	iop_sw_spu	iop_sw_spu0	rw_gio_oe_set_mask
0xb0021450	iop_sw_spu	iop_sw_spu0	r_gio_in
0xb0021454	iop_sw_spu	iop_sw_spu0	rw_bus0_clr_mask_lo
0xb0021458	iop_sw_spu	iop_sw_spu0	rw_bus0_clr_mask_hi
0xb002145c	iop_sw_spu	iop_sw_spu0	rw_bus0_set_mask_lo
0xb0021460	iop_sw_spu	iop_sw_spu0	rw_bus0_set_mask_hi
0xb0021464	iop_sw_spu	iop_sw_spu0	rw_bus1_clr_mask_lo
0xb0021468	iop_sw_spu	iop_sw_spu0	rw_bus1_clr_mask_hi
0xb002146c	iop_sw_spu	iop_sw_spu0	rw_bus1_set_mask_lo
0xb0021470	iop_sw_spu	iop_sw_spu0	rw_bus1_set_mask_hi
0xb0021474	iop_sw_spu	iop_sw_spu0	rw_gio_clr_mask_lo
0xb0021478	iop_sw_spu	iop_sw_spu0	rw_gio_clr_mask_hi
0xb002147c	iop_sw_spu	iop_sw_spu0	rw_gio_set_mask_lo
0xb0021480	iop_sw_spu	iop_sw_spu0	rw_gio_set_mask_hi
0xb0021484	iop_sw_spu	iop_sw_spu0	rw_gio_oe_clr_mask_lo
0xb0021488	iop_sw_spu	iop_sw_spu0	rw_gio_oe_clr_mask_hi
0xb002148c	iop_sw_spu	iop_sw_spu0	rw_gio_oe_set_mask_lo
0xb0021490	iop_sw_spu	iop_sw_spu0	rw_gio_oe_set_mask_hi
0xb0021494	iop_sw_spu	iop_sw_spu0	rw_cpu_intr
0xb0021498	iop_sw_spu	iop_sw_spu0	r_cpu_intr
0xb002149c	iop_sw_spu	iop_sw_spu0	r_hw_intr
0xb00214a0	iop_sw_spu	iop_sw_spu0	rw_mpu_intr
0xb00214a4	iop_sw_spu	iop_sw_spu0	r_mpu_intr
0xb0021500	iop_sw_spu	iop_sw_spu1	rw_mc_ctrl
0xb0021504	iop_sw_spu	iop_sw_spu1	rw_mc_data
0xb0021508	iop_sw_spu	iop_sw_spu1	rw_mc_addr
0xb002150c	iop_sw_spu	iop_sw_spu1	rs_mc_data
0xb0021510	iop_sw_spu	iop_sw_spu1	r_mc_data
0xb0021514	iop_sw_spu	iop_sw_spu1	r_mc_stat
0xb0021518	iop_sw_spu	iop_sw_spu1	rw_bus0_clr_mask
0xb002151c	iop_sw_spu	iop_sw_spu1	rw_bus0_set_mask
0xb0021520	iop_sw_spu	iop_sw_spu1	rw_bus0_oe_clr_mask
0xb0021524	iop_sw_spu	iop_sw_spu1	rw_bus0_oe_set_mask
0xb0021528	iop_sw_spu	iop_sw_spu1	r_bus0_in
0xb002152c	iop_sw_spu	iop_sw_spu1	rw_bus1_clr_mask
0xb0021530	iop_sw_spu	iop_sw_spu1	rw_bus1_set_mask
0xb0021534	iop_sw_spu	iop_sw_spu1	rw_bus1_oe_clr_mask

