Ver 1.0

SPARC V8 Rad-Hard MicroProcessor

Datasheet

Part Number：BM3803FMGRH

中国航天 CASC
北京微电子技术研究所
Beijing Microelectronics Technology Institute

Page of Revise Control

Version No.	Publish Time	Revised Chapter	Revise Introduction	Note
1.0	04.16.2019		Document creation	

CONTENTS

1. Features

- Clock: 0MHz～100MHz
- SPARC V8 High Performance 32-bit Architecture
  - Integrated 32/64-bit IEEE 754 Floating-point Unit
  - 32 Kbyte Multi-sets Instruction Cache, 16 Kbyte Multi-sets Data Cache
- Embedded a powerful hardware multiplier/divider
- Memory Interface
  - 8/16/32-bit PROM/SRAM Controller and I/O Space
  - 32-bit SDRAM Controller
- Interrupt Controller with 4 External Programmable Inputs
- Three UARTs
- Two 24-bit Timers
- One 24-bit Watchdog
- 32 Parallel I/O Interface
- PCI Interface
- Fault Tolerance by Design
  - Full Triple Modular Redundancy
  - EDAC Protect
  - Parity Protect
- Debug and Test Facilities
  - Debug Unit (DU) for Trace and Debug
  - Four Hardware Watchpoints
- Operating Voltages
  - 3.3V±0.30V for I/O
  - 1.8V±0.15V for Core
- Operating Temperature
  - -55℃～+125℃
- Storage Temperature
  - -65℃～+150℃
- Power consumption: 1W at 100MHz
- Radiation Performance
  - Total dose radiation capability (parametric & functional)

100Krad(Si)

- SEU error rate better than 8E-5 error/device/day
- Latch up immunity better than 75MeV.cm^2/mg

- Package: CPGA391
- Mass: no more than 33g
- IEEE 1149.1 JTAG Interface
- IDE: SPE-C and SparcSDE

2. General Description

BM3803FMGRH is a highly integrated, high-performance 32-bit RISC embedded rad-hard microprocessor based on the SPARC V8 architecture, which is designed to be used as on-board embedded real-time computer system for space environment. It contains an on-chip Integer Unit (IU), a Floating Point Unit (FPU), separate instruction and data Caches, a hardware Multiplier/Divider, interrupt controller, debug unit with trace buffer, Timers, parallel I/O interface, a Watch-dog Timer, memory controller, a PCI controller and so on. BM3803FMGRH only requires memory and application specific peripherals to be added to form a complete on-board computer, which meets various functionality and performance requirement of space application.

3. Function Block Diagram

BM3803FMGRH function block diagram is shown in figure 3-1.

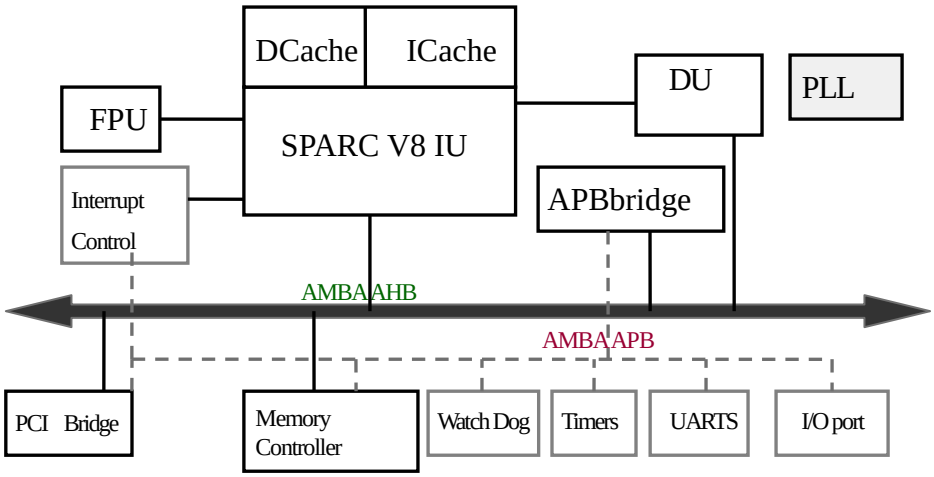


Figure 3-1 Function Block Diagram

4. Pin Description

The whole chip has 225 Signal pins, as follows:

- IU and FPU Signals (67): Address bus 28, Data bus 32, Check bits 7;
- Memory Interface Signals (35): Output enable, Bus ready, Read cycle, Write enable, PROM chip-select 2, SRAM output enable 5, SRAM chip-select 5, SRAM write enable 4, I/O select; SDRAM clock, SDRAM column address strobe(SDCASN), SDRAM chip select 2(SDCSN), SDRAM row address strobe(SDRASN), SDRAM data byte mask 4(SDDQM), SDRAM write enable(SDWEN), PROM bank size define 4(ROMBSD);
- System and clock Signals (4): Processor clock, Processor reset, Bus exception BEXCN, Processor error ERRORN;
- Timer and Watchdog Signals (3): timer0, timer1,wdogn;
- Parallel I/O port and serial port Signals (20): Parallel I/O port 16, extended serial port 4;
- DU Signals (5): DU active(DUACT) signal, DU break enable(DUBRE) signal, DU enable(DUEN) signal, DU receiver signal, DU transmitter signal;
- PCI Arbiter Signals (14): PCI bus request 7, PCI bus grant 7;
- PCI interface Signals (54): PCI Address Data(32), CBE(4), Parity check, Cycle Frame, Initiator Ready(IRDY), Target Ready(TRDY), Stop, Initialization Device

Select, Device Select, PCI bus request, PCI Bus Grant, PCI clock, PCI Reset, Parity check Error, System Error, PCI Host bridge and Guest bridge control, PCI bus interrupt A, PCI bus interrupt B, PCI bus interrupt C, PCI bus interrupt D;

- PLL Signals (7): PLL reset, PLL min 4, PLL output, PLL bypass;
- JTAG TCK Signals (5): TCK(Test Clock), TMS(Test Mode select), TDI(Test data input), TDO(Test data output), TRST(Test Reset);
- PCI test Signals (6): sim_pci_fast(PCI fast reset signal), testin0, testin1, testout0, testout1, pci_66;
- CLK Tree Delay Signals (2): skew(2);
- Several modes Signals (3): scan mode select (scan_mode); mbist mode select (bist_mode); scan mode control (scan_enable);

5. Pin Configurations

Each signal and corresponding pin mapping appears in appendix1.

6. Product Description

6.1 CPU Core IU

6.1.1 Overview

Processor core IU implements the SPARC V8 IU standard definition of the integer instruction set, has the following features:

- 5-stage instruction pipeline
- 8 register windows
- Hardware Multiple/Divider
- Register TMR structure, fault-tolerant registers file EDAC

6.1.2 Instruction Set

BM3803FMGRH instructions are divided into six classes' functional categories:

load/store, arithmetic/logical/shift, control transfer, read/write control register, floating-point, and miscellaneous. Please refer to SPARC V8 Architecture manual that presents all the implemented instructions.

BM3803FMGRH instruction execution cycle is shown in Table 6-1-1.

Table 6-1-1BM3803FMGRH instruction execution cycle (assuming cache hit and no load interlock)

Instruction	Cycle
JMPL	2
Double load	2
Single store	2
Double store	3
SMUL/UMUL	4
SDIV/UDIV	35
Taken Trap	4
Atomic load/store	3
All other instructions	1

6.1.3 Floating Point Unit (FPU)

BM3803FMGRH implements a floating-point unit FPU, which has its own 32-bit registers. The FPU implements all SPARC V8 FPU instructions. It operates on both single-precision and double-precision operands, but it cannot operates on quad-precision operands. Floating-point instruction executes in serial processing with IU instruction and IU is stopped during the execution of floating-point instructions.

6.1.4 Processor start

- operation mode

If DUBRE and DUEN signals are set low, then BM3803FMGRH is in operation mode after reset. When the processor is initiated, the value of PC pointer is 0x0000-0000, so the program should be started in this address space. In the whole design of BM3803FMGRH, this address is also the PROM address, so the program should be started from the PROM address 0x0000-0000. If started from PROM, the corresponding PROM memory data width need to be configured, that expressed by set

the bit PIO[1:0]. “00” indicates 8-bit, “01” indicates 16-bit; “1X” indicates 32-bit. When boot by 8-bit PROM, if PROM EDAC is enabled, the size of rom bank should be determined by external pin rombsd[3:0] before boot.

- debug mode

If DUBRE and DUEN signals are effective, BM3803FMGRH will run 4 instructions from address 0x0000-0000 after reset, and then enter into debug mode. User can debug from DU at this time.

6.1.5 Processor reset

Holding the external reset signal of BM3803FMGRH for 10 clock cycles could make the processor reset. When the processor is reset, after three sdclk cycles, the processor will provide address of PROM, after another sdclk cycle, the processor will provide CS, and Read enable signals of PROM.

6.1.6 Synchronous Traps

BM3803FMGRH implemented traps and their individual priority is shown in Table 6-1-2.

Table 6-1-2 Trap Description

Trap	TT	Priority	Description
reset	0x00	1	Power-on reset
write error	0x2B	2	Write buffer error
instruction_access_error	0x01	3	Error during instruction fetch
illegal_instruction	0x02	5	UNIMP or other un-implemented instruction
privileged_instruction	0x03	4	Execution of privileged instruction in user mode
fp_disabled	0x04	6	FP instruction while FPU disabled
cp_disabled	0x24	6	co-processor instruction while co-processor disabled
watchpoint_detected	0x0B	7	Instruction or data watch point match
window_overflow	0x05	8	SAVE into invalid window
window_underflow	0x06	8	RESTORE into invalid window
register_hardware_error	0x20	9	register file uncorrectable EDAC error
mem_address_not_aligned	0x07	10	Memory access to un-aligned address
fp_exception	0x08	11	FPU exception
data_access_exception	0x09	13	Access error during load or store Instruction
tag_overflow	0x0A	14	Tagged arithmetic overflow

divide_exception	0x2A	15	Divide by zero
interrupt_level_1	0x11	31	Asynchronous interrupt 1
interrupt_level_2	0x12	30	Asynchronous interrupt 2
interrupt_level_3	0x13	29	Asynchronous interrupt 3
interrupt_level_4	0x14	28	Asynchronous interrupt 4
interrupt_level_5	0x15	27	Asynchronous interrupt 5
interrupt_level_6	0x16	26	Asynchronous interrupt 6
interrupt_level_7	0x17	25	Asynchronous interrupt 7
interrupt_level_8	0x18	24	Asynchronous interrupt 8
interrupt_level_9	0x19	23	Asynchronous interrupt 9
interrupt_level_10	0x1A	22	Asynchronous interrupt 10
interrupt_level_11	0x1B	21	Asynchronous interrupt 11
interrupt_level_12	0x1C	20	Asynchronous interrupt 12
interrupt_level_13	0x1D	19	Asynchronous interrupt 13
interrupt_level_14	0x1E	18	Asynchronous interrupt 14
interrupt_level_15	0x1F	17	Asynchronous interrupt 15
trap_instruction	0x80-0xFF	16	Software trap instruction (TA)

6.1.7 Fault Tolerance

IU fault-tolerance design includes two parts: Full triple modular redundancy (TMR) architecture; EDAC protection on regfile.

BM3803FMGRH EDAC protects the data in the register file using Hamming codes, which could detect two bit error and correct one bit error. When corrected data error is detected, the pipeline is halted, and the corrected data is written to the register file according to the former read address, then, restart the pipeline, run the program from the stop position. When non-correct error is detected, handle by trap 0x20, which generates fault register hardware traps.

6.2 Cache Memories

6.2.1 Overview

BM3803FMGRH is a Harvard architecture processor, and implements separate instruction and data buses，which connected to two independent cache controllers. Multi-set-caches are used for both instruction and data caches in order to improve the

speed performance of the CPU core. LRU algorithm is the cache replacement policy used for both instruction and data caches. LRU algorithm means the least recently used set of the cache is replaced when new data need to be stored in cache. The size of the Data Cache space is 16 Kbytes, the size of the Instruction Cache space is 32 Kbytes.

6.2.2 Cache mapping

The cacheable areas include the PROM and RAM areas, while the I/O space and the internal space are non-cacheable. Table 6-2-1 presents the caching capabilities of the processor.

Table 6-2-1 Cache Capability List

Address Range	Area	Cache status
0x0000-0000~0x1FFF-FFFF	PROM	Cached
0x2000-0000~0x3FFF-FFFF	I/O	Non-cacheable
0x4000-0000~0x7FFF-FFFF	RAM	Cached
0x8000-0000~0xFFFF-FFFF	Internal	Non-cacheable

Operation

Different ASI implementation is used to indicate different cache operations. The ASI implementation on the BM3803FMGRH is shown in Table 6-2-2.

Table 6-2-2 ASI Usage of BM3803FMGRH

ASI	Usage
0x0, 0x1, 0x2, 0x3	Force cache miss (replace if cacheable)
0x4, 0x7	Force cache miss (update on hit)
0x5	Flush instruction cache
0x6	Flush data cache
0x8, 0x9, 0xA, 0xB	Normal cache access (replace if cacheable)
0xC	Instruction cache tags
0xD	Instruction cache data
0xE	Data cache tags
0xF	Data cache data

Using the LDA/STA instructions to access various address spaces. During normal operation, the processor accesses instructions and data using ASI 0x8 - 0xB.

ASI 0xC - ASI 0xF are used for the mapping of the instruction cache and data cache.

6.2.3 Diagnostic Cache Control

Parity fault tolerance strategy is used in Cache section. When the parity error appears, read data from external memory directly.

The diagnosis of the Cache fault tolerance is mostly implemented through setting the extension Cache control register. The main functions including building up errors to test the fault tolerance function of Cache area in different Cache areas and count number of errors.

6.3 Traps and Interrupts

6.3.1 Overview

BM3803FMGRH supports three types of traps: synchronous traps, asynchronous traps and interrupts. Synchronous traps and asynchronous traps are caused by hardware responding to a particular instruction: they occur during execution of the instruction that caused them. Interrupts occur when an external event interrupts the processor. They are not related to any particular instruction and occur between the executions of instructions.

Trap Control

Traps are controlled by several registers: exceptions and interrupts request via the enable traps (ET) field in the PSR. Interrupts also request via the processor interrupt level (PIL) field in the PSR. Floating-point exceptions request via the trap enable mask (TEM) in the FSR.

The ET bit in the PSR must be 1 for traps to occur normally. While ET = 1, the IU — between the execution duration of instructions — prioritizes the unsettled exceptions and interrupt requests. At a given time, only the highest priority exception or interrupt request is taken as an exception. When there are multiple unsettled exceptions or interrupt requests, SPARC assumes that lower priority interrupt requests will persist.

- Control by ET and PIL

Exceptions can only be taken when ET field in the PSR is set one.

While ET = 1, if an exception occurs, the processer deals with the exception according to the trap response program. While ET = 0, if an exception occurs, the processor halts execution and enters the error mode.

While ET = 1, if an interrupt occurs, the processer compares PIL in the PSR with ILEVEL and IMASK in the Interrupt Mask and Priority Register. The interrupt will be dealt with if it fits the following terms: 1) the interrupt request level is the highest at this time; 2) the number of the interrupt is the highest at this time and is bigger than PIL; 3) IMASK of this interrupt = 1. While ET = 0, interrupting traps cannot occur, and all interrupt.

- Control by TEM

When a floating exception occurs, if TEM in FSR is set one, then floating exception is enabled. At this time, if ET = 1, the processer deals with the floating exception according to the floating trap response program, and the destination register of the instruction which cause the exception remain unchanged, FCC and AEXC filed of FSR remain unchanged too. If TEM in FSR is set zero, the floating exception is ignored. At this time, the exception is recorded in AEXC filed of FSR, and the destination register of the instruction which cause the exception is refreshed, FCC filed of FSR is refreshed too.

6.3.2 Traps

Trap type

BM3803FMGRH trap types are according with SPARC V8 specification. The table 6-3-1 shows the implemented traps.

Table 6-3-1 Trap Overview

Trap	TT	Description	Priority
reset	0x00	Power-on reset	1
write error	0x2b	Write buffer error	2
instruction_access_exception	0x01	Edac uncorrectable error during instruction fetch	3
illegal_instruction	0x02	UNIMP or other un-implemented instruction	5
privileged_instruction	0x03	Execution of privileged instruction in user mode	4
fp_disabled	0x04	FP instruction while FPU disabled	6

cp_disabled	0x24	co-processor instruction while co-processor disabled	6
watchpoint_detected	0x0B	Instruction or data watch point match	7
window_overflow	0x05	SAVE into invalid window	8
window_underflow	0x06	RESTORE into invalid window	8
register_hadrware_error	0x20	register file uncorrectable EDAC error	9
mem_address_not_aligned	0x07	Memory access to un-aligned address	10
fp_exception	0x08	FPU exception	11
data_access_exception	0x09	Access error during load or store instruction	13
tag overflow	0x0A	Tagged arithmetic overflow	14
divide_exception	0x2A	Divide by zero	15
trap_instruction	0x80 -0xFF	Software trap instruction (TA)	16

Traps Description

- reset: A reset trap is caused by an external reset request. It causes the processor to begin executing at physics address 0. After a Reset Trap, no special memory states are defined except the PSR's 'et' and 's' bits that are initialized respectively '0' and '1'.
- write error: An error exception occurred on a data store to memory.
- instruction_access_exception: A blocking error exception occurred on an instruction access.
- illegal_instruction: An attempt was made to execute an instruction with an unimplemented opcode, or an UNIMP instruction, or an instruction that would result in illegal processor state.
- privileged_instruction: An attempt was made to execute a privileged instruction while supervisor bit (s) in PSR is '0' (not in supervisor mode).
- fp_disabled: An attempt was made to execute an FPU instruction while FPU is not enabled or not present.
- cp_disabled: An attempt was made to execute a co-processor instruction while co-processor is not enabled or not present.
- watchpoint_detected: An instruction fetch memory address or load/store data memory address matched the contents of a pre-loaded implementation-dependent "watchpoint" register.
- window_overflow: A SAVE instruction attempted to cause the current window pointer (CWP) to point to an invalid window in the WIM.
- window_underflow: A RESTORE or RETT instruction attempted to cause the

current window pointer (CWP) to point to an invalid window in the WIM.

- register_hardware_error: An error exception occurred on a read only register access.
- mem_address_not_aligned: A load/store instruction would have generated a memory address that was not properly aligned according to the instruction, or a JMPL or RETT instruction would have generated a non-word-aligned address.
- fp_exception: An FPU instruction generated an IEEE_754_exception and its corresponding trap enable mask (TEM) bit was 1, or the FPU instruction was unimplemented, or the FPU instruction did not complete, or there was a sequence or hardware error in the FPU. The type of floating-point exception is encoded in the FSR's FTT field.
- data_access_exception: A blocking error exception occurred on a load/store data access. EDAC uncorrectable error.
- tag_overflow: A tagged arithmetic instruction was executed, and either arithmetic overflow occurred or at least one of the tag bits of the operands was non zero.
- trap_division_by_zero: An integer divide instruction attempted to divide by zero.
- trap_instruction: A soft instruction (Ticc) was executed and the trap condition evaluated to true.

When multiple synchronous traps occur at the same cycle, the highest priority trap is taken, and lower priority traps are ignored.