0xb0021538	iop_sw_spu	iop_sw_spu1	rw_bus1_oe_set_mask
0xb002153c	iop_sw_spu	iop_sw_spu1	r_bus1_in
0xb0021540	iop_sw_spu	iop_sw_spu1	rw_gio_clr_mask
0xb0021544	iop_sw_spu	iop_sw_spu1	rw_gio_set_mask
0xb0021548	iop_sw_spu	iop_sw_spu1	rw_gio_oe_clr_mask
0xb002154c	iop_sw_spu	iop_sw_spu1	rw_gio_oe_set_mask
0xb0021550	iop_sw_spu	iop_sw_spu1	r_gio_in
0xb0021554	iop_sw_spu	iop_sw_spu1	rw_bus0_clr_mask_lo
0xb0021558	iop_sw_spu	iop_sw_spu1	rw_bus0_clr_mask_hi
0xb002155c	iop_sw_spu	iop_sw_spu1	rw_bus0_set_mask_lo
0xb0021560	iop_sw_spu	iop_sw_spu1	rw_bus0_set_mask_hi
0xb0021564	iop_sw_spu	iop_sw_spu1	rw_bus1_clr_mask_lo
0xb0021568	iop_sw_spu	iop_sw_spu1	rw_bus1_clr_mask_hi
0xb002156c	iop_sw_spu	iop_sw_spu1	rw_bus1_set_mask_lo
0xb0021570	iop_sw_spu	iop_sw_spu1	rw_bus1_set_mask_hi
0xb0021574	iop_sw_spu	iop_sw_spu1	rw_gio_clr_mask_lo
0xb0021578	iop_sw_spu	iop_sw_spu1	rw_gio_clr_mask_hi
0xb002157c	iop_sw_spu	iop_sw_spu1	rw_gio_set_mask_lo
0xb0021580	iop_sw_spu	iop_sw_spu1	rw_gio_set_mask_hi
0xb0021584	iop_sw_spu	iop_sw_spu1	rw_gio_oe_clr_mask_lo
0xb0021588	iop_sw_spu	iop_sw_spu1	rw_gio_oe_clr_mask_hi
0xb002158c	iop_sw_spu	iop_sw_spu1	rw_gio_oe_set_mask_lo
0xb0021590	iop_sw_spu	iop_sw_spu1	rw_gio_oe_set_mask_hi
0xb0021594	iop_sw_spu	iop_sw_spu1	rw_cpu_intr
0xb0021598	iop_sw_spu	iop_sw_spu1	r_cpu_intr
0xb002159c	iop_sw_spu	iop_sw_spu1	r_hw_intr
0xb00215a0	iop_sw_spu	iop_sw_spu1	rw_mpu_intr
0xb00215a4	iop_sw_spu	iop_sw_spu1	r_mpu_intr
0xb0021600	iop_mpu	iop_mpu	rw_r
0xb0021604	iop_mpu	iop_mpu	rw_r
0xb0021608	iop_mpu	iop_mpu	rw_r
0xb002160c	iop_mpu	iop_mpu	rw_r
0xb0021610	iop_mpu	iop_mpu	rw_r
0xb0021614	iop_mpu	iop_mpu	rw_r
0xb0021618	iop_mpu	iop_mpu	rw_r
0xb002161c	iop_mpu	iop_mpu	rw_r
0xb0021620	iop_mpu	iop_mpu	rw_r
0xb0021624	iop_mpu	iop_mpu	rw_r
0xb0021628	iop_mpu	iop_mpu	rw_r
0xb002162c	iop_mpu	iop_mpu	rw_r
0xb0021630	iop_mpu	iop_mpu	rw_r