IEEE_754_exception floating-point trap type

In floating-point status register (FSR), there are IEEE_754 Floating-Point Exception Fields, the implementation as follows in Figure 6-3-1, Figure 6-3-2 and Figure 6-3-3:

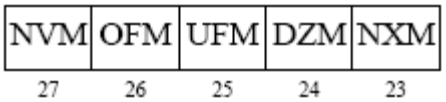


Figure 6-3-1 Trap Enable Mask (TEM) Fields of FSR

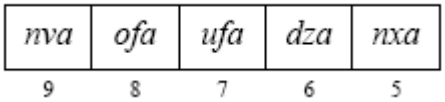


Figure 6-3-2 FSR Accrued Exception Bits (AEXC) Fields of FSR

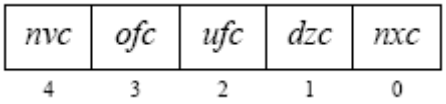


Figure 6-3-3 Current Exception Bits (CEXC) Fields of FSR

FSR_invalid(nvc, nva): An operand is improper for the operation to be performed. For example, $0 \div 0$ and $\infty - \infty$ are invalid. 1 = invalid operand, 0 = valid operand(s).

FSR_overflow(ofc, ofa): The rounded result would be larger in magnitude than the largest normalized number in the specified format. 1 = overflow, 0 = no overflow.

FSR_underflow(ufc, ufa): The rounded result is inexact and would be smaller in magnitude than the smallest normalized number in the indicated format. 1 = underflow, 0 = no underflow.

FSR_division-by-zero(dzc, dza): $X \div 0$, where X is subnormal or normalized. Note that 0. 0 does not set the dzc bit. 1 = division-by-zero, 0 = no division-by-zero.

FSR_inexact(nxc, nxa): The rounded result of an operation differs from the infinitely precise correct result. 1 = inexact result, 0 = exact result.

The occurrence of floating-point IEEE_754_exceptions can be controlled via the user-accessible trap enable mask (TEM) field of the FSR. If a particular bit of TEM is 1, the associated IEEE_754_exception can cause an fp_exception trap.

If a particular bit of TEM is 0, the associated IEEE_754_exception does not cause an fp_exception trap. Instead, the occurrence of the IEEE_754_exception is recorded in the FSR's accrued exception field (aexc).

If a floating-point exception of type IEEE_754_exception results in an fp_exception trap, then the destination f register, FCC, and AEXC fields remain unchanged. However, if an IEEE_754_exception does not result in a trap, then the f register, FCC, and AEXC fields are updated to their new values.

6.3.3 Interrupts

Interrupts Controller

BM3803FMGRH handles 15 interrupts and have two priority levels. Interrupts can be due to external interrupt requests not directly related to any particular instruction or can be due to exception caused by particular previously executed instruction.

The Interrupt Controller Block Diagram is shown in Figure 6-3-4.

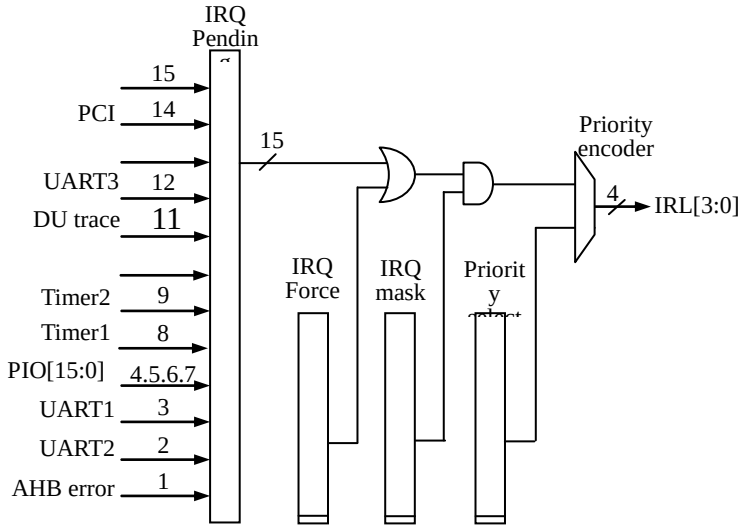


Figure 6-3-4 Interrupt Controller Block Diagram

When an interrupt is generated, the corresponding bit is set in the interrupt pending register (ITP). The highest interrupt from priority level 1 will be forwarded to the IU.

When the IU acknowledges the interrupt, the corresponding pending bit will automatically be cleared. Interrupt can also be forced by setting a bit in the interrupt force register. In this case, the IU acknowledgement will clear the force bit rather than the pending bit. After reset, the interrupt mask register is set to all zeros while the remaining control registers are undefined.

Interrupt List

Interrupt list is shown in Table 6-3-2.

Table 6-3-2 Interrupt Overview

Interrupt	TT	Source	Priority
15	0x1F	unused	17
14	0x1E	PCI	18
13	0x1D	unused	19
12	0x1C	UART3	20

11	0x1B	DU trace buffer	21
10	0x1A	Unused	22
9	0x19	Timer 2	23
8	0x18	Timer 1	24
7	0x17	IO[3]	25
6	0x16	IO[2]	26
5	0x15	IO[1]	27
4	0x14	IO[0]	28
3	0x13	UART 1	29
2	0x12	UART 2	30
1	0x11	AHB bus error	31

I/O interrupts

BM3803FMGRH allows external devices to input interrupt from GPIO. Up to four external interrupts can be programmed at the same time, which are assigned to interrupt 4, 5, 6 and 7. The trigger mode of the four external interrupts can be programmed to low level, high level, falling edge or rising edge by the I/O interrupt register (IOIT).

The IOIT can be divided in to four parts: [31:24] are used for interrupt 7, [23:16] are used for interrupt 6, [15:8] are used for interrupt 5, and [7:0] are used for interrupt 4. Each 8-bit in the IOIT consists of four fields: ENx, LEx, PLx and ISELx. Setting logical one the ENx bit in the IOIT register enables an I/O interrupt. Setting this bit logical zero disables the interrupt. The ISELx field in the IOIT register defines which port of the general purpose interface should generate I/O interrupt x. The 32 bits port includes PIO[15:0] and D[15:0].

The level/edge bit (LEx) and the polarity bit (PLx) are used to program the trigger mode of each interrupt. Each I/O interrupt can have its trigger mode and its polarity individually configured. When the LEx bit is set logical zero, the corresponding I/O interrupt is level sensitive. If the polarity bit is driven logical zero the interrupt triggers when a low level is applied on the pin. If the PLx is driven logical one the interrupt triggers when a high level is applied on the pin. When the LEx bit is set logical one, the corresponding I/O interrupt is edge triggered. If the PLx is driven logical zero the interrupt triggers when a falling edge is applied on the pin. If

the PLx is set logical one the interrupt triggers when a rising edge is applied on the pin.

Table 6-3-3 summarizes the I/O interrupt configurations:

Table 6-3-3 I/O Interrupt Configuration

LEx	PLx	Trigger
0	0	low level
0	1	high level
1	0	falling edge
1	1	rising edge

Interrupt Priority

The 15 interrupts handled by BM3803FMGRH are prioritized, with interrupt 15 (TT = 0x1F) having the highest priority and interrupt 1 (TT = 0x11) the lowest. The interrupt mask and priority register (ITMP) can be used to change the priority level of an interrupt using the two priority levels from. Each interrupt can be assigned to one of two levels as programmed in the Interrupt mask and priority register. Level 1 has higher priority than level 0. Within each level the interrupts are prioritized by the interrupt number.

6.4 Memory Interface

6.4.1 Overview

BM3803FMGRH provides a 32-bit bus capable to interface PROM, asynchronous static rams (SRAM), synchronous dynamic rams (SDRAM) and memories mapped I/O devices. Table 6-4-1 presents the memory controller address map:

Table 6-4-1 Memory Controller address map

Address Range	Size	Mapping
0x0000-0000 ~ 0x1FFF-FFFF	512MB	PROM
0x2000-0000 ~ 0x3FFF-FFFF	256MB	I/O
0x4000-0000 ~	1GB	SRAM/SDRAM

0x7FFF-FFFF		

The three memory controller registers: MCFG1, MCFG2 and MCFG3, are used to configurate the memory interface. MCFG1 is the register dedicated to PROM and IO configuration. SRAM and SDRAM are configured through MCFG2. MCFG3 is the register dedicated to SDRAM configuration.

Figure 6-4-1 is an overview of the 32-bit interconnection between the BM3803FMGRH and external memories.

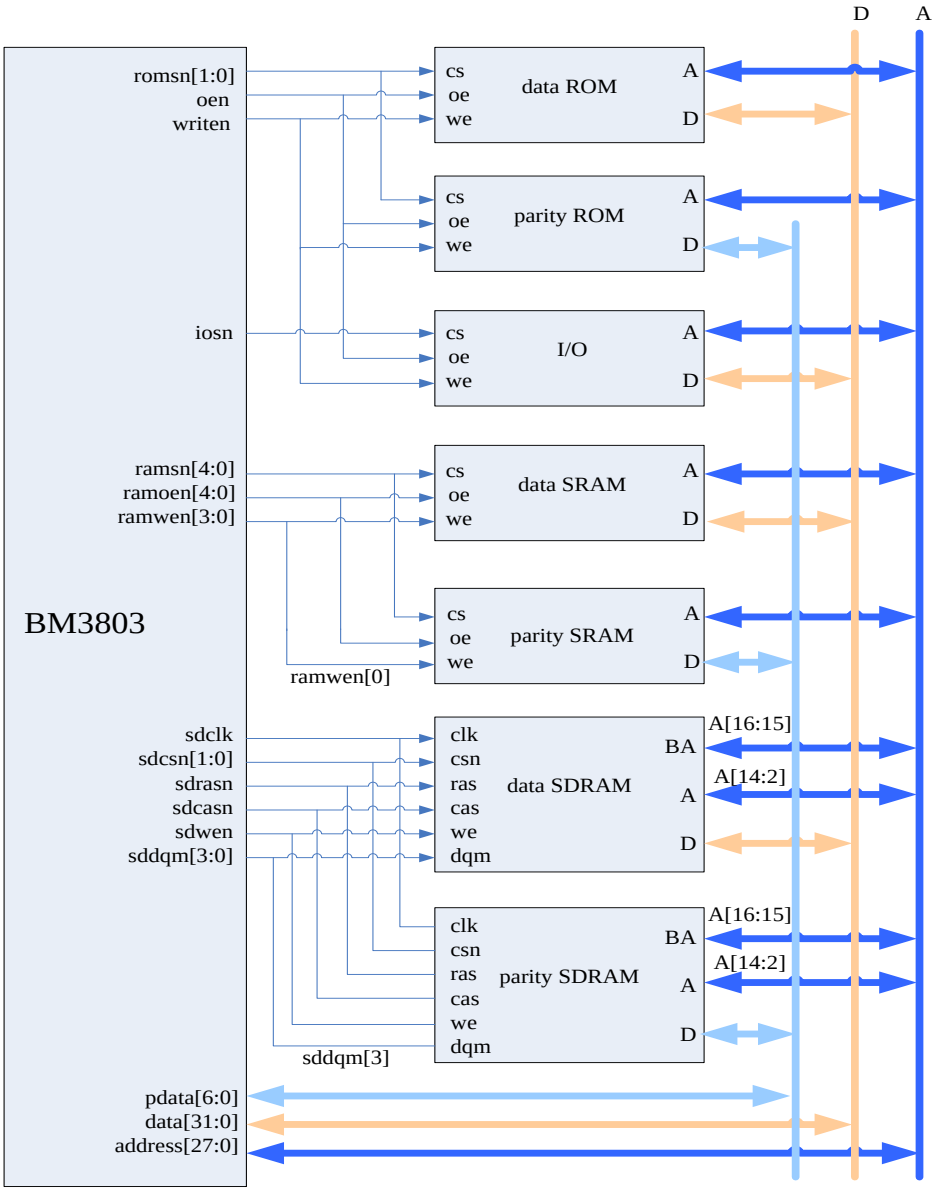


Figure 6-4-1 Memory Interface Overview

6.4.2 RAM Interface

The BM3803FMGRH RAM interface supports two RAM types: asynchronous static RAM (SRAM) and synchronous dynamic RAM (SDRAM). Address space distribution of the SRAM and SDRAM is shown in Table 6-4-2.

Table 6-4-2 RAM address space distribution

SRAM disable（*SRDIS*）	SDRAM enable（*SDREN*）	Address	RAM type
0 or 1	0	0x4000-0000 ~ 0x5FFF-FFFF	SRAM0、1、2、3
		0x6000-0000 ~ 0x7FFF-FFFF	SRAM4
0	1	0x4000-0000 ~ 0x5FFF-FFFF	SRAM0、1、2、3
		0x6000-0000 ~ 0x7FFF-FFFF	SDRAM0、1
1	1	0x4000-0000 ~ 0x7FFF-FFFF	SDRAM0、1

The SRAM interface can manage up to 5 SRAM banks. Size of the four first banks can be programmed in binary step from 8 Kbytes to 256 Mbytes. The fifth SRAM bank size doesn't need program, and the bank size is up to 256 Mbytes large.

The SDRAM interface can manage up to 2 SDRAM banks. The bank size can be programmed from 4 Mbytes to 512 Mbytes.

SRAM Interface

The control of the SRAM memory accesses uses a standard set of pin, including chip selects (RAMSN), output enable (RAMOEN) and write enable (RWEN) lines. The bank size of RAMSN[3:0] banks of the SRAM area can be configured by setting MCFG2[12:9], the value of the SRAM bank size field in MCFG2. Size of RAMSN[3:0] banks can be programmed from 8 Kbytes to 256 Mbytes. When the size is 256 Mbytes, only RAMSN [1:0] is useful instead of RAMSN [3:0]. The fifth SRAM bank controlled by RAMSN [4] is always 256 Mbytes large, and always resides at the upper address 0x60000000.

SRAM data bus width can be configured to 8-bit, 16-bit and 32-bit. The bit width size can be configured by MCFG2[5:4], the value "00" indicates 8-bit bus, "01" indicates 16-bit bus, "1x" indicates 32-bit bus.

When SRAM data width is configured 8-bit, processor pins data[31:24] are linking to pins of data SRAM. The LSB address bit of the data SRAM is linking to

address[0] of the processor.

When SRAM data width is configured 16-bit, processor pins data[31:16] are linking to pins of data SRAM. The LSB address bit of the data SRAM is linking to address[1] of the processor.

When SRAM data width is configured 32-bit, processor pins data[31:0] are linking to pins of data SRAM. The LSB address bit of the data SRAM is linking to address[2] of the processor. If EDAC is enaled in a 32-bit SRAM, then parity bit pdata[6:0] of the processor is linking to the parity SRAM.

Write operation of SRAM can be divided into two mode: read-modify-write mode or write through mode. It is configured by MCFG2 [6] bit.

Setting MCFG2 [6] to one programs SRAM to read-modify-write mode. The 4 pin RWEN [3:0] will be effective or not at the same time. When SRAM data width is configured 32-bit, for write access by byte or half-word, the word including the byte or half-word would be read at first, then the controller modify the byte or half-word in the word inside the chip, at last, the whole word would be written back into the SRAM. When SRAM data width is configured 16-bit, for write access by byte, the half-word including the byte would be read at first, then the controller modify the byte in the half-word inside the chip, at last, the half-word would be written back into the SRAM.

Setting MCFG2 [6] to zero programs SRAM to write through mode. The 4 pin RWEN [3:0] will be effective or not respectively. For write access by byte or half-word, RWEN according to the byte or half-word would be active, while other RWEN are not active.

Table 6-4-3 shows the corresponding relationship between RWEN [3:0] and data[31:0]。

Table 6-4-3 RWEN of SRAM and its corresponding data

RWEN of SRAM	corresponding data byte
RWEN[0]	data[31:24]
RWEN[1]	data[23:16]
RWEN[2]	data[15:8]
RWEN[3]	data[7:0]

The external signal BRDYN could also be used for the fifth bank SRAM to delay fix the external read/write time. The use of BRDYN is configured by MCFG2[7]. When MCFG2[7] is set high, BRDYN need to be driven low level before SRAM read/write finish.

The Figure 6-4-2, Figure 6-4-3, and Figure 6-4-4 show the SRAM read cycle access timing:

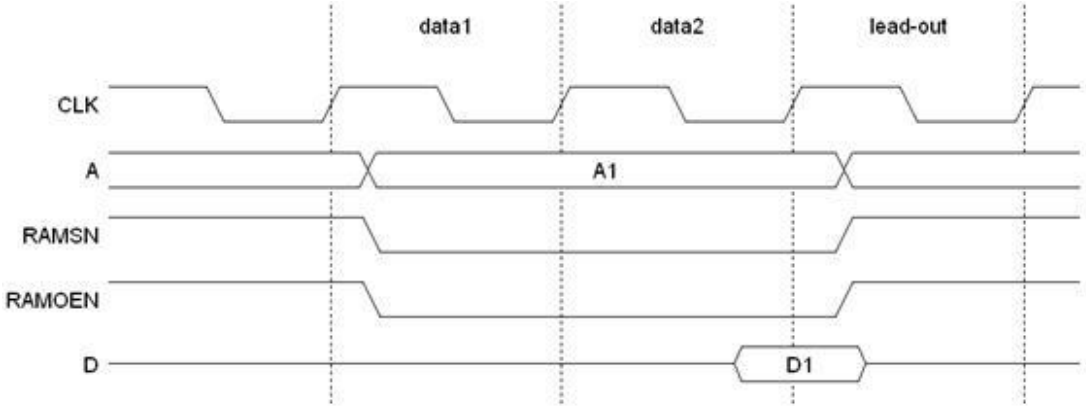


Figure 6-4-2 SRAM read cycle (EDAC disable, no delay)

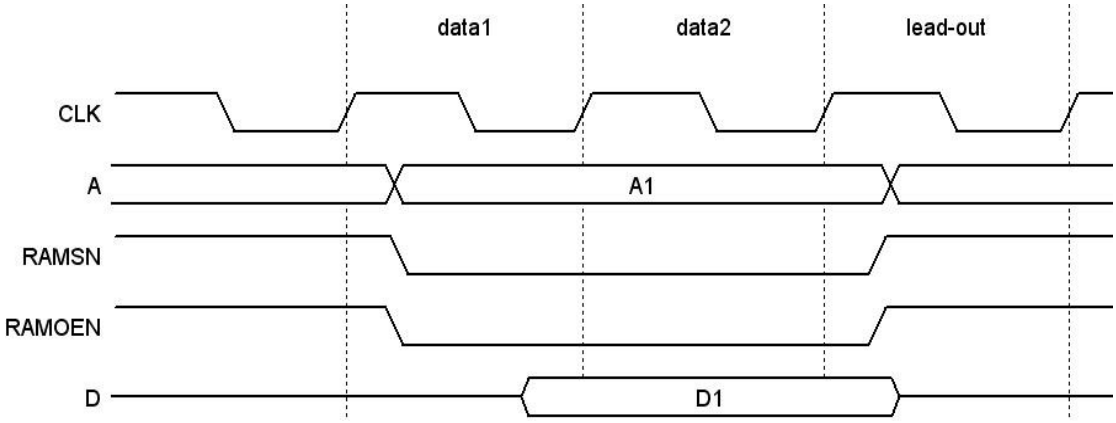


Figure 6-4-3 SRAM read cycle (EDAC enable, no delay)

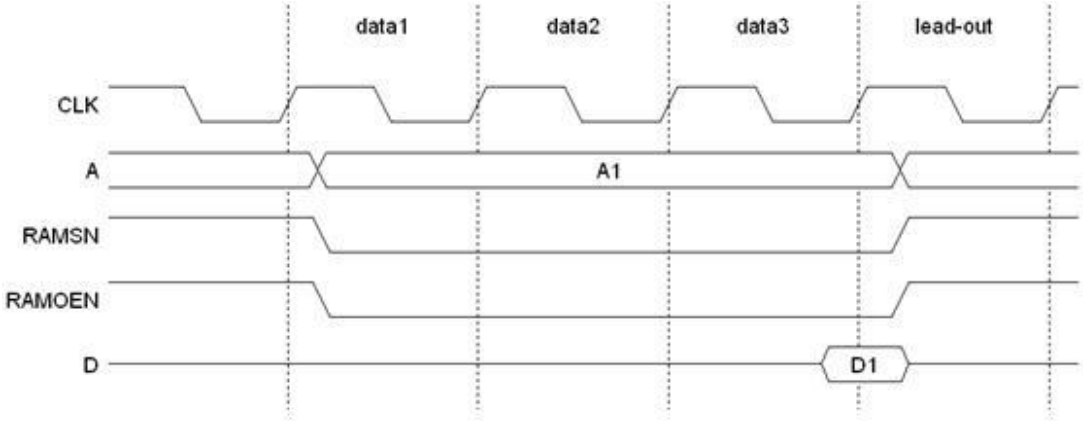


Figure 6-4-4 SRAM read cycle (EDAC disable, 1 clk delay)

The address in “lead_out” cycle of SRAM read cycle is not from the address in the former cycle, but from the processor internal bus address in the former cycle.