0xb0021634	iop_mpu	iop_mpu	rw_r
0xb0021638	iop_mpu	iop_mpu	rw_r
0xb002163c	iop_mpu	iop_mpu	rw_r
0xb0021680	iop_mpu	iop_mpu	rw_ctrl
0xb0021684	iop_mpu	iop_mpu	r_pc
0xb0021688	iop_mpu	iop_mpu	r_stat
0xb002168c	iop_mpu	iop_mpu	rw_instr
0xb0021690	iop_mpu	iop_mpu	rw_immediate
0xb0021694	iop_mpu	iop_mpu	r_trace
0xb0021698	iop_mpu	iop_mpu	r_wr_stat
0xb002169c	iop_mpu	iop_mpu	rw_thread
0xb00216a0	iop_mpu	iop_mpu	rw_thread
0xb00216a4	iop_mpu	iop_mpu	rw_thread
0xb00216a8	iop_mpu	iop_mpu	rw_thread
0xb00216c4	iop_mpu	iop_mpu	rw_intr
0xb00216c8	iop_mpu	iop_mpu	rw_intr
0xb00216cc	iop_mpu	iop_mpu	rw_intr
0xb00216d0	iop_mpu	iop_mpu	rw_intr
0xb00216d4	iop_mpu	iop_mpu	rw_intr
0xb00216d8	iop_mpu	iop_mpu	rw_intr
0xb00216dc	iop_mpu	iop_mpu	rw_intr
0xb00216e0	iop_mpu	iop_mpu	rw_intr
0xb00216e4	iop_mpu	iop_mpu	rw_intr
0xb00216e8	iop_mpu	iop_mpu	rw_intr
0xb00216ec	iop_mpu	iop_mpu	rw_intr
0xb00216f0	iop_mpu	iop_mpu	rw_intr
0xb00216f4	iop_mpu	iop_mpu	rw_intr
0xb00216f8	iop_mpu	iop_mpu	rw_intr
0xb00216fc	iop_mpu	iop_mpu	rw_intr
0xb0021700	iop_mpu	iop_mpu	rw_intr
0xb0022000	sser	sser0	rw_cfg
0xb0022004	sser	sser0	rw_frm_cfg
0xb0022008	sser	sser0	rw_tr_cfg
0xb002200c	sser	sser0	rw_rec_cfg
0xb0022010	sser	sser0	rw_tr_data
0xb0022014	sser	sser0	r_rec_data
0xb0022018	sser	sser0	rw_extra
0xb002201c	sser	sser0	rw_intr_mask
0xb0022020	sser	sser0	rw_ack_intr
0xb0022024	sser	sser0	r_intr
0xb0022028	sser	sser0	r_masked_intr

0xb0024000	sser	sser1	rw_cfg
0xb0024004	sser	sser1	rw_frm_cfg
0xb0024008	sser	sser1	rw_tr_cfg
0xb002400c	sser	sser1	rw_rec_cfg
0xb0024010	sser	sser1	rw_tr_data
0xb0024014	sser	sser1	r_rec_data
0xb0024018	sser	sser1	rw_extra
0xb002401c	sser	sser1	rw_intr_mask
0xb0024020	sser	sser1	rw_ack_intr
0xb0024024	sser	sser1	r_intr
0xb0024028	sser	sser1	r_masked_intr
0xb0026000	ser	ser0	rw_tr_ctrl
0xb0026004	ser	ser0	rw_tr_dma_en
0xb0026008	ser	ser0	rw_rec_ctrl
0xb002600c	ser	ser0	rw_tr_baud_div
0xb0026010	ser	ser0	rw_rec_baud_div
0xb0026014	ser	ser0	rw_xoff
0xb0026018	ser	ser0	rw_xoff_clr
0xb002601c	ser	ser0	rw_dout
0xb0026020	ser	ser0	rs_stat_din
0xb0026024	ser	ser0	r_stat_din
0xb0026028	ser	ser0	rw_rec_eop
0xb002602c	ser	ser0	rw_intr_mask
0xb0026030	ser	ser0	rw_ack_intr
0xb0026034	ser	ser0	r_intr
0xb0026038	ser	ser0	r_masked_intr
0xb0028000	ser	ser1	rw_tr_ctrl
0xb0028004	ser	ser1	rw_tr_dma_en
0xb0028008	ser	ser1	rw_rec_ctrl
0xb002800c	ser	ser1	rw_tr_baud_div
0xb0028010	ser	ser1	rw_rec_baud_div
0xb0028014	ser	ser1	rw_xoff
0xb0028018	ser	ser1	rw_xoff_clr
0xb002801c	ser	ser1	rw_dout
0xb0028020	ser	ser1	rs_stat_din
0xb0028024	ser	ser1	r_stat_din
0xb0028028	ser	ser1	rw_rec_eop
0xb002802c	ser	ser1	rw_intr_mask
0xb0028030	ser	ser1	rw_ack_intr
0xb0028034	ser	ser1	r_intr
0xb0028038	ser	ser1	r_masked_intr