The Figure 6-4-5 , and Figure 6-4-6 show the SRAM write cycle access timing:

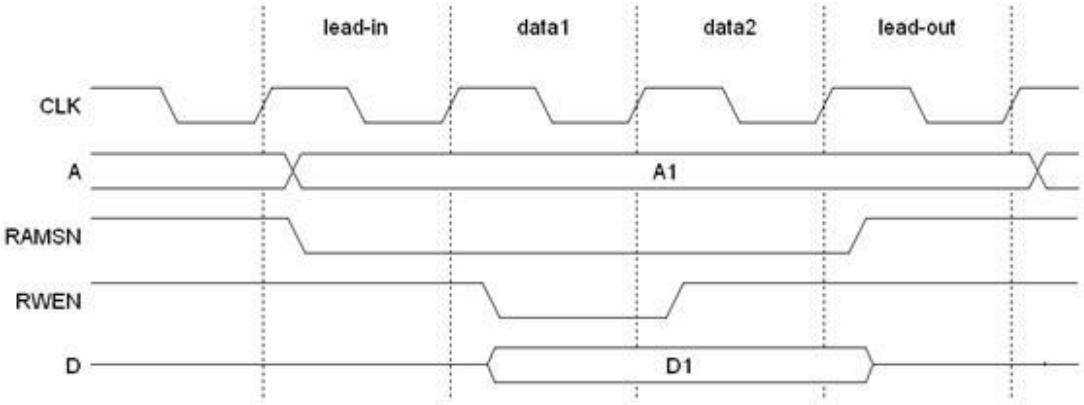


Figure 6-4-5 SRAM write cycle (no delay)

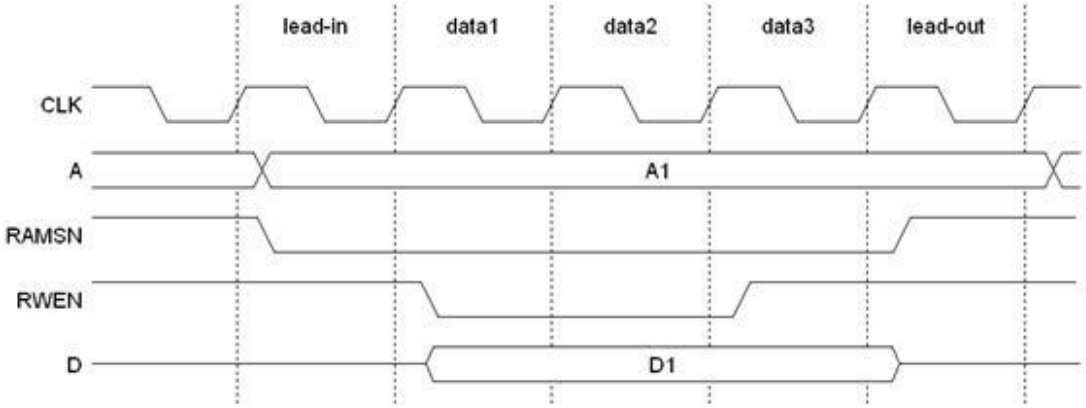


Figure 6-4-6 SRAM write cycle (1 clk delay)

SDRAM Interface

The control of the SDRAM memory accesses uses a standard set of pin, including chip selects (SDCSN), row address select (SDRASN), column address select (SDCASN), write enable (SDWEN), data masks (SDDQM) and clock lines (SDCLK). The bank size of the two SDRAM banks can be configured by setting MCFG2 [25:23], the value of the SDRAM bank size field in MCFG2. Each bank size can be programmed in binary step from 4 Mbytes to 512 Mbytes. The controller supports 64M, 256M and 512M devices. The column-address of the devices is from 8 to 12 bits, row-address is up to 13 bits, and there are 4 banks inside the chip. SDRAM banks only support 32-bit data bus width.

The address bus of the SDRAMs shall be connected to the processor address [14:2], the bank address connected to the processor address [16:15]. Devices with less than 13 address pins should only connect to the less significant bits of address [14:2].

The data bus of the SDRAMs shall be connected to the processor data [31:0]. If EDAC enabled, the parity pin of the processor pdata[6:0] shall be connected to parity SDRAM.

● SDRAM Timing Parameters

To provide optimum access cycles for different SDRAM devices some SDRAM parameters can be programmed through MCFG2 and MCFG3. Table 6-4-4 shows the programmable SDRAM parameters:

Table 6-4-4 SDRAM Programmable Timing Parameters

Function	Parameter	Range	Unit
CAS latency	TCAS	2 - 3	clocks
Precharge to activate	TRP	2 - 3	clocks
Auto-refresh command period	TRFC	3 - 11	clocks
Auto-refresh interval	TREFRESH	10 - 32768	clocks

● SDRAM Commands

The SDRAM controller can handle three SDRAM commands. Commands to be executed are programmed through MCFG2[20:19], the SDRAM command field in the memory configuration register. When a nonzero value is written to this field, a SDRAM command is issued:

“01”: Precharge command is sent;

“10”: Auto-Refresh command is sent;

“11”: Load Mode Reg (LMR) is sent.

MCFG2[20:19] is cleared after a command is executed.

The SDRAM access mode, including the CAS delay period and other fixed mode, is configured when the LMR command is issued. Other fixed mode means page burst on read, single location access on write and burst in order. CAS latency is programmed by MCFG2[26], which names SdrCAS. A CAS latency of 2 is programmed when SdrCAS is 0, and a CAS latency of 3 is programmed when SdrCAS is 1. When changing the value of the CAS delay, a LOAD-MODE-REGISTER command must be generated at the same time.

When the LMR command is issued, the output value of each address pin is

shown in table 6-4-5. The value will be load into the SDRAM mode registers at the same time.

Table 6-4-5 The output value when the LMR command is issued

SDRAM		BM3803FMGRH	
address	Mode Reg	address	Output value
BA0	M13	address[15]	0
A12	M12	address[14]	0
A11	M11	address[13]	0
A10	M10	address[12]	0
A9	M9	address[11]	1
A8	M8	address[10]	0
A7	M7	address[9]	0
A6	M6	address[8]	0
A5	M5	address[7]	1
A4	M4	address[6]	SDRCAS
A3	M3	address[5]	0
A2	M2	address[4]	1
A1	M1	address[3]	1
A0	M0	address[2]	1

The SDRAM controller provides a refresh by cycle command. It can be enabled by setting logical one SDRREF, the refresh enable bit in the memory configuration register MCFG2[31]. The Auto-Refresh command is sent to both SDRAM banks at the same time, and enables a periodical refresh for both SDRAM banks. The period between two Auto-Refresh commands is programmed in MCFG3[26:12], the refresh counter reload field of the third memory configuration register.

Required period is typically 7.8μs or 15.6μs, depending on SDRAM type. This corresponds to780or 1560 clock cycle at 100MHz.

Refresh period is calculated as:

Refresh Period = (Reload value+1)/ sysclk

- SDRAM Initialization

After enable SDRAM by setting logical one MCFG2[14], the SDRAM controller automatically performs the SDRAM initialization sequence to both banks simultaneously. It consists of PRECHARGE, two AUTO-REFRESH cycles and LMR.

- SDRAM Read Access

A read cycle consists of three main operations.

First, an ACTIVE command to the desired bank and row is sent.

Then, after the programmed CAS delay, a READ command is performed.

The third step, a PRE-CHARGE command terminates the read cycle.

No bank is left open between two accesses. A burst read is performed if a burst access is requested on the internal bus.

Figure 6-4-7 shows the SDRAM read cycle access timing:

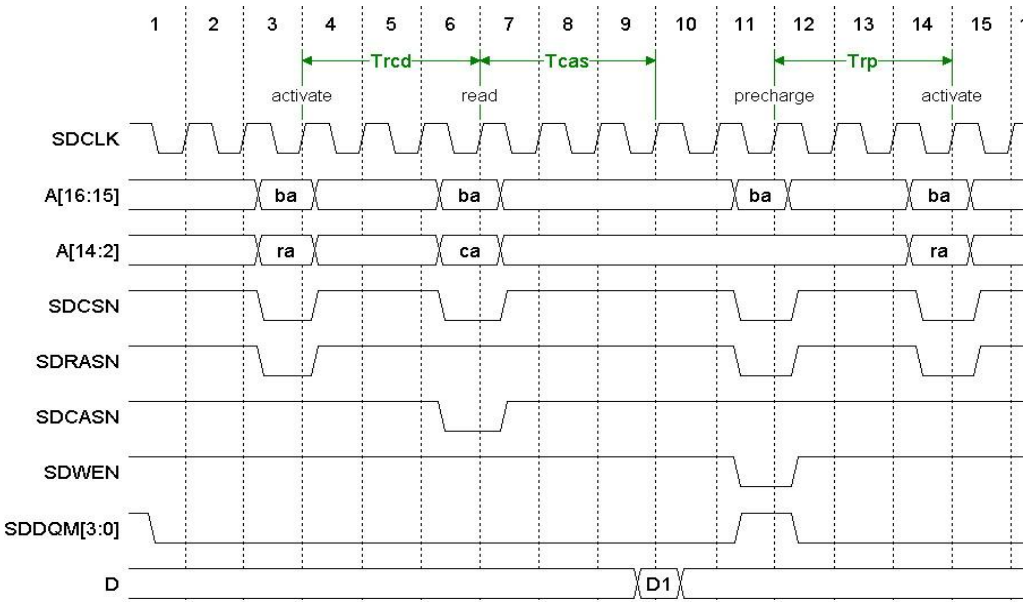


Figure 6-4-7 SDRAM read cycle (Burst length = 1; CL = 3)

- SDRAM Write Access

A write cycles consists of three main operations.

First, an ACTIVE command to the desired bank and row is sent.

Then, after the programmed CAS delay, a WRITE command is performed.

The third step, a PRE-CHARGE command terminates the write cycle.

A burst write on the internal bus generates a burst of write commands without idle cycle in-between.

Figure 6-4-8 shows the SDRAM read cycle access timing:

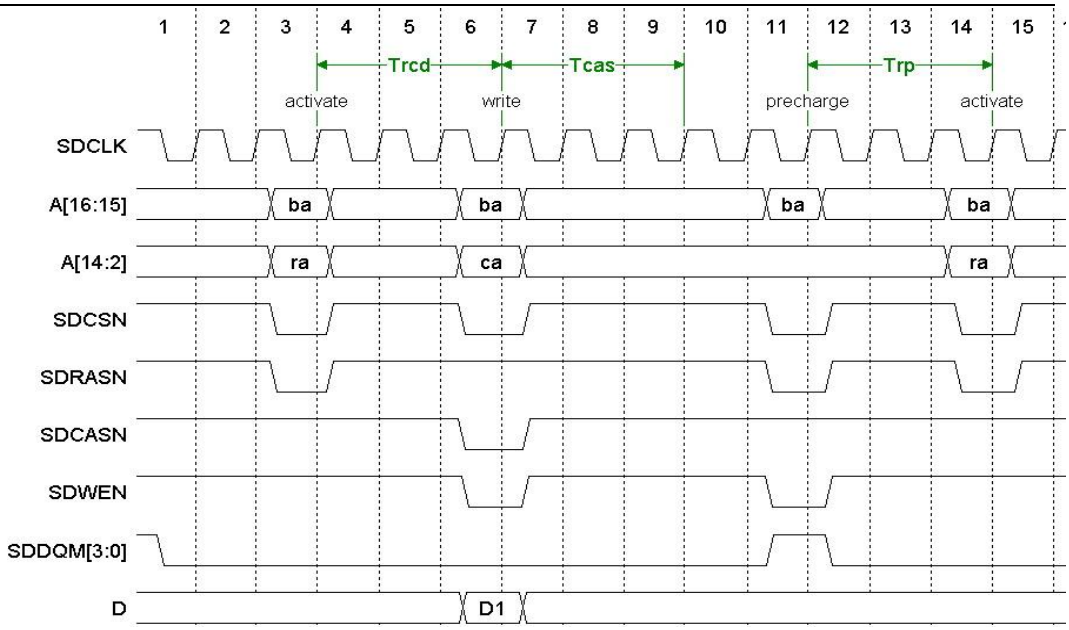


Figure 6-4-8 SRAM write cycle (Burst length = 1; CL = 3)

6.4.3 PROM Interface

The PROM interface can manage up to two PROM banks. The bank size is not programmable, and each bank size is up to 256 Mbytes. The control of the PROM memory accesses uses a standard set of pin, including chip selects (ROMSN), output enable (OEN) and write (WRITE) lines. The lower half part of the PROM area (0x00000000 up to 0x0FFFFFFF) is controlled by the PROM select signal ROMSN0. The upper half part of the PROM area (0x10000000 up to 0x1FFFFFFF) is controlled by the PROM select signal ROMSN1.

PROM access mainly through configure MCFG1 register to choose various states. PROM write and read access wait cycles could be set separately through set MCFG1[16:12] and MCFG1[4:0], wait cycle from 0 to 31. MCFG1[11] determine the PROM write enable or not. MCFG1[9:8] select the data width of PROM.

When PROM data width is configured 8-bit, processor pins data[31:24] are linking to pins of data PROM. The LSB address bit of the data PROM is linking to address[0] of the processor.

When PROM data width is configured 16-bit, processor pins data[31:16] are linking to pins of data PROM. The LSB address bit of the data PROM is linking to address[1] of the processor.

When PROM data width is configured 32-bit, processor pins data[31:0] are linking to pins of data PROM. The LSB address bit of the data PROM is linking to address[2] of the processor. If EDAC is enabled in a 32-bit PROM, then parity bit pdata[6:0] of the processor is linking to the parity PROM.

Figure 6-4-9, Figure 6-4-10 and Figure 6-4-11 shows the PROM read cycle access timing:

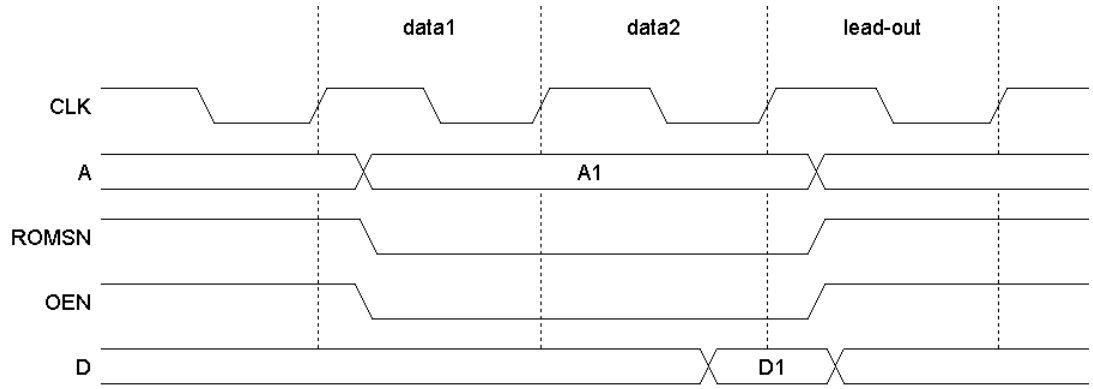


Figure 6-4-9 PROM read cycle (EDAC disable, no delay)

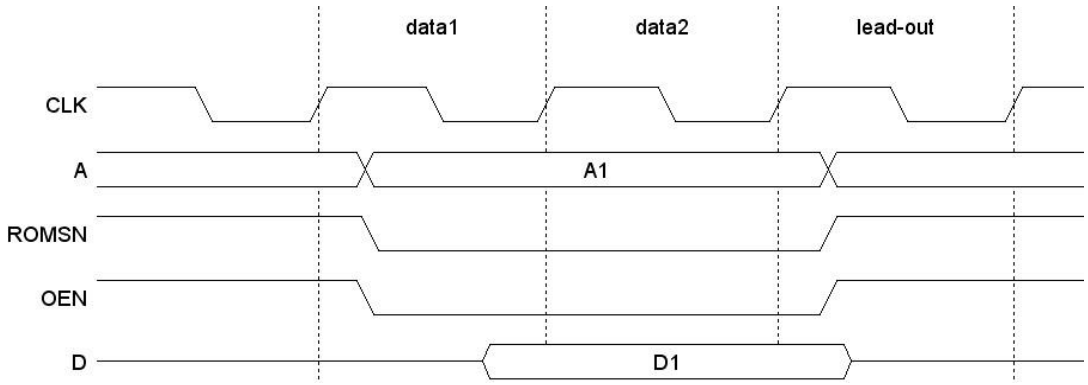


Figure 6-4-10 PROM read cycle (EDAC enable, no delay)

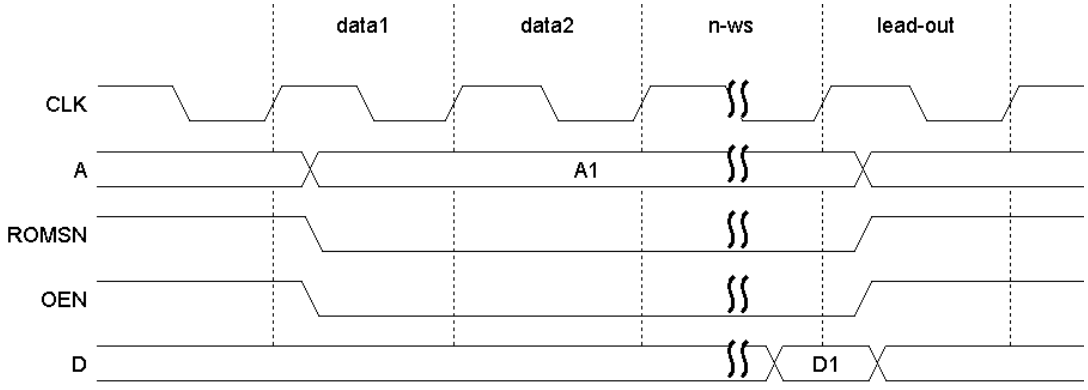


Figure 6-4-11 PROM read cycle (EDAC enable, n clk delay)

The address in “lead_out” cycle of PROM read cycle is not from the address in the former cycle, but from the processor internal bus address in the former cycle.

Figure 6-4-12 and Figure 6-4-13 shows the PROM write cycle access timing:

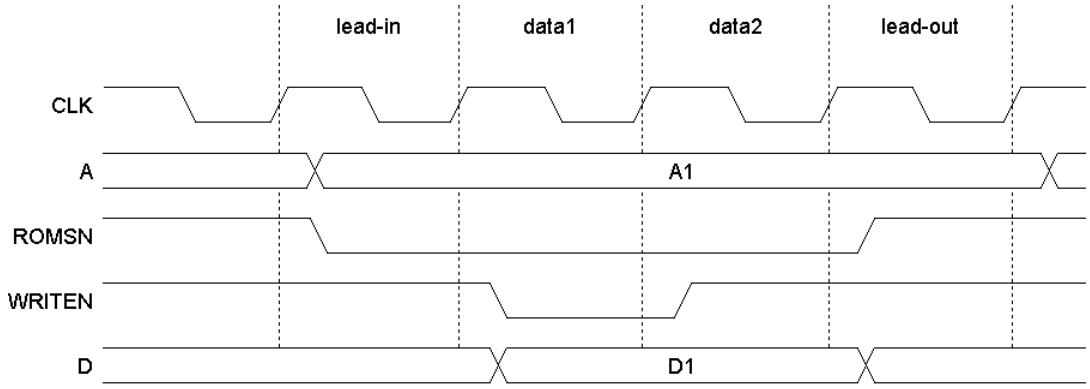


Figure 6-4-12 PROM write cycle (no delay)

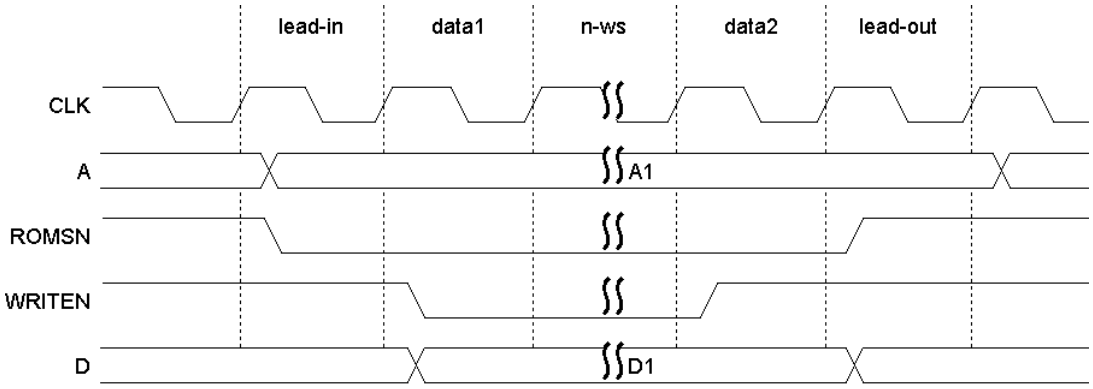


Figure 6-4-13 PROM write cycle (n clk delay)

6.4.4 Memory Mapped I/O

The I/O area can manage only one large bank. The bank size is not programmable, and the bank size is up to 256 Mbytes. The address of I/O area is 0x2000-0000 ~ 0x3FFF-FFFF, among which 0x2000-0000 ~ 0x2FFF-FFFF and 0x3000-0000 ~ 0x3FFF-FFFF are overlapped. That means access 0x3000-0000 is the same with access 0x2000-0000, and access 0x3000-0001 is the same with access 0x2000-0001, etc. The control of the I/O area accesses uses a standard set of pin, including chip selects (IOSN), output enable (OEN) and write (WRITE) lines.

The I/O device data bus width is controlled by MCFG1[28:27], the value “00”

indicates 8-bit bus, “01” indicates 16-bit bus, “1x” indicates 32-bit bus.

When I/O device data width is configured 8-bit, processor pins data[31:24] are linking to data pins of the I/O device. The LSB address bit of the I/O device is linking to address[0] of the processor.

When I/O device data width is configured 16-bit, processor pins data[31:16] are linking to data pins of the I/O device. The LSB address bit of the I/O device is linking to address[1] of the processor.

When I/O device data width is configured 32-bit, processor pins data[31:0] are linking to data pins of the I/O device. The LSB address bit of the I/O device is linking to address[2] of the processor.

The read/write cycle of 8/16-bit data bus I/O device is the same with 32-bit I/O device, the only difference is the data bus width.

When I/O device data bus width is configured 8-bit, only LDUB and STB instruction are supported.

When I/O device data bus width is configured 16-bit, only LDUH and STH instruction are supported.

Figure 6-4-14 shows the I/O space read cycle access timing:

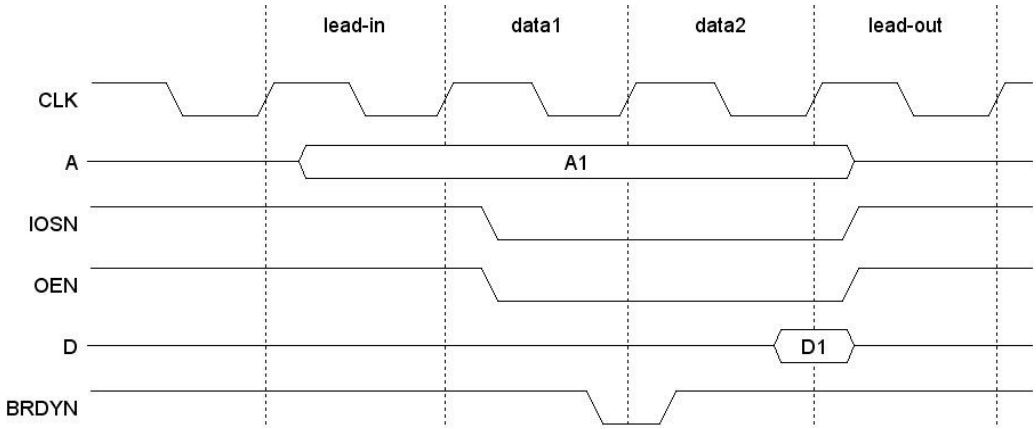


Figure 6-4-14 I/O read cycle (no delay)

The address in “lead_out” cycle of I/O read cycle is not from the address in the former cycle, but from the processor internal bus address in the former cycle.

Figure 6-4-15 shows the I/O space write cycle access timing:

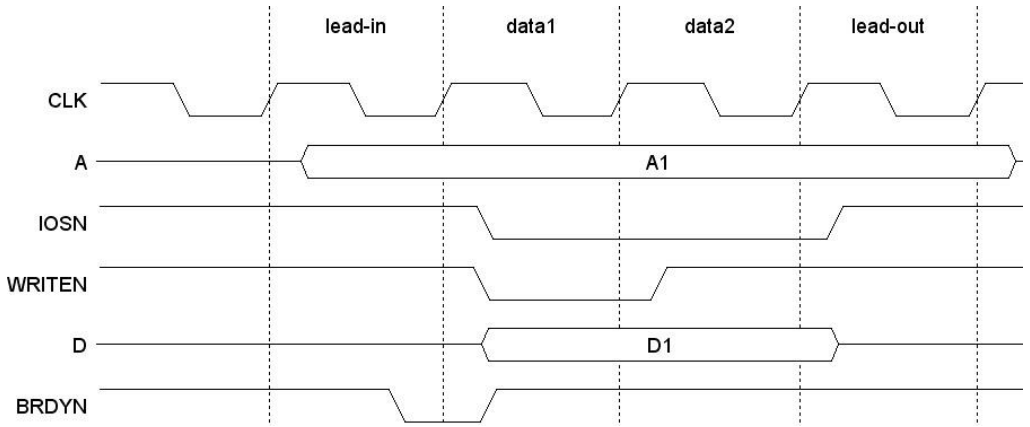


Figure 6-4-15 I/O write cycle (no delay)

I/O device access read enable cycle and write enable cycle can be defined by MCFG1[23:20], the IOWS bit, and the max waiting cycle is 15. If set IOWS=15 still cannot meet the I/O device access, access can be delayed through external BRDYN signal. MCFG1[26], the brdy bit, is used to set BRDYN, one indicates enable BRDYN and zero indicates disable BRDYN. When BRDYN enables, for I/O device access, the processor detect brdyn pin signal after waiting cycle set by IOWS. The waiting cycle ends when brdyn is driven low level, and the read/write signal ends one clk cycle after brdyn is driven low.

6.4.5 Access Error

For interface PROM, SRAM and memories mapped I/O devices, an access error can be indicated to the processor asserting the BEXCN signal.

If enabled by setting logical one MCFG1[25], the BEXC bit, the BEXCN signal is sampled with the data. If the BEXCN signal is driven low by the external device, an error response is generated on the internal bus.

If an instruction fetch is in progress, trap 0x01 is taken;

If a data load is in progress, trap 0x09 is taken;

If a data store is in progress, trap 0x2B is taken.

6.4.6 Error Management - EDAC

BM3803FMGRH processor implements EDAC for external memory access by

an on-chip error detector and corrector. The on-chip memory EDAC can correct one-bit error in a 32-bit word and detect two-bit errors in a 32-bit word. The error bit number counts both error in the 32-bit data bit and error in the 7-bit parity bit. The data bit and parity bit are treated as the same when EDAC.

EDAC capability can be used on 8/32-bit PROM, 8/32-bit SRAM and 32-bit SDRAM memory areas.

For SDRAM, when EDAC enable, internal AHB bus cannot use burst mode.

For 32-bit PROM/SRAM/SDRAM, 7-bit parity bank of PROM/SRAM/SDRAM is needed for the implementation of EDAC. For 8-bit PROM/SRAM, there is no need of parity bit because the data bit and the parity bit are store in the same bank. Data bit store in the front 3/4 bank while parity bit store in the back 1/4 bank. The first 4 byte in the 8-bit bank, also the first word, is corresponding to the 7-bit parity bit from the 3/4 bank.

The following Table 6-4-6 presents the EDAC protection capabilities provided by the processor.

Table 6-4-6 Memory EDAC configuration register MECFG1(0x80000100)

Bits	r/w	Name	Description
31	r/w	Sr1EEn	SRAM bank1 EDAC enable, high-effective.
30	r/w	Sr2EEn	SRAM bank2 EDAC enable, high-effective.
29	r/w	Sr3EEn	SRAM bank3 EDAC enable, high-effective.
28	r/w	Sr4EEn	SRAM bank4 EDAC enable, high-effective.
27	r/w	Sr5EEn	SRAM bank5 EDAC enable, high-effective.
26	r/w	romEEn	Prom EDAC enable, high-effective.The initialvalue of this field is reset on power configured by the GPIO bit 2.
25:23			Reserved.
22	r/w	EWB2	EDAC write bypass(EWB2), when EWB2, 1, 0=“011” effective
21	r/w	ERB	EDAC read bypass (ERB), high-effective.
20	r/w	EWB1	EDAC write bypass(EWB1), when EWB2, 1, 0=“011” effective
19	r/w	SdEEn	SDRAM EDAC enable, high-effective.
18	r/w	EWB0	EDAC write bypass (EWB0), when EWB2, 1, 0=“011” effective
17:14	r/w	PRBS	PROM bank size. Define the size of each PROM. (“0000”=8 Kbyte, “0001”=16 Kbyte , … , “1111”=256 Mbyte).Used for 8-bit prom EDAC. This field’s initial value is configured by external pin ROMBSD [3:0] when power-on reset.
13:10			Reserved.
9:8	r/w	TCBarea	TCB area. Each bank of the SRAM and the SDRAM is divided into 4

			parts area averagely. In the field, the data parity accessed can be stored into TCB fields, which will not be controlled by TCB area field except the SRAM in the fifth bank.
7			Reserved.
6:0	r/w	TCB	Test check bits (TCB).Used to test/calibration. If the ERB in MECFG1 is enabled, the normal parity read stored into this field.

This register implements enable-control of EDAC for different SRAM and SDRAM BANK. At the same time, it can verify the record data parity bit when EDAC enable.

MECFG1[26] romEEn is configured by GPIO[2] when power-on reset, after reset it could be configured by MECFG1. If GPIO[2] is‘1’, then romEEn is configured‘1’ when power-on reset.

MECFG1[17:14] PRBS is configured by ROMBSD[3:0] when power-on reset, after reset it could be configured by MECFG1.

EWB write bypass (for PROM):

- EWB write bypass enable or not has no effect on PROM.

EWB write bypass (for SRAM and SDRAM):

- No matter EDAC is enabled or not, if EWB is effective and the operation to 32-bit data width SRAM and SDRAM is written of words, then write data bit and write parity bit error is produced, that means the data bit is written reverse the MECFG2 and the parity bit is written reverse the PR field in MECFG3.
- No matter EDAC is enabled or not, if EWB is effective and the operation to 8-bit data width SRAM is written of words, then write parity error is produced, that means the parity bit written reverse the PR field in MECFG3.
- For 8-bit data width SRAM, if EWB is effective and stb or sth instruction is executing, then data bit error will be created, that is, the written data is the error data
- As long as EWB is valid, the error correction is not enabled.
- As long as EWB is invalid, for 32-bit SRAM and SDRAM, normal data bit and normal parity bit will be written, error will not appear.
- When EDAC is enabled, if EWB is invalid, for 8-bit SRAM, normal data bit and normal parity bit will be written, error will not appear.
- When EDAC is enabled, if EWB is invalid, then the error correction is enabled, which also names EDAC write enable.

ERB read bypass (for PROM):

- TCBarea has no effect on PROM. If ERB is effective while reading 8/32-bit PROM, the normal parity will be stored into TCB field in MECFG1.

ERB read bypass (for SRAM and SDRAM):

- No matter EDAC is enabled or not, if ERB is effective while read access to 8/32-bit SRAM and SDRAM, then the parity bit is stored into TCB field. Otherwise, TCB field is maintained.
- The fifth bank of SRAM is not controlled by TCBarea. When ERB enable, the parity bit of all data will be save to TCB of MECFG1.
- As long as ERB is enabled, the error correction is not enabled.
- As long as ERB is invalid, the TCB field is maintained.
- When EDAC is enabled, if ERB is invalid, then the error correction is enabled, also named EDAC read enable.

EDAC, EWB, ERB in MECFG1 (for SRAM and SDRAM):

- Combination 1（MECFG1_1）: When set EDAC enabled, EWB is invalid and ERB is cleared, it is EDAC error detection and correction setting. The correct data bit and the correct parity bit can be written and the actual parity bit cannot be read out.
- Combination 2（MECFG1_2）: When set EDAC enabled, EWB is invalid and ERB set, the correct data bit and the correct parity bit can be written, and the actual data bit and parity bit can be read. This combination set is for normal work and the normal test that is used to write the correct data bit and the correct parity bit, and read the actual data bit and the actual parity bit. This combination set neither detect error nor correct error.
- Combination 3（MECFG1_3）: When set EDAC enabled, EWB is effective and ERB cleared, cooperated with all 0 PR field in MECFG2 and MECFG1 is for EDAC error detection and not error correction.
- Combination 4（MECFG1_4）: When set EDAC enabled, EWB is effective and ERB set, for 32-bit SRAM and SDRAM, the wrong data bit and the wrong parity bit can be written and the actual data bit and the actual parity bit can be read, the parity bit is stored into TCB field. 8-bit SRAM cannot write the wrong data bit, can write the wrong parity bit, can read the actual data bit and the actual parity bit, the parity bit is stored into TCB field. This combination

set is for test, used to write the wrong data bit and the wrong parity bit, and read the actual data bit and the actual parity bit for 32-bit SRAM and SDRAM. And it is used to write the wrong parity bit and read the actual data bit and the actual parity bit for 8-bit SRAM. This combination set neither detect error nor correct error.

- Combination 5（MECFG1_5）:When set EDAC is not enabled, EWB is invalid and ERB cleared, for 32-bit SRAM and SDRAM，can write the correct data bit and the correct parity bit, cannot read the actual parity bit. For 8-bit SRAM can write the correct data bit, cannot write the parity bit, cannot read the actual parity bit. This combination set is for test. For 32-bit SRAM and SDRAM，it is used to write the correct data bit and the correct parity bit, the actual parity bit cannot be read. This combination set neither detect error nor correct error. This combination set is usually unused.
- Combination 6（MECFG1_6）: When set EDAC is not enabled, and EWB is invalid, and ERB set, for 32-bit SRAM and SDRAM，write the correct data bit and the correct parity bit, and read the actual data bit and the actual parity bit. For 8-bit SRAM can write the correct data bit, cannot write the parity bit, can read the actual data bit and the actual parity bit. This combination set is for test, used to write the correct data bit and the correct parity bit, and read the actual data bit and the actual parity bit for 32-bit SRAM and SDRAM. This combination set neither detect error nor correct error.
- Combination 7（MECFG1_7）: When set EDAC is not enabled, EWB is effective and ERB is cleared, for 32-bit SRAM and SDRAM can write the wrong data bit and the wrong parity bit, cannot read the actual parity bit. For 8-bit SRAM cannot write the wrong data bit, can write the wrong parity bit, cannot read the actual parity bit. This combination set is for test, used to write the wrong data bit and the wrong parity bit for 32-bit SRAM and SDRAM. It is used to write the wrong parity bit and make data bit error through stb and sth instruction directly for 8-bit SRAM. This combination set neither detect error nor correct error. This combination set is usually unused.
- Combination 8（MECFG1_8）:When set EDAC is not enabled, and EWB is effective, and ERB set, for 32-bit SRAM and SDRAM can write the wrong data bit and the wrong parity bit, and read the actual data bit and the actual

parity bit, the parity bit is stored into TCB field. For 8-bit SRAM cannot write the wrong data bit ,can write the wrong parity bit, can read the actual data bit and the actual parity bit ,the parity bit is stored into TCB field. This combination set is for test, used to write the wrong data bit and the wrong parity bit, and read the actual data bit and the actual parity bit for 32-bit SRAM and SDRAM. It is used to write the wrong parity bit and read the actual data bit and the actual parity bit For 8-bit SRAM. This combination set neither detect error nor correct error.

The PROM EDAC function configuration:

- When reading 8-bit, 32-bit PROM and ERB is effective, the normal parity will be stored into TCB field in MECFG1.
- PROM's EDAC function is no related to EWB and ERB.
- PROM has no make error function.

Memory EDAC configuration register MECFG2(0x80000104)

Mainly used to make error control mode, the certain data bit reverse.

The following Table 6-4-7 presents Memory EDAC configuration register MECFG2.

Table 6-4-7 Memory EDAC configuration register MECFG2(0x80000104)

Bit	r/w	Name	Description
31:0	r/w	DR	Data Reversal, high-effective

Description: When EWB in MECFG1 set effective, for 32-bit data width SRAM and SDRAM, for the whole word write operation, the write data bit will reverse according to MECFG2.

Memory EDAC configuration register MECFG3(0x80000108)

The following Table 6-4-8 presents Memory EDAC configuration register MECFG3.

Table 6-4-8 Memory EDAC configuration register MECFG3(0x80000108)

Bit	r/w	Name	Description
31:7			Reserved.
6:0	r/w	PR	Parity Reversal, high-effective

Description: When EWB in MECFG1 set effective, for 8-bit,32-bit data width SRAM and SDRAM，for the whole word write operation，the write parity bit will reverse according to MECFG3.