0xb002a000	ser	ser2	rw_tr_ctrl
0xb002a004	ser	ser2	rw_tr_dma_en
0xb002a008	ser	ser2	rw_rec_ctrl
0xb002a00c	ser	ser2	rw_tr_baud_div
0xb002a010	ser	ser2	rw_rec_baud_div
0xb002a014	ser	ser2	rw_xoff
0xb002a018	ser	ser2	rw_xoff_clr
0xb002a01c	ser	ser2	rw_dout
0xb002a020	ser	ser2	rs_stat_din
0xb002a024	ser	ser2	r_stat_din
0xb002a028	ser	ser2	rw_rec_eop
0xb002a02c	ser	ser2	rw_intr_mask
0xb002a030	ser	ser2	rw_ack_intr
0xb002a034	ser	ser2	r_intr
0xb002a038	ser	ser2	r_masked_intr
0xb002c000	ser	ser3	rw_tr_ctrl
0xb002c004	ser	ser3	rw_tr_dma_en
0xb002c008	ser	ser3	rw_rec_ctrl
0xb002c00c	ser	ser3	rw_tr_baud_div
0xb002c010	ser	ser3	rw_rec_baud_div
0xb002c014	ser	ser3	rw_xoff
0xb002c018	ser	ser3	rw_xoff_clr
0xb002c01c	ser	ser3	rw_dout
0xb002c020	ser	ser3	rs_stat_din
0xb002c024	ser	ser3	r_stat_din
0xb002c028	ser	ser3	rw_rec_eop
0xb002c02c	ser	ser3	rw_intr_mask
0xb002c030	ser	ser3	rw_ack_intr
0xb002c034	ser	ser3	r_intr
0xb002c038	ser	ser3	r_masked_intr
0xb0030000	strcop	strcop	rw_cfg
0xb0032000	ata	ata	rw_ctrl2
0xb0032004	ata	ata	rs_stat_data
0xb0032008	ata	ata	r_stat_data
0xb003200c	ata	ata	rw_ctrl0
0xb0032010	ata	ata	rw_ctrl1
0xb0032014	ata	ata	rw_trf_cnt
0xb0032018	ata	ata	r_stat_misc
0xb003201c	ata	ata	rw_intr_mask
0xb0032020	ata	ata	rw_ack_intr
0xb0032024	ata	ata	r_intr

0xb0032028	ata	ata	r_masked_intr
0xb0034000	eth	eth0	rw_ma0_lo
0xb0034004	eth	eth0	rw_ma0_hi
0xb0034008	eth	eth0	rw_ma1_lo
0xb003400c	eth	eth0	rw_ma1_hi
0xb0034010	eth	eth0	rw_ga_lo
0xb0034014	eth	eth0	rw_ga_hi
0xb0034018	eth	eth0	rw_gen_ctrl
0xb003401c	eth	eth0	rw_rec_ctrl
0xb0034020	eth	eth0	rw_tr_ctrl
0xb0034024	eth	eth0	rw_clr_err
0xb0034028	eth	eth0	rw_mgm_ctrl
0xb003402c	eth	eth0	r_stat
0xb0034030	eth	eth0	rs_rec_cnt
0xb0034034	eth	eth0	r_rec_cnt
0xb0034038	eth	eth0	rs_tr_cnt
0xb003403c	eth	eth0	r_tr_cnt
0xb0034040	eth	eth0	rs_phy_cnt
0xb0034044	eth	eth0	r_phy_cnt
0xb0034048	eth	eth0	rw_test_ctrl
0xb003404c	eth	eth0	rw_intr_mask
0xb0034050	eth	eth0	rw_ack_intr
0xb0034054	eth	eth0	r_intr
0xb0034058	eth	eth0	r_masked_intr
0xb0036000	eth	eth1	rw_ma0_lo
0xb0036004	eth	eth1	rw_ma0_hi
0xb0036008	eth	eth1	rw_ma1_lo
0xb003600c	eth	eth1	rw_ma1_hi
0xb0036010	eth	eth1	rw_ga_lo
0xb0036014	eth	eth1	rw_ga_hi
0xb0036018	eth	eth1	rw_gen_ctrl
0xb003601c	eth	eth1	rw_rec_ctrl
0xb0036020	eth	eth1	rw_tr_ctrl
0xb0036024	eth	eth1	rw_clr_err
0xb0036028	eth	eth1	rw_mgm_ctrl
0xb003602c	eth	eth1	r_stat
0xb0036030	eth	eth1	rs_rec_cnt
0xb0036034	eth	eth1	r_rec_cnt
0xb0036038	eth	eth1	rs_tr_cnt
0xb003603c	eth	eth1	r_tr_cnt
0xb0036040	eth	eth1	rs_phy_cnt