SRAM and SDRAM error disposal

- If MECFG1 is configured to mecfg1_1(EDAC is enabled, that is, Sr1EEn, Sr2EEn, Sr3EEn, Sr4EEn, Sr5EEn and SdEEn in MECFG1 is set, and EWB is invalid and ERB is cleared), then
  - If one error occurs when reading memory (whether data bit or parity bit), then the memory controller will correct this data and write it back to the memory. It does not generate any trap or interrupt.
  - If two error occurs when reading memory, then

    While reading data, generate trap0x09;

    While fetching instruction, generate trap0x01.
  - When write byte or half word, one error occurs in the word that the byte or half word is in (whether data bit or parity bit), then the memory controller correct the error and modify the byte or half word which wanted to write in , then write the whole word into the memory. It does not generate any trap or interrupt.
  - When write byte or half word, two error occurs in the word that the byte or half word is in (whether data bit or parity bit), then generate trap0x2b.
- If MECFG1 is configured to mecfg1_3(Sr1EEn, Sr2EEn, Sr3EEn, Sr4EEn, Sr5EEn and SdEEn in MECFG1 is set, and EWB effective and ERB is cleared, and PR field in MECFG3 and MECFG2 are all 0, it will write the correct data bit and correct parity bit, parity bit is not stored into the TCB field in MECFG1 when reading, then
  - If one error occurs when reading memory(whether data bit or parity bit), then,

    While reading data, generate trap0x09;

    While fetching instruction, generate trap0x01.
  - If two error occurs when reading memory, then,

    While reading data, generate trap0x09;

    While fetching instruction, generate trap0x01.
  - For 32-bit sram, if write byte or half word, one error occurs in the word that the byte or half word is in (whether data bit or parity bit), then

    If rmw is set, generate trap0x2b;

    If rmw is clear, then do nothing else.

- For 8-bit sram and 32-bit sdram, if write byte or half word, one error occurs in the word that the byte or half word is in (whether data bit or parity bit), then,

  do nothing else, whatever rmw is set or not.
- For 32-bit sram, if write byte or half word, two error occurs in the word that the byte or half word is in (whether data bit or parity bit), then,

  If rmw is set, generate trap0x2b;

  If rmw is clear, do nothing else.
- For 8-bit sram and 32-bit sdram, if write byte or half word, two error occurs in the word that the byte or half word is in (whether data bit or parity bit), then,

  do nothing else, whatever rmw is set or not.

- If EDAC is disable, that means Sr1EEn, Sr2EEn, Sr3EEn, Sr4EEn, Sr5EEn and SdEEn in MECFG1 are clear, neither detect error nor correct error, does not generate any traps or interrupts, whatever ERB and EWB.
- If ERB is set, whatever EWB and EDAC, neither detects error nor corrects error and does not generate any traps or interrupts, and when reading memory, the parity bit will be stored into the TCB field of MECFG1.

8-bit, 32-bit PROM error disposal:

- If the prom EDAC is enable, that means the romEEn in MECFG1 is set, then when read the 8-bit, 32-bit prom,
  - If one error occurs (whether data bit or parity bit), then the memory controller will correct this data and transfer inside, but will not write it back to memory. It does not generate any trap or interrupt.
  - If two error occurs, then,

    While reading data, generate trap0x09;

    While fetching instruction, generate trap0x01.
- If the prom EDAC is disable, that means the romEEn in MECFG1 is clear, then when read the 8-bit, 32-bit prom, neither detect error or correct error,
  - If read prom data, do not generate any trap or interrupt.
  - If fetching prom instructions, one error or two errors occurs and become un-implemented instruction, then generate trap0x01.

Others

For 32-bit SRAM and SDRAM, data error and parity error can be made by word write but not byte write or half word write through MECFG2 and MECFG3.

For 8-bit SRAM, data error cannot be made through MECFG2 whatever byte write or half word write or word write, data error can be made by byte write or half word without MECFG2.

For 8-bit SRAM, parity error can be made by word write but not byte write or half word write through MECFG3.

For 8-bit and 32-bit SRAM, for EDAC write is enable, if write byte or half word, the writing is in the "rmw" mode whatever rmw is set or not.

For SDRAM, for EDAC write is enable, if write byte or half word, the writing is in the "rmw" mode.

32-bit PROM memory and its 8-bit parity PROM memory share the same signals including romsn, oen, written, address.

32-bit SRAM memory and its 8-bit parity SRAM memory share the same signals including ramsn, ramoen, rwen, address. Parity SRAM memory's write enable signal use rwen[0].

32-bit SDRAM memory and its 8-bit parity SDRAM memory share the same signals including sdcsn, sdcasn, sdrasn, sdwen, address. Parity SDRAM memory's DQM[0] signal use sddqm[3].

6.5 PCI Interface and Arbiter

6.5.1 Overview

BM3803FMGRH PCI interface implementation is compliant with the PCI 2.3 specification. It is interfaced to the processor core through the PCI to AHB bridge. Two different operating modes can be used with the PCI interface: Host Bridge or Guest Bridge. This module implements the standard 32-bit PCI interface.

PCI Host devices and Guest device are shown in Figure 6-5-1. In the left of Figure 6-5-1, BM3803FMGRH is in Host mode, that configurates other PCI devices on the bus and provides arbitration to other devices. In the right of figure 6-5-1, BM3803FMGRH is in Guest mode that configures by PCI Host device and

communicates under the management of the bus.

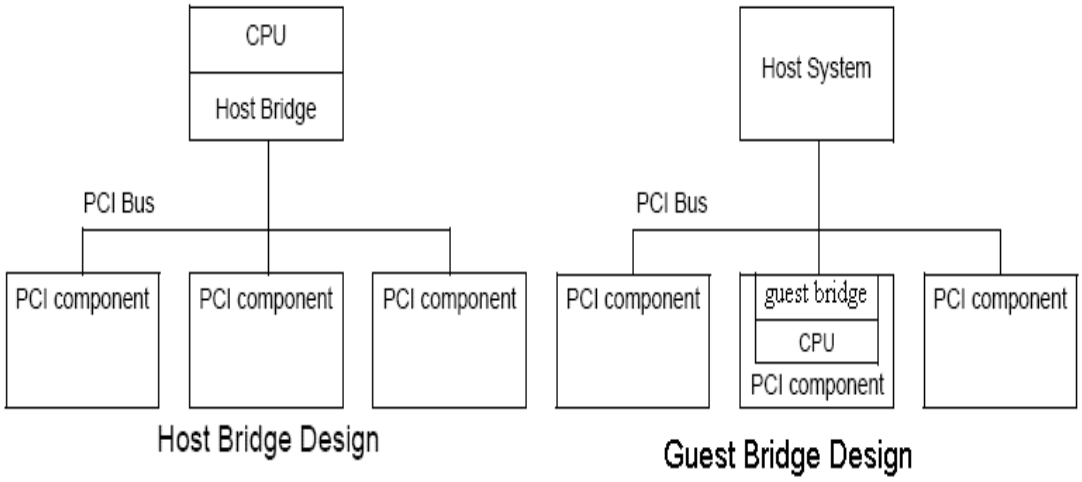


Figure 6-5-1 PCI bus devices

6.5.2 AHB-PCI module structure

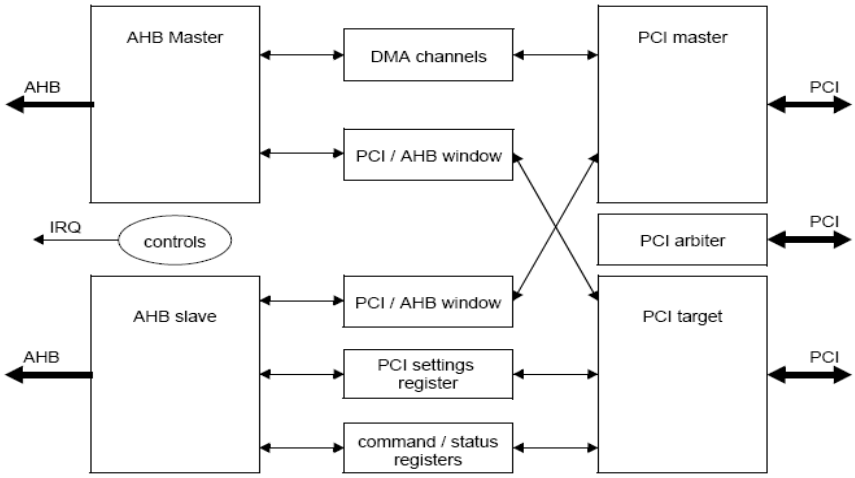


Figure 6-5-2 General architecture of the AHB/PCI Bridge

PCI interface is shown in Figure 6-5-2, that PCI master/target connect the AHB system to the PCI bus; BM3803FMGRH processor control and configure the AHB-PCI bridge module through the AHB slave module; BM3803FMGRH processor can access the PCI bus devices through the DMA channel. In Host mode, the AHB-PCI bridge contains an embedded PCI arbitration module which can manage 7 external PCI devices.

PCI-AHB window allow PCI bus masters to directly access CPU memory and

AHB devices.

AHB-PCI window allow AHB bus masters to read/write from PCI devices directly in both memory and I/O space.

6.5.3 PCI Interface Reset

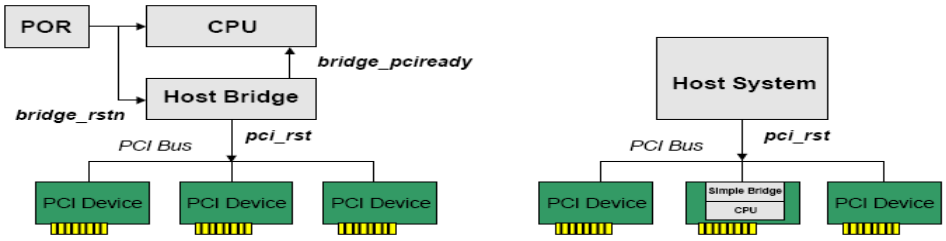


Figure 6-5-3 Reset Configurations

As shown in Figure 6-5-3, BM3803FMGRH PCI interface is in Host mode, while BM3803FMGRH is reset, it sent a PCI bus reset signal, and report whether PCI bus reset is complete. BM3803FMGRH write 0x05 to AHB address register 0x8000-00D0 to make the PCI bus reset.

In Guest mode of BM3803FMGRH PCI interface, PCI side is reset when receive the reset signal on the PCI bus, AHB side is reset when receive the reset signal of BM3803FMGRH.

6.5.4 PCI interface AHB side address space

The AHB Slave interface controls four separate memory spaces, as illustrated in Figure 6-5-4.

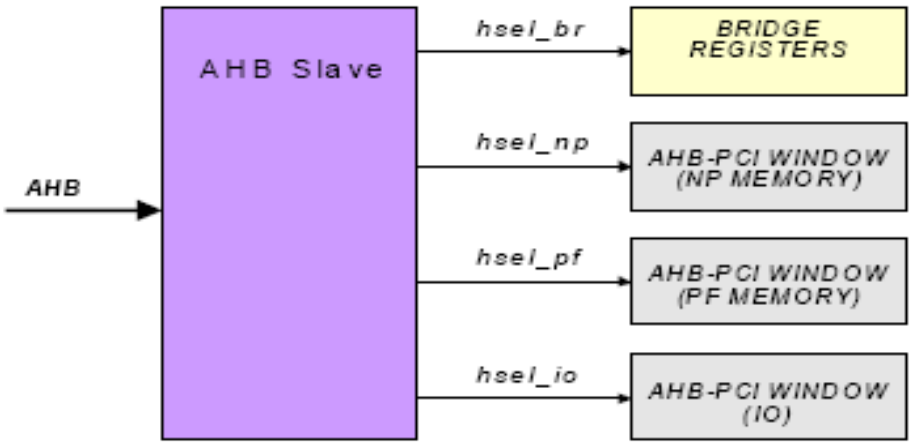


Figure 6-5-4 AHB target memory spaces

Space allocated as follows:

Register space (bridge registers): 0xC000-0000 - 0xC7FF-FFFF

AHB-PCI non-prefetchable address space (NP): 0xD000-0000 -0xDFFF-FFFF

AHB-PCI prefetchable address space (PF): 0xE000-0000 - 0xFFFF-FFFF

AHB-PCI I/O address space (I/O): 0xC800-0000 - 0xCFFF-FFFF

AHB side bridge register space, base address 0xC000-0000, as shown in Table 6-5-1

Table 6-5-1 AHB register mapping

Byte Offset	Mode	Name	Description
000h	R/W	WDMA_PCI_ADDR	Write DMA start address on the PCI bus
004h	R/W	WDMA_AHB_ADDR	Write DMA transfer start address on the AHB bus
008h	R/W	WDMA_CONTROL	Write DMA size & control
00Ch ... 01Ch	reserved		
020h	R/W	RDMA_PCI_ADDR	Read DMA start address on the PCI bus
024h	R/W	RDMA_AHB_ADDR	Read DMA transfer start address on the AHB bus
028h	R/W	RDMA_CONTROL	Read DMA size & control
02Ch ... 03Ch	reserved		
040h	R/W	CPU_IMASK	Interrupt mask
044h	R/W/C	CPU_ISTATUS	Interrupt status
048h	W	CPU_ICMD	Interrupt command
04Ch	R	CPU_VERSION	Bridge version and miscellaneous information
050h ... 06Ch	reserved		
070h	R/W	PCIAHB_ADDR_NP	PCI-AHB window non-prefetchable range control
074h	R/W	PCIAHB_ADDR_PF	PCI-AHB window prefetchable range control
078h	R/W	PCIAHB_TIMER	PCI-AHB window discard timer
07Ch	R/W	AHBPCI_TIMER	AHB-PCI window discard timer
080h	R/W/C	PCI_CONTROL	PCI control bits
084h	R or R/W	PCI_DV	PCI device and vendor ID
088h	R or R/W	PCI_SUB	PCI subsystem device and vendor ID
08Ch	R or R/W	PCI_CREV	PCI class code and revision ID
090h	R/W/C	PCI_BROKEN	PCI arbiter broken master register
094h	R or R/W	PCIAHB_SIZ_NP	PCI-AHB window non-prefetchable range size
098h	R or R/W	PCIAHB_SIZ_PF	PCI-AHB window prefetchable range size
09Ch ... 3FCh	reserved		

6.5.5 PCI arbiter

PCI arbitrator module is effective in PCI Host mode. The arbiter can provide arbitration for the bridge itself and up to 7 external PCI devices. This module is used to:

- Enable PCI bus Master device to start the DMA transfer;
- Avoid PCI bus conflict;
- Ensure that all PCI bus masters have an equal chance to transfer data regularly.

The Arbiter detects and black-lists malfunctioning masters so that they will not impact system performance. A master device is considered malfunctioning if it requests the bus for 16 PCI clock cycles or more without starting a transaction.

The first 8 bits of the PCI_BROKEN register indicate the status of the device. A bit set to 1 indicates that the corresponding master is black-listed and not granted access to the bus.

6.5.6 PCI side register space

In Guest mode, BM3803FMGRH PCI interface implements PCI standard configuration space, in addition, PCI side register also includes the standard PCI interrupted bridge registers and the PCI version registers (these registers between address 0x80-0x8c). PCI side register address space is shown in Table6-5-2

Table 6-5-2 PCI configure register address space

Offset	Mode	Name	Description
00h…3Ch	-	-	Standard PCI Header
40h…7Ch	-	-	reserved
80h	R/W	PCI_IMASK	Interrupt mask
84h	R/W/C	PCI_ISTATUS	Interrupt status
88h	W	PCI_ICMD	Interrupt command
8Ch	R	PCI_VERSION	Bridge version and miscellaneous information
90h…FCh	-	-	reserved

PCI interface in Guest mode implements the standard PCI configuration registers stated by PCI2.3 standard. It include Six PCI address space, the bar3-bar5 space is

defined, and bar0-bar2 space is reserved, as shown in Table6-5-3

Table 6-5-3 PCI Target address space

Space	Size	Resource
BAR0...BAR2		reserved
BAR3	user defined	PCI-AHB window non-prefetchable area
BAR4	user defined	PCI-AHB window prefetchable area
BAR5	256 bytes	same as configuration space (Guest Bridge only)

When BM3803FMGRH PCI interface is in Host mode, the PCI side register does not exist, but can configure other PCI bus devices through configuring read and write.

PCI Host device assign address space for BM3803FMGRH PCI Guest device source(include Memory and I/O) in PCI bus, that needs to configure three registers BAR3, BAR4, BAR5; BAR5 configures PCI bus base-address space, which would allow BM3803FMGRH PCI Guest to access other device configuration registers by read/write memory but not read/write configuration, such as read/write interrupt register(see the relevant sections of the interrupt), or reconfiguring the BAR3 or/and BAR4 of the configuration space register in order to implement the address space redistribution.

The use of interrupt related registers see 6.5.9.

6.5.7 Address Space Mapping

AHB-PCI address space mapping

AHB-PCI address space mapping translates AHB bus read/write operation into PCI bus read/write operation.

AHB bus Master device accesses the PCI device by AHB-PCI address space mapping. AHB-PCI address space mapping captures the AHB operation pointing to the PCI interface AHB space, and transfers these operations to the PCI bus. These operations do not need other modules such as IU.

AHB-PCI address space mapping includes non-prefetch memory mapping space, prefetch memory mapping space, prefetch I/O mapping space, AHB side address range is inside of the PCI interface AHB side address range.

Figure 6-5-5 shows AHB-PCI address space mapping.

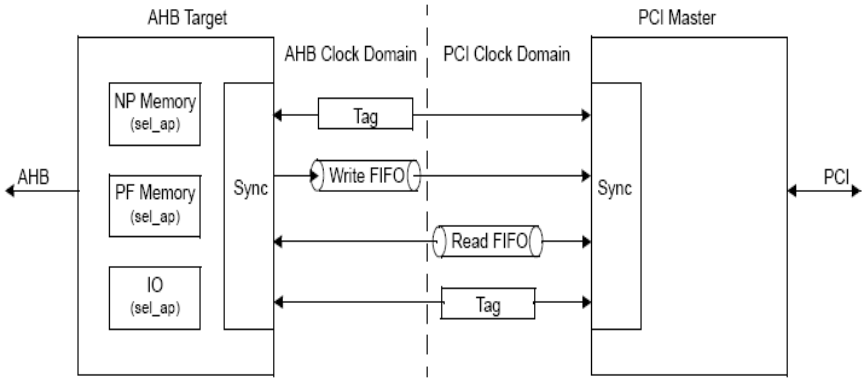


Figure 6-5-51AHB-PCI address space mapping

AHB-PCI address space mapping uses a flat addressing, the AHB bus address is the same as the translated PCI bus address.

When BM3803FMGRH access AHB address space 0xC800-0000 - 0xFFFF-FFFF, PCI bus operation is started automatically, that is PCI interface access PCI address space 0xC800-0000 - 0xFFFF-FFFF:

- 0xC800-0000 - 0xCFFF-FFFF map AHB-PCI I/O address space; then perform PCI I/O access.
- 0xE000-0000 - 0xFFFF-FFFF map AHB-PCI prefetch address space; then perform PCI prefetch memory access.
- 0xD000-0000 - 0xDFFF-FFFF map AHB-PCI non-prefetch address space; then perform PCI non-prefetch memory access.

BM3803FMGRH based on address space mapping ， access the PCI bus address of 0xC800-0000 - 0xFFFF-FFFF by access AHB bus address directly.(Include prefetch address space mapping and non-prefetch address space mapping and I/O address space mapping), and the other PCI address space could be accessed by the DMA system of PCI interface.

PCI-AHB address space mapping

PCI-AHB address space mapping translates PCI bus read/write operation into AHB bus read/write operation.

The other Master devices on PCI bus access BM3803FMGRH local memory and AHB bus device directly by PCI-AHB address space mapping. PCI-AHB address space mapping captures the PCI operation pointing to the PCI space, and transfer these operations to the AHB bus. Similar to the AHB-PCI address space mapping, these operations do not need other modules such as IU.

PCI-AHB address space mapping includes three parts: Bar3, Bar4, Bar5, that is non-prefetch PCI memory mapping space, prefetch PCI memory mapping space and PCI interface PCI side register space. The address translate system is different in different PCI interface modes.

Figure 6-5-6 shows the PCI-AHB Window structure.

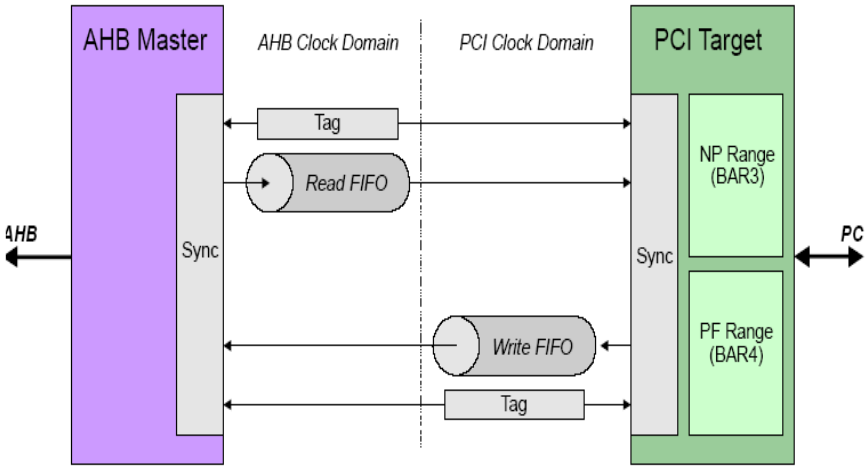


Figure 6-5-6 PCI-AHB Window structure

PCI-AHB address space mapping includes three parts: Bar3, Bar4, Bar5:

- The Non-prefetchable address range is mapped in BAR3. No data prefetch is performed when serving PCI transactions targeting this address range.
- The prefetchable address range is mapped in BAR4. Data prefetch is automatically performed when serving transactions targeting this address range.
- PCI interface PCI side register space is mapped to the PCI device BAR5. Then accessing memory can also modify the PCI interface configuration register.

Each address space of Bar3 and Bar4 can range from 256 bytes to 1GB. The AHB side address is defined in the AHB side register. The AHB base address is defined by PCIAHB_ADDR_NP and PCIAHB_ADDR_PF respectively, and the size is defined by PCIAHB_SIZ_NP and PCIAHB_SIZ_PF.

In Host mode, PCI-AHB address space mapping do not translate the address, the PCI bus accessing address is the same as the AHB bus target address translated. Conversely in Host mode the PCI side address range of address space mapping is defined by AHB side register.

In Guest mode, the PCI side address range of address space mapping is defined by BAR3 base address, BAR4 base address, BAR5 base address(PCI spec2.3). The relation between PCI space and AHB space is:

Non-prefetchable range:

AHB address = PCIAHB_ADDR_NP + (PCI address - BAR3 base address);

Prefetchable range:

AHB address = PCIAHB_ADDR_PF + (PCI address - BAR4 base address).

6.5.8 DMA

PCI interface is usually accessed by DMA, DMA channel as shown in Figure 6-5-7, DMA controller control the PCI bus, as well as the AHB bus and transfer data without the CPU:

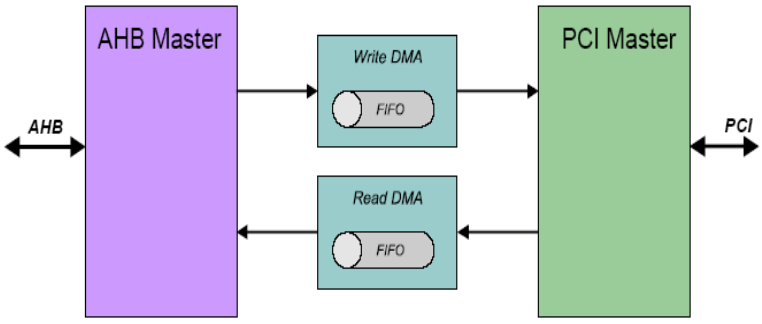


Figure 6-5-72 DMA data path

DMA can transfer Memory or I/O data between AHB bus and PCI bus, and perform configuration, rarely need CPU.

In order to store data, the CPU must first initialize and allocate a memory buffer that can be read from or written to. The CPU then initializes the DMA registers and specifies a PCI transfer type and target address. No CPU action is necessary once a transaction is initiated as the AHB master and PCI master perform all required data transfers automatically (Unless the user terminates the DMA transfer). DMA transfer status is reported in the CPU_ISTATUS register and can be signaled using a processor interrupt.

In DMA operation, DMA read means read data from PCI address space to BM3803FMGRH's AHB address space; DMA write means write data from BM3803FMGRH's AHB address space to PCI address space.

Initiate DMA operation needs three registers: PCI address register, AHB address register and DMA control register, while DMA read register and DMA write register have different address. DMA write control register (address: 0xC000-0008), DMA read control register (address: 0xC000-0028), and the AHB address (WDMA_AHB_ADDR, RDMA_AHB_ADDR) will increase automatically until the end of DMA.

DMA control register DMA_CONTROL (include read and write) defines the PCI command and control operation used to transfer data on PCI bus. Figure 6-5-8 shows the DMA control register:

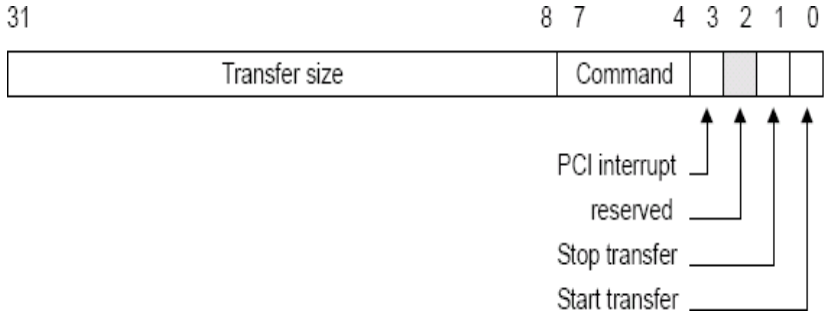


Figure 6-5-8 DMA control register

- Bit[0]: Assertion of this bit initiates a DMA transfer. This bit is automatically cleared when a transfer is completed.
- Bit[1]: This bit is used to abort a transfer in progress. You should only consider this as a last-resort;
- Bit[3]: If this bit is set then an interrupt is issued on the PCI bus when a transfer terminates. No interrupt is generated if a transaction is aborted.

Transfer size is the transfer byte length of DMA, the maximum size is 16MB, the value is 0x1-0xFFFFFC;(16MB should be 24bit)

Command part specifies the DMA type. The DMA commands are shown in Table 6-5-4:

Table 6-5-4 DMA commands

Channel	Command	Description
Read DMA	0000	Interrupt Acknowledge
	0010	I/O Read
	0110	Memory Read
	1010	Configuration Read

	1100	Memory Read Multiple
Write DMA	0001	Special Cycle
	0011	I/O Write
	0111	Memory Write
	1011	Configuration Write

For example, the code

*(unsigned int *)0xc0000000 = 0xc0000000; // Write DMA start address on the PCI bus

*(unsigned int *)0xc0000004 = 0x40000000; // Write DMA start address on the AHB bus

*(unsigned int *)0xc0000008 = 0x10071; // Write DMA size & control

write 0x100 bytes data from AHB address 0x4000-0000 into PCI address 0xC000-0000.

Code

*(unsigned int *)0xc0000020 = 0x40000000;

*(unsigned int *)0xc0000024 = 0x40001000;

*(unsigned int *)0xc0000028 = 0x100a1;

read 0x100 bytes data from 0x4000-0000 PCI device configuration space.

When BM3803FMGRH PCI Host device configures PCI through configuring DMA, single DMA read/write should be used as can as possible. When configuring DMA read/write the PCI address is related to the position of PCI bus motherboard.

The format of PCI bus configuration transactions type 0 and type 1 is shown in Figure 6-5-9:

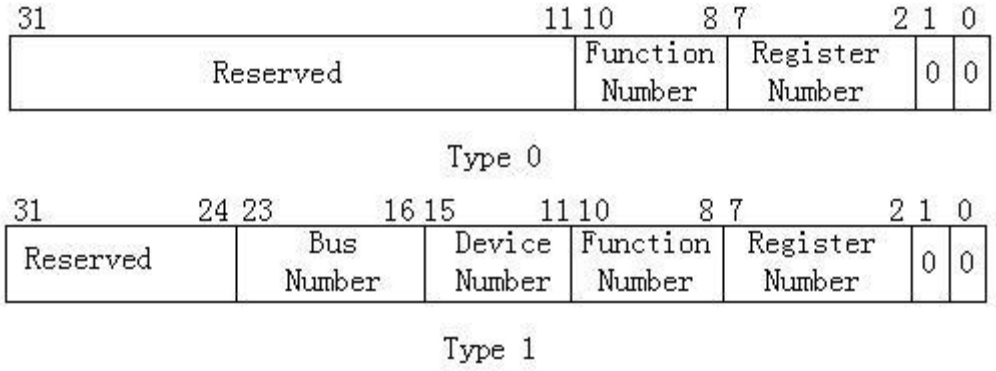


Figure 6-5-93 Type 0 and Type 1 configuration addresses

BM3803FMGRH AHB bus is a big-endian bus, while PCI bus is a little-endian

bus, so the DMA transfer between them should be according to certain request to ensure the data order identical.

When BM3803FMGRH destination or source address is 32 bit data length RAM space and the PCI devices that used for DMA transfer are two BM3803FMGRH PCI devices, the following principles should be followed in order to avoid bytes dislocation:

- When the two low significant bits of PCI address and AHB address are same, any type byte length DMA can be performed;
- When PCI address and AHB address 2bytes(half word) are aligned, the length of DMA is 2 or a multiple of 2;
- When PCI address and AHB address 4bytes(word) are aligned, the length of DMA is 4 or a multiple of 4.

When 3803 DMA destination or source address is 8 bit data length RAM space, then PCI address and AHB address need to 4bytes (word) aligned, the length of DMA is 4 or a multiple of 4.

PCI address and AHB address 4bytes (word) aligned is recommended, the length of DMA is 4 or a multiple of 4. In this case no matter what situations will not be influenced by the big or little end problem.

6.5.9 PCI Module Interrupt

PCI Module Interrupt either can send interrupt request to the IU module of BM3803FMGRH through AHB bus, or can send interrupt (Guest mode) through PCI bus or response the interrupt request (Host mode) from PCI bus. The former called Processor interrupt, the latter called PCI interrupt.

Processor Interrupt

Processor interrupt of PCI interface is the 14th interrupt of BM3803FMGRH processor. The interrupts are some different between Host and Guest mode. AHB side has three registers related to interrupt:

- 0xC000-0040 :CPU_IMASK CPU Interrupt Mask Register
- 0xC000-0044 :CPU_ISTATUS CPU Interrupt Status Register

- 0xC000-0048 :CPU_ICMD CPU Interrupt Command Register

Figure 6-5-10 shows the CPU Interrupt Register.

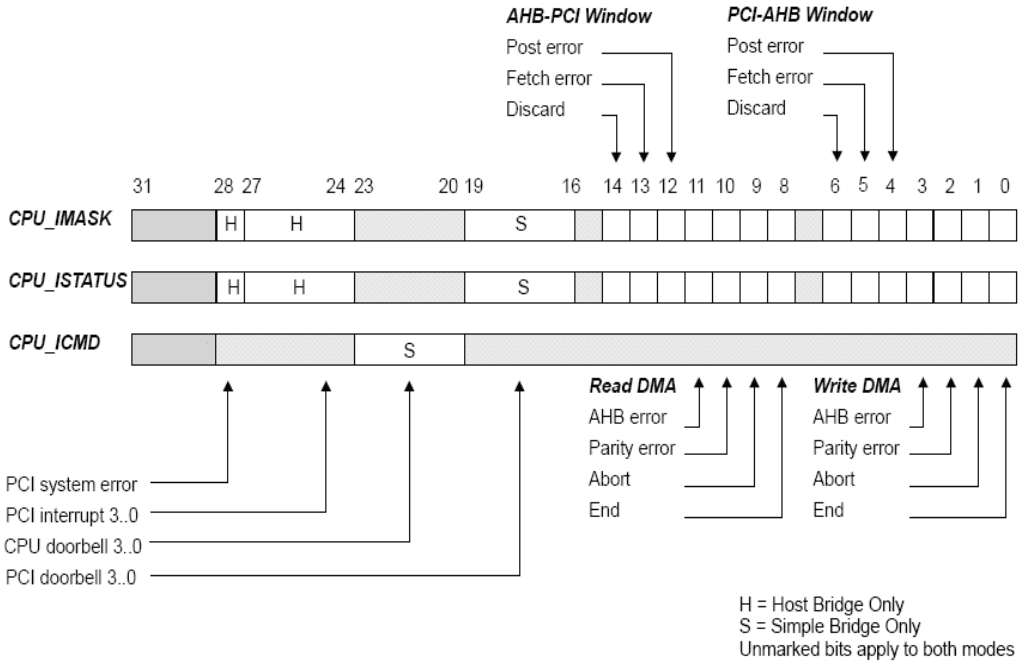


Figure 6-5-10 CPU Interrupt Register

- CPU_ISTATUS: This register is a read/write/clear register. Status register bits are automatically set when the corresponding interrupt source is activated. Each source is independent and thus multiple sources may be active simultaneously. The CPU monitors and clears status bits; 1 clears a bit, 0 has no effect.
- CPU_IMASK: This register is a read/write register. Setting a bit enables the associated interrupt source and clearing a bit masks the interrupt source. These settings do not affect the CPU_ISTATUS register: i.e., a masked interrupt is still normally reported to the corresponding status bit. If one or more interrupt sources are active and not masked the Bridge asserts a CPU interrupt line.
- CPU_ICMD: This register is used to manually activate some interrupt sources. This register is write-only and reading it always returns 0. Writing a 1 to a bit set it, writing a 0 has no effect.

PCI Interrupt

In PCI side configuration space there are three interrupt registers, PCI_IMASK,

PCI_ISTATUST, PCI_ICMD, that is PCI interrupt mask register, PCI interrupt status register, and PCI interrupt command register. The CPU Doorbell bit and PCI Doorbell bit of the three registers could be configured considering with the corresponding bits of the CPU interrupt registers to generate interrupt request (effective only in Guest mode). The three registers can only be read/write by PCI bus, as shown in Figure 6-5-114.

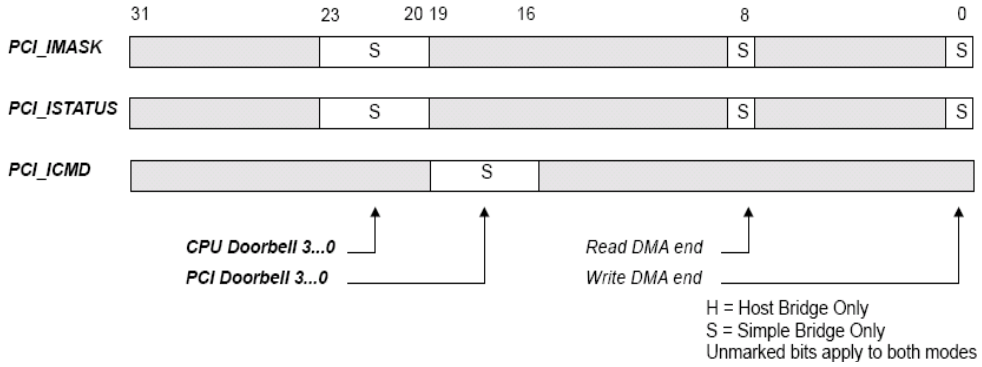


Figure 6-5-115 PCI Interrupt Register

- PCI Doorbell: PCI Doorbell bits are accessible for PCI devices that send an interrupt to the CPU. Setting a PCI Doorbell bit generates an interrupt to the processor if the interrupt source is enabled in the CPU_IMASK register. Doorbell bits are write-only. Writing a 0 to these bits has no effect (Guest Bridge only).
- CPU Doorbell: CPU Doorbell interrupt sources can be triggered by the embedded processor. This allows the processor to send an interrupt to the PCI bus INTA# line (if the interrupt source is enabled in the PCI_IMASK register) for any user-defined purpose. (Guest Bridge only).
- Read DMA end and Write DMA end is similar with the CPU Doorbell, that send interrupt signals to PCI bus Host device through PCI bus INTA#, the interrupt is triggered by the DMA.
- PCI interrupt is level sensitive. After handling the PCI interrupt, it needs to be cleared. At first the corresponding bit of CPU_ISTATUS register need to be written to 1, then clear the 14th cpu interrupt.

6.6 GPIO, UART and Timer

6.6.1 GPIO

The general purpose input output (GPIO) consists in a 32-bit wide I/O port.

PIO[15:0] are able to be configured as input or output respectively.

PIO[31:16] are accessible through D[15:0].

PIO[31:16] can only be used when all memory areas (ROM, RAM and I/O) are 8-bit or 16-bit wide. If the SDRAM controller is enabled, PIO[31:16] cannot be used.

PIO[31:16] can only be configured as outputs or inputs on byte.

GPIO data can be accessed by the register 0x8000-00A0, GPIO direction can be configured by the register 0x8000-00A4.

PIO[15:0] have other alternate functions，is shown in Table 6-6-1:

Table 6-6-1 other functions

I/O	Function	Type	Description
PIO [15]	TXD1	Out	UART1 transmitter data
PIO [14]	RXD1	In	UART1 receiver data
PIO [13]	RTS1	Out	UART1 request-to-send
PIO [12]	CTS1	In	UART1 clear-to-send
PIO [11]	TXD2	Out	UART2 transmitter data
PIO [10]	RXD2	In	UART2 receiver data
PIO [9]	RTS2	Out	UART2 request-to-send
PIO [8]	CTS2	In	UART2 clear-to-send
PIO [3]	UART CLOCK	In	UART clock
PIO [2]	PROM EDAC	In	Enable PROM EDAC at reset
PIO [1:0]	PROM width	In	Define PROM width at reset

GPIO can be used as interrupt input also.GPIO interrupt can be configured by register 0x8000-00A8.

The interrupt configuration register has 4 groups, each group has 8 bits: the enable bit EN, edge/level triggered mode(LE), polar(PL), and one of 32 bit I/O port selected as an interrupt input.

6.6.2 UART

Three UARTs are implemented. Two Uarts have other alternate functions as general purpose input output (GPIO).

Each UART consists of a transmitter holding register, a receiver holding register, a transmitter shift register, and a receiver shift register.

Each UART is controlled by four registers including UART control register, UART status register, UART scalar reload register and UART data register. Serial data's send and received through the operation of the above registers.

Flow control /Loop back mode

If flow control is enabled, the transmitter starts to transmit data when the CTSN input of transmitter is low. The transmitter stops transmitting data after finishing current frame when the CTSN input of transmitter is high.

If flow control is enabled, the RTSN of receiver will be high when it received valid start bit. When the holding register is read, the RTSN will automatically be low.