0xb0036044	eth	eth1	r_phy_cnt
0xb0036048	eth	eth1	rw_test_ctrl
0xb003604c	eth	eth1	rw_intr_mask
0xb0036050	eth	eth1	rw_ack_intr
0xb0036054	eth	eth1	r_intr
0xb0036058	eth	eth1	r_masked_intr
0xb0038000	pinmux	pinmux	rw_pa
0xb0038004	pinmux	pinmux	rw_hwprot
0xb0038008	pinmux	pinmux	rw_pb_gio
0xb003800c	pinmux	pinmux	rw_pb_iop
0xb0038010	pinmux	pinmux	rw_pc_gio
0xb0038014	pinmux	pinmux	rw_pc_iop
0xb0038018	pinmux	pinmux	rw_pd_gio
0xb003801c	pinmux	pinmux	rw_pd_iop
0xb0038020	pinmux	pinmux	rw_pe_gio
0xb0038024	pinmux	pinmux	rw_pe_iop
0xb0038028	pinmux	pinmux	rw_usb_phy
0xb003a000	strmux	strmux	rw_cfg
0xb003c000	config	config	r_bootsel
0xb003c004	config	config	rw_clk_ctrl
0xb003c008	config	config	rw_pad_ctrl
0xb003e000	marb	marb	rw_int_slots
0xb003e004	marb	marb	rw_int_slots
0xb003e008	marb	marb	rw_int_slots
0xb003e00c	marb	marb	rw_int_slots
0xb003e010	marb	marb	rw_int_slots
0xb003e014	marb	marb	rw_int_slots
0xb003e018	marb	marb	rw_int_slots
0xb003e01c	marb	marb	rw_int_slots
0xb003e020	marb	marb	rw_int_slots
0xb003e024	marb	marb	rw_int_slots
0xb003e028	marb	marb	rw_int_slots
0xb003e02c	marb	marb	rw_int_slots
0xb003e030	marb	marb	rw_int_slots
0xb003e034	marb	marb	rw_int_slots
0xb003e038	marb	marb	rw_int_slots
0xb003e03c	marb	marb	rw_int_slots
0xb003e040	marb	marb	rw_int_slots
0xb003e044	marb	marb	rw_int_slots
0xb003e048	marb	marb	rw_int_slots
0xb003e04c	marb	marb	rw_int_slots

0xb003e050	marb	marb	rw_int_slots
0xb003e054	marb	marb	rw_int_slots
0xb003e058	marb	marb	rw_int_slots
0xb003e05c	marb	marb	rw_int_slots
0xb003e060	marb	marb	rw_int_slots
0xb003e064	marb	marb	rw_int_slots
0xb003e068	marb	marb	rw_int_slots
0xb003e06c	marb	marb	rw_int_slots
0xb003e070	marb	marb	rw_int_slots
0xb003e074	marb	marb	rw_int_slots
0xb003e078	marb	marb	rw_int_slots
0xb003e07c	marb	marb	rw_int_slots
0xb003e080	marb	marb	rw_int_slots
0xb003e084	marb	marb	rw_int_slots
0xb003e088	marb	marb	rw_int_slots
0xb003e08c	marb	marb	rw_int_slots
0xb003e090	marb	marb	rw_int_slots
0xb003e094	marb	marb	rw_int_slots
0xb003e098	marb	marb	rw_int_slots
0xb003e09c	marb	marb	rw_int_slots
0xb003e0a0	marb	marb	rw_int_slots
0xb003e0a4	marb	marb	rw_int_slots
0xb003e0a8	marb	marb	rw_int_slots
0xb003e0ac	marb	marb	rw_int_slots
0xb003e0b0	marb	marb	rw_int_slots
0xb003e0b4	marb	marb	rw_int_slots
0xb003e0b8	marb	marb	rw_int_slots
0xb003e0bc	marb	marb	rw_int_slots
0xb003e0c0	marb	marb	rw_int_slots
0xb003e0c4	marb	marb	rw_int_slots
0xb003e0c8	marb	marb	rw_int_slots
0xb003e0cc	marb	marb	rw_int_slots
0xb003e0d0	marb	marb	rw_int_slots
0xb003e0d4	marb	marb	rw_int_slots
0xb003e0d8	marb	marb	rw_int_slots
0xb003e0dc	marb	marb	rw_int_slots
0xb003e0e0	marb	marb	rw_int_slots
0xb003e0e4	marb	marb	rw_int_slots
0xb003e0e8	marb	marb	rw_int_slots
0xb003e0ec	marb	marb	rw_int_slots
0xb003e0f0	marb	marb	rw_int_slots