If the LB bit in the UART control register is set, the UART will be in loop back mode. In loop back mode, the transmitter output and the receiver input is internally connected and the RTS is connected to the CTS. It is used for tests.

Baud Rate

Each UART contains a 12-bit down-counting scalar to generate the desired baud-rate. The scalar is clocked by the system clock when the EC bit of UART control register is low. The scalar is clocked by PIO[3] when the EC bit of UART control register is high. the frequency of PIO[3] must be less than half the frequency of the system clock.

The scalar value is calculated by the following equation:

scalar =((uartclk *10)/(baudrate *8) - 5)/10

Interrupt

An UART interrupt will be generated when the transmitter is enabled, the transmitter interrupt is enabled，when flow control is disabled and the transmitter holding register moves from full to empty.

An UART interrupt will be generated when the receiver is enabled, the receiver

interrupt is enabled and the receiver holding register moves from empty to full.

6.6.3 Timer Unit and watchdog

There are two 24-bit timers, one 24-bit watchdog and they share one 10-bit prescalar.

The prescalar is clocked by the system clock. The prescalar is decremented on each clock cycle. When the prescalar underflows, it generate a tick and is automatically reloaded with the prescalar reload register.

The timer or watchdog value is decremented at each prescalar's tick.When timer or watchdog underflows, it will generate an interrupt.

The effective division rate is equal to prescalar reload register value + 1.

Timer and watchdog can be enabled/disabled by setting the enable bit (en) in the control register.

6.7 Debug Unit-DU

6.7.1 Overview

There is a debug unit(DU) in the processor.

The DU access to all processor registers and cache memories in debug mode. There is a 512*16_byte trace buffer which stores executed instructions or data transfers on the internal bus. The DU communications through a standard UART.

6.7.2 Debug Unit

The debug unit is used to control the trace buffer and the processor debug mode. The DU master occupies an address space on the internal bus. Through this address space, any other masters can access the processor registers and the contents of the trace buffer.

The DU control registers can be accessed at any time, while the processor registers and caches can only be accessed when the processor has entered debug mode. The trace buffer can be accessed only when tracing is disabled or completed. In debug

mode, the processor pipeline is held and the processor is controlled by the DU.

Entering the debug mode can occur on the following events:

- executing a breakpoint instruction (ta 1);
- integer unit hardware breakpoint/watchpoint hit (trap 0x0B);
- rising edge of the external break signal (DUBRE);
- setting the break-now (BN) bit in the DU control register;
- a trap that would cause the processor to enter error mode;
- occurrence of any, or a selection of traps as defined in the DU control register;
- after a single-step operation;
- DU breakpoint hit.

The debug mode can only be entered when the debug unit is enabled through an external pin (DUEN). Driving the DUEN pin high enables the debug mode. When the debug mode is entered, the following actions are taken:

- PC and nPC are saved in temporary registers (accessible by the debug unit);
- an output signal (DUACT) is asserted to indicate the debug state;
- the timer unit is stopped to freeze the timers and watchdog.

The instruction that caused the processor to enter debug mode is not executed, and the processor state is kept unmodified. Execution is resumed by clearing the BN bit in the DU control register or by de-asserting DUEN. The timer unit will be re-enabled and execution will continue from the saved PC and nPC. Debug mode can also be entered after the processor has entered error mode, for instance when an application has terminated and halted the processor. The error mode can be reset and the processor restarted at any address.

Time Tag

There is a time tag counter in the DU. This counter is decremented each clock as long as the processor is running. The counter is stopped when the processor enters debug mode. This time tag counter is stored in the trace as an execution time reference. It is restarted when execution is resumed.

Trace Buffer

The trace buffer consists of a circular buffer that stores the executed instructions or the internal bus data transfers. The size of the trace buffer is 512 lines of 16 bytes. The trace buffer operation is controlled through the DU control register (DUC) and the trace buffer control register (TBC). When the processor enters debug mode,

tracing is suspended.

The trace buffer can contain three modes:

- Instruction trace mode(record instruction only)
- Bus trace mode(record transferred data only)
- Mixed trace mode(record instruction and data)

The trace buffer control register (TBC) contains two counters (BCNT and ICNT) that store the address of the trace buffer location that will be written on next trace. Since the buffer is circular, it actually points to the oldest entry in the buffer. The indexes are automatically incremented after each stored trace entry.

Instruction trace

The instruction trace mode is enabled by setting logical one to the trace instruction enable bit (TI) in the trace buffer control register (TBC).

During instruction tracing, one instruction is stored per line in the trace buffer with the exception of multi-cycle instructions. Multi-cycle instructions can be entered two or three times in the trace buffer:

- For store instructions, bits [95:64] correspond to the store address on the first entry and to the stored data on the second entry (and third in case of STD: bits [95:64] correspond to the store address on the first entry and to the stored high 32-bit data on the second entry and stored low 32-bit data on the third entry. Bit 126 is set logical one on the second and third entry to indicate this.
- A double load (LDD) is entered twice in the trace buffer, with bits [95:64] containing stored high 32 bit data, the second entry stored low 32 bit the loaded data.
- Multiply and divide instructions are entered twice, but only the last entry contains the result. For multiply instruction, bits [95:64] correspond to the stored high 32-bit data on the first entry, bits [95:64] correspond to the stored low 32-bit data on the second entry; For divide instruction, bits [95:64] correspond to the stored expanded symbols on the first entry, bits [95:64] correspond to the stored operation results on the second entry. Bit 126 is set for the second entry.
- For FPU operation producing a double-precision result, the first entry puts the MSB 32 bits of the results in bit [95:64] while the second entry puts the LSB 32 bits in this field.

The Instruction trace record format is shown in Figure 6-7-1 and Table 6-7-1:

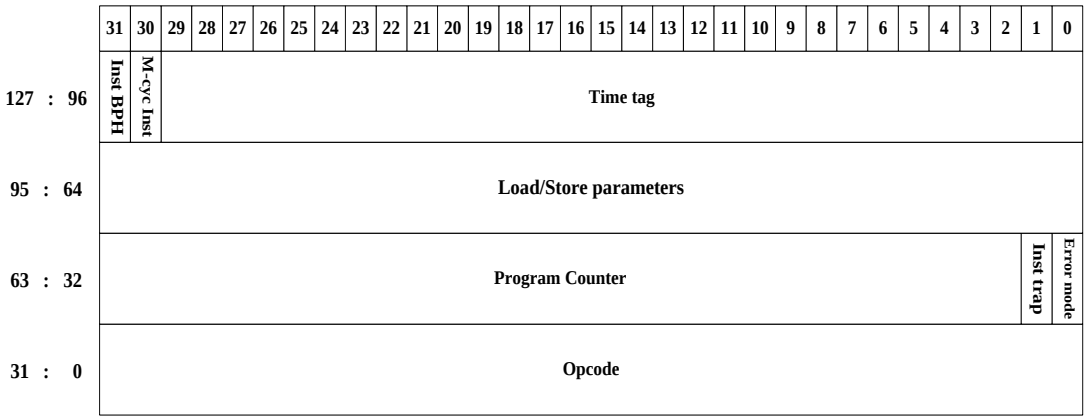


Figure 6-7-1 Trace buffer data allocation, Instruction tracing mode

Table 6-7-1 Trace buffer data allocation, Instruction tracing mode

Bits	Name	Definition
127	Instruction breakpoint hit	Set to '1' if a DU instruction breakpoint hit occurred.
126	Multi-cycle instruction	Set to '1' on the second and third instance of a multi-cycle instruction (LDD, ST or FPOP)
125:96	DU counter	The value of the DU counter
95:64	Load/Store parameters	Instruction result, Store address or Store data
63:34	Program counter	Program counter (2 lsb bits removed since they are always zero)
33	Instruction trap	Set to '1' if traced instruction trapped
32	Processor error mode	Set to '1' if the traced instruction caused processor error mode
31:0	Opcode	Instruction opcode

When a trace is frozen, interrupt 11 is generated.

Bus Trace

The bus trace mode is enabled by setting logical one to the trace AHB enable bit (TA) in the trace buffer control register (TBC).

During bus tracing, one operation of the internal bus is stored per line in the trace buffer. The record as follows:

The Bus trace record format is shown in Figure 6-7-2 and Table 6-7-2:

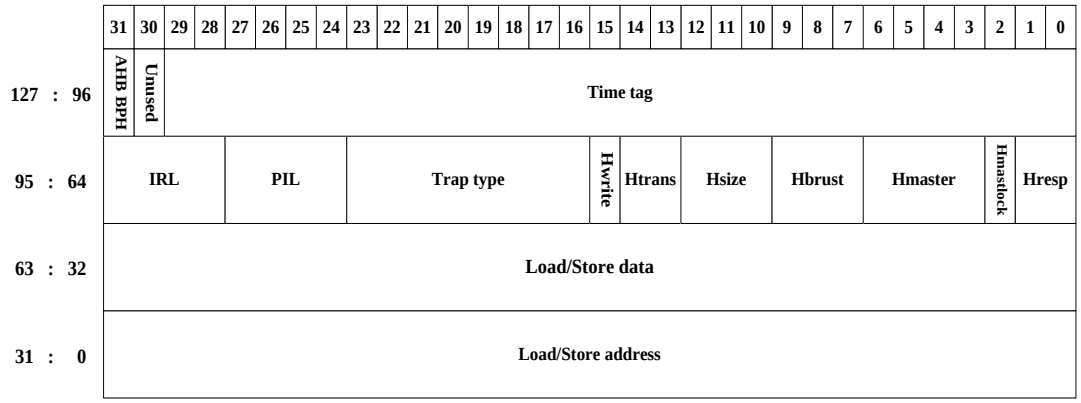


Figure 6-7-21 Trace Buffer Data Allocation, Internal bus Tracing Mode

Table 6-7-22 Trace Buffer Data Allocation, Internal bus Tracing Mode

Bits	Name	Definition
127	AHB breakpoint hit	Set to '1' if a DU AHB breakpoint hit occurred.
126	-	Unused
125:96	DU counter	The value of the DU counter
95:92	IRL	Processor interrupt request input
91:88	PIL	Processor interrupt level (psr.pil)
95:80	trap type	Processor trap type (psr.tt)
79	Hwrite	AHB HWRITE
78:77	Htrans	AHB HTRANS
76:74	Hsize	AHB HSIZE
73:71	Hburst	AHB HBURST
70:67	Hmaster	AHB HMASTER
66	Hmastlock	AHB HMASTLOCK
65:64	Hresp	AHB HRESP
63:32	Load/Store data	AHB HRDATA or HWDATA
31:0	Load/Store address	AHB HADDR

Mixed Trace

In mixed mode, the buffer is divided on two halves, with instructions stored in the lower half and bus transfers in the upper half. The MSB bit of the AHB index counter is then automatically kept high, while the MSB of the instruction index counter is kept low, the initial address of instruction trace is 0x9001-0000, the initial address of AHB trace is 0x9001-1000.

DU Memory Map

Table 6-7-3 shows DU memory map.

Table 6-7-3 DU Map

Address	Register
0x8000-00c4	DU UART status register
0x8000-00c8	DU UART control register
0x8000-00cc	DU UART scalar register
0x9000-0000	DU control register
0x9000-0004	Trace buffer control register
0x9000-0008	Time tag counter
0x9000-0010	AHB break address 1
0x9000-0014	AHB mask 1
0x9000-0018	AHB break address 2
0x9000-001C	AHB mask 2
0x9001-0000 - 0x9001-1FFF	Trace buffer
0x9002-0000 - 0x9002-029F	IU/FPU register file
0x9008-0000	Y register
0x9008-0004	PSR register
0x9008-0008	WIM register
0x9008-000C	TBR register
0x9008-0010	PC register
0x9008-0014	NPC register
0x9008-0018	FSR register
0x9008-001C	DU trap register
0x9008-0060	ASR24
0x9008-0064	ASR25
0x9008-0068	ASR26
0x9008-006C	ASR27
0x9008-0070	ASR28
0x9008-0074	ASR29
0x9008-0078	ASR30
0x9008-007C	ASR31
0x9010-0000 - 0x9010-7FFF	Instruction cache tags
0x9010-0000 - 0x9010-1FFF	Instruction cache tags 1
0x9010-2000 - 0x9010-3FFF	Instruction cache tags 2
0x9010-4000 - 0x9010-5FFF	Instruction cache tags 3
0x9010-6000 - 0x9010-7FFF	Instruction cache tags 4
0x9014-0000 - 0x9014-7FFF	Instruction cache data
0x9014-0000 - 0x9014-1FFF	Instruction cache data 1
0x9014-2000 - 0x9014-3FFF	Instruction cache data 2
0x9014-4000 - 0x9014-5FFF	Instruction cache data 3
0x9014-6000 - 0x9014-7FFF	Instruction cache data 4

0x9018-0000 - 0x9018-3FFF	Data cache tags
0x9018-0000 - 0x9018-1FFF	Data cache tags 1
0x9018-2000 - 0x9018-3FFF	Data cache tags 2
0x901C-0000 - 0x901C-3FFF	Data cache data
0x901C-0000 - 0x901C-1FFF	Data cache data 1
0x901C-2000 - 0x901C-3FFF	Data cache data 2

DU Breakpoint

The DU contains two breakpoint registers for matching either internal bus addresses or executed processor instructions. A breakpoint hit is typically used to freeze the trace buffer, but can also put the processor in debug mode.

Freeze operation can be delayed by programming the DCNT field in the DU control register to a non-zero value. In this case, the DCNT value will be decremented for each additional trace until it reaches zero, after which the trace buffer is frozen. If the brake on trace freeze bit (BT) is set logical one in the DU control register, the DU forces the processor into debug mode when the trace buffer is frozen.

Note: Due to pipeline delays, up to 4 additional instructions can be executed before the processor is placed in debug mode.

A mask register is associated with each breakpoint, allowing breaking on a block of addresses. Only address bits with the corresponding mask bit set to '1' are compared during breakpoint detection.

6.7.3 DU Communication

The DU communicate to the internal bus as a master by a UART.

The DU communication link command consists of a control byte, a 32-bit address and optional write data. If the LR bit in the DU control register is set, a response byte will be sent after each AHB transfer. If the LR bit is not set, a write access does not return any response, while a read access only returns the read data.

DU Commands

Through the communication link, a read or write transfer can be generated to any address on the internal bus. A response byte is can optionally be sent.

Block transfers can be performed be setting the length field to n-1, where n denotes the number of transferred words. For write accesses, the control byte and address is sent once, followed by the number of data words to be written. The address

is automatically incremented after each data word. For read accesses, the control byte and address is sent once and the corresponding number of data words is returned.

Clock Generation

The DU UART contains a down-counting scalar to generate the desired baud-rate. The scalar is clocked by the system clock. When the scalar underflows, it generate a tick and is automatically reloaded with the scalar reload register. The resulting UART tick frequency should be 8 times the desired baud-rate.

If not programmed by software, the baud rate will be automatically discovered. If the BL bit is reset by software, the baud rate discovery process is restarted. The baud-rate discovery is also restarted when a 'break' is received by the receiver, allowing change to baud-rate from the external transmitter. For proper baud-rate detection, the value 0x55 should be transmitted to the receiver after reset or after sending break.

The best scalar value for manually programming the baud-rate can be calculated as follows:

$$\text{scaler} = \frac{\frac{\text{sysclk} \times 10}{\text{baudrate} \times 8} - 5}{10}$$

6.7.4 Booting from DU

By asserting DUEN and DUBRE at reset time, the processor will directly enter debug mode without executing any instructions. The system can then be initialized from the communication link, and applications can be downloaded and debugged. Additionally, external (flash) PROMs for standalone booting can be re-programmed.

6.8 On-chip registers

6.8.1 On-chip registers

Table 6-8-1 shows the On-chip registers.

Table 6-8-1 On-chip registers

Address	Register	Address	Register
0x8000-0000	Memory configuration	0x8000-00A0	I/O port input/output register

	register 1		
0x8000-0004	Memory configuration register 2	0x8000-00A4	I/O port direction register
0x8000-0008	Memory configuration register 3	0x8000-00A8	I/O port interrupt config. register
0x8000-000C	AHB Failing address register		
0x8000-0010	AHB status register		
0x8000-0014	Cache control register		
0x8000-0018	Power-down register		
0x8000-001C	Write protection register 1		
0x8000-0020	Write protection register 2	0x8000-00C4	DU UART status register
0x8000-0024	Configuration register	0x8000-00C8	DU UART control register
0x8000-0040	Timer 1 counter register	0x8000-00CC	DU UART scalar register
0x8000-0044	Timer 1 reload register		
0x8000-0048	Timer 1 control register	0x8000-00D0	PCI reset control
0x8000-004C	Watchdog register		
0x8000-0050	Timer 2 counter register		
0x8000-0054	Timer 2 reload register		
0x8000-0058	Timer 2 control register	0x8000-00E0	UART3 data register
0x8000-0060	Prescalar counter register	0x8000-00E4	UART3 status register
0x8000-0064	Prescalar reload register	0x8000-00E8	UART3 control register
0x8000-0070	Uart 1 data register	0x8000-00EC	UART3 scalar register
0x8000-0074	Uart 1 status register		
0x8000-0078	Uart 1 control register	0x8000-0100	Memory edac configuration register 1
0x8000-007C	Uart 1 scalar register	0x8000-0104	Memory edac configuration register 2
0x8000-0080	Uart 2 data register	0x8000-0108	Memory edac configuration register 3
0x8000-0084	Uart 2 status register	0x8000-010C	Memory edac configuration register 4
0x8000-0088	Uart 2 control register		
0x8000-008C	Uart 2 scalar register	0x8000-0110	Expanded Cache Controller1
0x8000-0090	Interrupt Mask and Priority register	0x8000-0114	Expanded Cache Controller2
0x8000-0094	Interrupt pending register	0x8000-0118	Expanded Cache Controller3
0x8000-0098	Interrupt force register	0x8000-011C	Reserved
0x8000-009C	Interrupt clear register		

6.8.2 AHB status register

An error on the AHB bus will be registered in two registers: AHB failing address register (0x8000-000C) and AHB status register (0x8000-0010). The failing address register will store the address of the access while the AHB status register will store the access and error types. The registers are updated when an error occurs, and the NE bit of AHB status register is set. When the NE bit is set, interrupt 1 is generated to inform the processor about the error. After an error, the NE bit has to be reset by software.

6.8.3 Write Protection Register - WPR

Write protection is effective only to RAM area between 0x4000-0000 and 0x7FFF-FFFF. The ROM area can be written protected by clearing the ROM write enable bit MCFG1.

Write protection has two block protect units. Each block protect unit is controlled through a control register. The units operate as follows: on each write access to RAM, address bits (29:15) are xored with the tag field in the control register, and anded with the mask field. A write protection error is generated if the result is equal to zero, the corresponding unit is enabled and the block protect bit (BP) is set, or if the BP bit is cleared and the result is not equal to zero. If a write protection error is detected, the write cycle is aborted and a memory access error is generated.

Address: 0x8000-001C and 0x8000-0020

EN(31)	BP(30)	TAG(29:15)	MASK(14:0)

- MASK(14:0): Address mask. This field contains the address mask;
- TAG(29:15): Address tag . This field is compared against address(29:15);
- BP(30):Block protect. If set, selects block protect mode;
- EN (31): Enable. If set, enables the write protect unit.

6.8.4 Other registers setting

Other registers See Appendix 4.

6.9 PLL

PLL is implemented in the processor.

Pins of the PLL: pll_bp, pll_min[3:0], pll_rst, pll_out.