0xb003e0f4	marb	marb	rw_int_slots
0xb003e0f8	marb	marb	rw_int_slots
0xb003e0fc	marb	marb	rw_int_slots
0xb003e100	marb	marb	rw_ext_slots
0xb003e104	marb	marb	rw_ext_slots
0xb003e108	marb	marb	rw_ext_slots
0xb003e10c	marb	marb	rw_ext_slots
0xb003e110	marb	marb	rw_ext_slots
0xb003e114	marb	marb	rw_ext_slots
0xb003e118	marb	marb	rw_ext_slots
0xb003e11c	marb	marb	rw_ext_slots
0xb003e120	marb	marb	rw_ext_slots
0xb003e124	marb	marb	rw_ext_slots
0xb003e128	marb	marb	rw_ext_slots
0xb003e12c	marb	marb	rw_ext_slots
0xb003e130	marb	marb	rw_ext_slots
0xb003e134	marb	marb	rw_ext_slots
0xb003e138	marb	marb	rw_ext_slots
0xb003e13c	marb	marb	rw_ext_slots
0xb003e140	marb	marb	rw_ext_slots
0xb003e144	marb	marb	rw_ext_slots
0xb003e148	marb	marb	rw_ext_slots
0xb003e14c	marb	marb	rw_ext_slots
0xb003e150	marb	marb	rw_ext_slots
0xb003e154	marb	marb	rw_ext_slots
0xb003e158	marb	marb	rw_ext_slots
0xb003e15c	marb	marb	rw_ext_slots
0xb003e160	marb	marb	rw_ext_slots
0xb003e164	marb	marb	rw_ext_slots
0xb003e168	marb	marb	rw_ext_slots
0xb003e16c	marb	marb	rw_ext_slots
0xb003e170	marb	marb	rw_ext_slots
0xb003e174	marb	marb	rw_ext_slots
0xb003e178	marb	marb	rw_ext_slots
0xb003e17c	marb	marb	rw_ext_slots
0xb003e180	marb	marb	rw_ext_slots
0xb003e184	marb	marb	rw_ext_slots
0xb003e188	marb	marb	rw_ext_slots
0xb003e18c	marb	marb	rw_ext_slots
0xb003e190	marb	marb	rw_ext_slots
0xb003e194	marb	marb	rw_ext_slots

0xb003e198	marb	marb	rw_ext_slots
0xb003e19c	marb	marb	rw_ext_slots
0xb003e1a0	marb	marb	rw_ext_slots
0xb003e1a4	marb	marb	rw_ext_slots
0xb003e1a8	marb	marb	rw_ext_slots
0xb003e1ac	marb	marb	rw_ext_slots
0xb003e1b0	marb	marb	rw_ext_slots
0xb003e1b4	marb	marb	rw_ext_slots
0xb003e1b8	marb	marb	rw_ext_slots
0xb003e1bc	marb	marb	rw_ext_slots
0xb003e1c0	marb	marb	rw_ext_slots
0xb003e1c4	marb	marb	rw_ext_slots
0xb003e1c8	marb	marb	rw_ext_slots
0xb003e1cc	marb	marb	rw_ext_slots
0xb003e1d0	marb	marb	rw_ext_slots
0xb003e1d4	marb	marb	rw_ext_slots
0xb003e1d8	marb	marb	rw_ext_slots
0xb003e1dc	marb	marb	rw_ext_slots
0xb003e1e0	marb	marb	rw_ext_slots
0xb003e1e4	marb	marb	rw_ext_slots
0xb003e1e8	marb	marb	rw_ext_slots
0xb003e1ec	marb	marb	rw_ext_slots
0xb003e1f0	marb	marb	rw_ext_slots
0xb003e1f4	marb	marb	rw_ext_slots
0xb003e1f8	marb	marb	rw_ext_slots
0xb003e1fc	marb	marb	rw_ext_slots
0xb003e200	marb	marb	rw_regs_slots
0xb003e204	marb	marb	rw_regs_slots
0xb003e208	marb	marb	rw_regs_slots
0xb003e20c	marb	marb	rw_regs_slots
0xb003e210	marb	marb	rw_intr_mask
0xb003e214	marb	marb	rw_ack_intr
0xb003e218	marb	marb	r_intr
0xb003e21c	marb	marb	r_masked_intr
0xb003e220	marb	marb	rw_stop_mask
0xb003e224	marb	marb	r_stopped
0xb003e240	marb_bp	marb_bp0	rw_first_addr
0xb003e244	marb_bp	marb_bp0	rw_last_addr
0xb003e248	marb_bp	marb_bp0	rw_op
0xb003e24c	marb_bp	marb_bp0	rw_clients
0xb003e250	marb_bp	marb_bp0	rw_options

0xb003e254	marb_bp	marb_bp0	r_brk_addr
0xb003e258	marb_bp	marb_bp0	r_brk_op
0xb003e25c	marb_bp	marb_bp0	r_brk_clients
0xb003e260	marb_bp	marb_bp0	r_brk_first_client
0xb003e264	marb_bp	marb_bp0	r_brk_size
0xb003e268	marb_bp	marb_bp0	rw_ack
0xb003e280	marb_bp	marb_bp1	rw_first_addr
0xb003e284	marb_bp	marb_bp1	rw_last_addr
0xb003e288	marb_bp	marb_bp1	rw_op
0xb003e28c	marb_bp	marb_bp1	rw_clients
0xb003e290	marb_bp	marb_bp1	rw_options
0xb003e294	marb_bp	marb_bp1	r_brk_addr
0xb003e298	marb_bp	marb_bp1	r_brk_op
0xb003e29c	marb_bp	marb_bp1	r_brk_clients
0xb003e2a0	marb_bp	marb_bp1	r_brk_first_client
0xb003e2a4	marb_bp	marb_bp1	r_brk_size
0xb003e2a8	marb_bp	marb_bp1	rw_ack
0xb003e2c0	marb_bp	marb_bp2	rw_first_addr
0xb003e2c4	marb_bp	marb_bp2	rw_last_addr
0xb003e2c8	marb_bp	marb_bp2	rw_op
0xb003e2cc	marb_bp	marb_bp2	rw_clients
0xb003e2d0	marb_bp	marb_bp2	rw_options
0xb003e2d4	marb_bp	marb_bp2	r_brk_addr
0xb003e2d8	marb_bp	marb_bp2	r_brk_op
0xb003e2dc	marb_bp	marb_bp2	r_brk_clients
0xb003e2e0	marb_bp	marb_bp2	r_brk_first_client
0xb003e2e4	marb_bp	marb_bp2	r_brk_size
0xb003e2e8	marb_bp	marb_bp2	rw_ack
0xb003e300	marb_bp	marb_bp3	rw_first_addr
0xb003e304	marb_bp	marb_bp3	rw_last_addr
0xb003e308	marb_bp	marb_bp3	rw_op
0xb003e30c	marb_bp	marb_bp3	rw_clients
0xb003e310	marb_bp	marb_bp3	rw_options
0xb003e314	marb_bp	marb_bp3	r_brk_addr
0xb003e318	marb_bp	marb_bp3	r_brk_op
0xb003e31c	marb_bp	marb_bp3	r_brk_clients
0xb003e320	marb_bp	marb_bp3	r_brk_first_client
0xb003e324	marb_bp	marb_bp3	r_brk_size
0xb003e328	marb_bp	marb_bp3	rw_ack
0xb003e340	marb	marb	rw_no_snoop
0xb003e344	marb	marb	rw_no_snoop_rq

0xb0040000	rt_trace	trace	rw_cfg
0xb0040004	rt_trace	trace	rw_tap_ctrl
0xb0040008	rt_trace	trace	r_tap_stat
0xb004000c	rt_trace	trace	rw_tap_data
0xb0040010	rt_trace	trace	rw_tap_hdata
0xb0040014	rt_trace	trace	r_redir

List of Tables

List of Figures