CLK is the clock input pin for the processor working. When BP pin is 0, then CLK multiply by PLL and is provided to the processor working clock. When BP pin is 1, then the PLL is bypassed.

PLL_MIN[3:0] is the multiple factor of the PLL, when BP is 0, PLL_MIN could not be 0(If PLL_MIN is 0， PLL will not work normally). PLL_MIN multiple number is from 1 to 15. The range of CLK frequency is 2MHz-30MHz.

The PLL stable time is 10ms.

6.10 JTAG Interface

6.10.1 Overview

The BM3803FMGRH implements a standard interface compliant with the IEEE 1149.1 JTAG specification. This interface can be used for PCB testing using the JTAG boundary-scan capability.

6.10.2 Application

The JTAG interface is accessed through five dedicated pins with details as shown in Table 6-10-1.

Table 6-10-1 Description of JTAG pins

Pin	I/O Type	Pullup/ Pulldown	Description
tck	I	pulldown	Tck is a clock signal, and can be asynchronous with CLK.

tms	I	Pullup	Primary control signal for the state machine. Synchronous with TCK.
tdi	I	Pullup	Serial input data to the boundary scan latches. Synchronous with TCK
tdo	O	——	Serial output data from the boundary scan latches. Synchronous with TCK.
trst	I	Pullup	Active-low resets the JTAG, and can be asynchronous with TCK. Please pulldown it when the JTAG unused.

Ensure the trst pin is pulldown when the JTAG unused. The recommended connection is as shown in Figure 6-10-1.

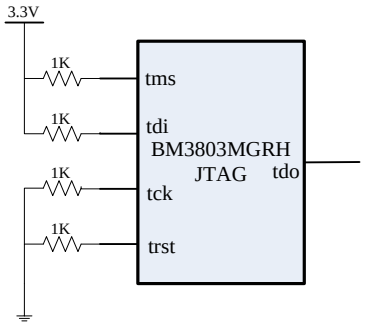


Figure 6-10-1 connection of the trst pin when the JTAG unused

As shown above, the tms and tdi pins connect to the 3.3V power supply through a 1K pullup resistance. The tck and trst pins connect to the ground through a 1K pulldown resistance.

When JTAG is in use, tms, tdi and tck pins connect to the external output signals, and the trst pin is recommended to connect as shown in Figure 6-10-2.

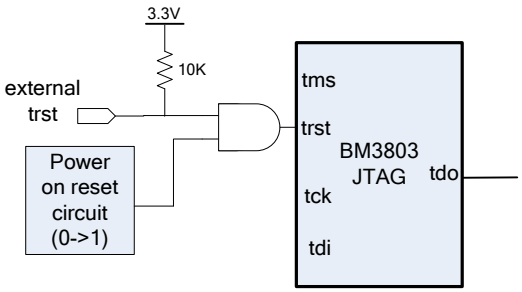


Figure 6-10-2 connection of the JTAG pins when the JTAG is in use

The external trst is an external input signal of JTAG, and connects to a 10K pullup resistance. The output of "Power on reset circuit" and the external trst are the

inputs of the AND gate. The output of the AND gate connects to the trst pin of the BM3803FMGRH.

6.11 Various state performance

See Appendix 2

7. Electricity parameter

See Appendix 2

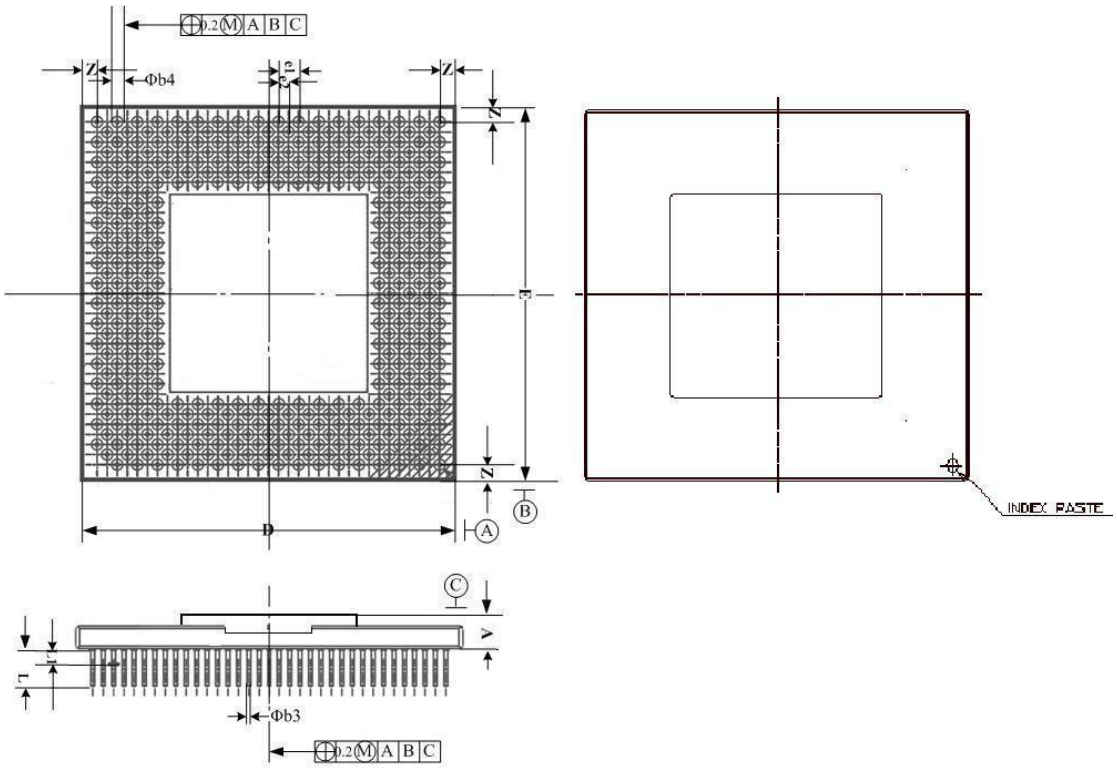
8. Characteristic application description

See Appendix 3

9. Package Description

The package of the processor is CPGA391.

Figure 9-1 shows the package information.



Dimension：mm

symbol	value		
	min	nominal	max
A	1.78	2.74	3.68
φb3	0.41	0.46	0.51
φb4	1.30	1.40	1.50
e1	——	2.54	——
e2	——	1.27	——
L_1	1.02	1.27	1.52
L	2.54	4.57	5.08
D	46.76	47.24	47.72
E	46.76	47.24	47.72
Z	1.91	2.03	2.54

Figure 9-1 package

Appendix1 BM3803FMGRH CPGA391 pin out

Table A1-1 BM3803FMGRH CPGA391 pin out and description

pin name	pin number	input/output	description
resetn	P4	input	Processor reset, when asserted, this active low input will reset the processor and all on-chip peripherals. Notice: the signal must be for at least 10 processor cycles after the clock stability.
calk	D28	input	Processor clock, the CLK input provides the main processor clock reference.
error	P6	open-drain output	Processor error, this active low output is asserted when the processor has entered error state and is halted. This happens when traps are disabled and a synchronous (un-maskable) trap occurs.
address[0]	M2	output	address bus, LSB
address[1]	N5	output	address bus
address[2]	M4	output	address bus
address[3]	K2	output	address bus
address[4]	M6	output	address bus
address[5]	J1	output	address bus
address[6]	L5	output	address bus
address[7]	J3	output	address bus
address[8]	G1	output	address bus
address[9]	J5	output	address bus
address[10]	F2	output	address bus
address[11]	K6	output	address bus
address[12]	G3	output	address bus
address[13]	D2	output	address bus
address[14]	F4	output	address bus
address[15]	E3	output	address bus
address[16]	B2	output	address bus
address[17]	C5	output	address bus
address[18]	D6	output	address bus
address[19]	B4	output	address bus

pin name	pin number	input/output	description
address[20]	F8	output	address bus
address[21]	C7	output	address bus
address[22]	B6	output	address bus
address[23]	A5	output	address bus
address[24]	E9	output	address bus
address[25]	C9	output	address bus
address[26]	D10	output	address bus
address[27]	B8	output	address bus, MSB
bexcn	N3	input/pull-up	Bus exception, this active low input is sampled simultaneously with the data during accesses on the memory bus. If asserted, a memory error will be generated. It's for PROM, SRAM and memory mapped I/O.
brdyn	G35	input/pull-up	Bus ready, when driven low, this input indicates to the processor that the current memory access can be terminated on the next rising clock edge. When driven high, this input indicates to the processor that it must wait and not end the current access. It's used to the fifth SRAM bank and memory mapped I/O.
data[0]	E11	bi-direction	data bus, LSB
data[1]	A9	bi-direction	data bus
data[2]	C11	bi-direction	data bus
data[3]	A11	bi-direction	data bus
data[4]	E13	bi-direction	data bus
data[5]	B12	bi-direction	data bus
data[6]	D14	bi-direction	data bus
data[7]	A13	bi-direction	data bus
data[8]	B14	bi-direction	data bus
data[9]	E15	bi-direction	data bus
data[10]	C15	bi-direction	data bus
data[11]	D16	bi-direction	data bus
data[12]	B16	bi-direction	data bus
data[13]	C17	bi-direction	data bus
data[14]	E17	bi-direction	data bus
data[15]	A17	bi-direction	data bus

pin name	pin number	input/output	description
data[16]	D18	bi-direction	data bus
data[17]	A19	bi-direction	data bus
data[18]	E19	bi-direction	data bus
data[19]	C19	bi-direction	data bus
data[20]	B20	bi-direction	data bus
data[21]	F20	bi-direction	data bus
data[22]	C21	bi-direction	data bus
data[23]	E21	bi-direction	data bus
data[24]	B22	bi-direction	data bus
data[25]	A23	bi-direction	data bus
data[26]	D22	bi-direction	data bus
data[27]	E23	bi-direction	data bus
data[28]	A25	bi-direction	data bus
data[29]	C25	bi-direction	data bus
data[30]	D24	bi-direction	data bus
data[31]	B26	bi-direction	data bus, MSB
duact	U1	output	DU active, this active high output is asserted when the processor is in debug mode and controlled by the DU.
dubre	V4	input	DU break enable, when the duen signal is high input, if during the three clock cycles before the end of the system reset the dubre signal is low input, the processor run the first four instructions to enter debug mode after the end of the system reset.
duen	W3	input	DU enable, the active high input enables the processor to enter debug mode when debugging trigger condition is met.
durx	V2	input/pull-up	DU receiver.
dutx	W5	output	DU transmitter.
pci_66	AK20	input	Test signal. Notice: this signal must be low input.
sim_fast_reset	AE1	input/pull-down	Test signal. Notice: this signal must meet 1KΩ resistor pull-down.
testin0	AC5	input/pull-down	Test signal. Notice: this signal must meet 1KΩ resistor pull-down.
testin1	AH4	input	Test signal. Notice: this signal must meet 1KΩ resistor pull-down.

pin name	pin number	input/output	description
testout0	AD2	output	Test signal.
testout1	AL1	bi-direction	Test signal. Notice: this signal must meet 1KΩ resistor pull-up.
pci_ad[0]	AR27	bi-direction	PCI address data, LSB
pci_ad[1]	AK24	bi-direction	PCI address data
pci_ad[2]	AP26	bi-direction	PCI address data
pci_ad[3]	AM24	bi-direction	PCI address data
pci_ad[4]	AN25	bi-direction	PCI address data
pci_ad[5]	AP24	bi-direction	PCI address data
pci_ad[6]	AK22	bi-direction	PCI address data
pci_ad[7]	AN23	bi-direction	PCI address data
pci_ad[8]	AR23	bi-direction	PCI address data
pci_ad[9]	AN21	bi-direction	PCI address data
pci_ad[10]	AR21	bi-direction	PCI address data
pci_ad[11]	AM20	bi-direction	PCI address data
pci_ad[12]	AP20	bi-direction	PCI address data
pci_ad[13]	AN19	bi-direction	PCI address data
pci_ad[14]	AL19	bi-direction	PCI address data
pci_ad[15]	AR19	bi-direction	PCI address data
pci_ad[16]	AM14	bi-direction	PCI address data
pci_ad[17]	AN13	bi-direction	PCI address data
pci_ad[18]	AK14	bi-direction	PCI address data
pci_ad[19]	AP12	bi-direction	PCI address data
pci_ad[20]	AL13	bi-direction	PCI address data
pci_ad[21]	AR11	bi-direction	PCI address data
pci_ad[22]	AP10	bi-direction	PCI address data
pci_ad[23]	AK12	bi-direction	PCI address data
pci_ad[24]	AP8	bi-direction	PCI address data
pci_ad[25]	AM8	bi-direction	PCI address data
pci_ad[26]	AR7	bi-direction	PCI address data
pci_ad[27]	AL9	bi-direction	PCI address data
pci_ad[28]	AR5	bi-direction	PCI address data
pci_ad[29]	AP6	bi-direction	PCI address data
pci_ad[30]	AK10	bi-direction	PCI address data
pci_ad[31]	AR3	bi-direction	PCI address data, MSB
pci_arb_gnt_n[0]	AL29	output	PCI bus grant, when asserted, this active low input indicates that a PCI agent is granted the PCI bus.

pin name	pin number	input/output	description
pci_arb_gnt_n[1]	AR33	output	PCI bus grant, when asserted, this active low input indicates that a PCI agent is granted the PCI bus.
pci_arb_gnt_n[2]	AK28	output	PCI bus grant, when asserted, this active low input indicates that a PCI agent is granted the PCI bus.
pci_arb_gnt_n[3]	AP32	output	PCI bus grant, when asserted, this active low input indicates that a PCI agent is granted the PCI bus.
pci_arb_gnt_n[4]	AM30	output	PCI bus grant, when asserted, this active low input indicates that a PCI agent is granted the PCI bus.
pci_arb_gnt_n[5]	AN31	output	PCI bus grant, when asserted, this active low input indicates that a PCI agent is granted the PCI bus.
pci_arb_gnt_n[6]	AM32	output	PCI bus grant, when asserted, this active low input indicates that a PCI agent is granted the PCI bus.
pci_arb_req[0]	AM26	input	PCI bus request, when asserted, this active low input indicates that a PCI agent is requesting the PCI bus.
pci_arb_req[1]	AN27	input	PCI bus request, when asserted, this active low input indicates that a PCI agent is requesting the PCI bus.
pci_arb_req[2]	AM28	input	PCI bus request, when asserted, this active low input indicates that a PCI agent is requesting the PCI bus.
pci_arb_req[3]	AR29	input	PCI bus request, when asserted, this active low input indicates that a PCI agent is requesting the PCI bus.
pci_arb_req[4]	AL27	input	PCI bus request, when asserted, this active low input indicates that a PCI agent is requesting the PCI bus.
pci_arb_req[5]	AR31	input	PCI bus request, when asserted, this active low input indicates that a PCI agent is requesting the PCI bus.
pci_arb_req[6]	AP30	input	PCI bus request, when asserted, this active low input indicates that a PCI agent is requesting the PCI bus.

pin name	pin number	input/output	description
pci_cbe[0]	AM22	bi-direction	PCI bus command and byte enables.
pci_cbe[1]	AK18	bi-direction	PCI bus command and byte enables.
pci_cbe[2]	AR13	bi-direction	PCI bus command and byte enables.
pci_cbe[3]	AL11	bi-direction	PCI bus command and byte enables.
pci_clk	AM4	input	PCI clock，33MHz
pci_devsel	AM16	bi-direction	PCI device select
pci_frame	AN15	bi-direction	PCI cycle frame
pci_gnt	AP4	input	PCI bus grant
pci_host	AE3	input	PCI host
pci_idry	AK16	bi-direction	PCI initiator ready
pci_idsel	AR9	input	PCI initialization device select
pci_inta_n	AK2	bi-direction	PCI interrupt A，this active low signal provides a interrupt signal.
pci_intb_n	AJ3	input	PCI interrupt B，this active low signal provides a interrupt signal.
pci_intc_n	AH6	input	PCI interrupt C，this active low signal provides a interrupt signal.
pci_intd_n	AM2	input	PCI interrupt D，this active low signal provides a interrupt signal.
pci_par	AR17	bi-direction	PCI parity
pci_perr	AN17	bi-direction	PCI parity error
pci_req	AK8	output	PCI bus request
pci_rst	AK4	bi-direction	PCI reset
pci_serr	AL17	bi-direction	PCI system error
pci_stop	AP16	bi-direction	PCI stop
pci_trdy	AR15	bi-direction	PCI target ready
pdata[0]	E25	bi-direction	check bits, LSB
pdata[1]	B28	bi-direction	check bits
pdata[2]	D26	bi-direction	check bits
pdata[3]	E27	bi-direction	check bits
pdata[4]	A31	bi-direction	check bits
pdata[5]	B30	bi-direction	check bits
pdata[6]	F26	bi-direction	check bits, MSB
pio[0]	AE33	bi-direction	Parallel I/O port, meanwhile, pio[1:0] have alternate function of PROM width, pio[1:0] defines PROM bus width at reset. pio[1:0] = "00"--8-bit width;"01"--16-bit width; "1x"--32-bit width.

pin name	pin number	input/output	description
pio[1]	AD32	bi-direction	Parallel I/O port, meanwhile, pio [1:0] have alternate function of PROM width, pio [1:0] defines PROM bus width at reset. Pio [1:0] = "00"--8-bit width;"01"--16-bit width; "1x"--32-bit width.
pio[2]	AF34	bi-direction	Parallel I/O port, meanwhile, pio [2] has alternate function of EDAC enable, pio [2] enable EDAC checking at reset.
pio[3]	AD30	bi-direction	Parallel I/O port, meanwhile, pio [3] has alternate function of UART clock, pio [3] use as alternative UART clock.
pio[4]	AG35	bi-direction	parallel I/O port
pio[5]	AE31	bi-direction	parallel I/O port
pio[6]	AH34	bi-direction	parallel I/O port
pio[7]	AH32	bi-direction	parallel I/O port
pci_serr	AL17	bi-direction	PCI system error
pci_stop	AP16	bi-direction	PCI stop
pci_trdy	AR15	bi-direction	PCI target ready
pdata[0]	E25	bi-direction	check bits, LSB
pdata[1]	B28	bi-direction	check bits
pdata[2]	D26	bi-direction	check bits
pdata[3]	E27	bi-direction	check bits
pdata[4]	A31	bi-direction	check bits
pdata[5]	B30	bi-direction	check bits
pdata[6]	F26	bi-direction	check bits, MSB
pio[0]	AE33	bi-direction	Parallel I/O port, meanwhile, pio [1:0] have alternate function of PROM width, pio [1:0] defines PROM bus width at reset. Pio [1:0] = "00"--8-bit width;"01"--16-bit width; "1x"--32-bit width.
pio[1]	AD32	bi-direction	Parallel I/O port, meanwhile, pio [1:0] have alternate function of PROM width, pio [1:0] defines PROM bus width at reset. Pio [1:0] = "00"--8-bit width;"01"--16-bit width; "1x"--32-bit width.
pio[2]	AF34	bi-direction	Parallel I/O port, meanwhile, pio [2] has alternate function of EDAC enable, pio [2] enable EDAC checking at reset.

pin name	pin number	input/output	description
pio[3]	AD30	bi-direction	Parallel I/O port, meanwhile, pio [3] has alternate function of UART clock, pio [3] use as alternative UART clock.
pio[4]	AG35	bi-direction	parallel I/O port
pio[5]	AE31	bi-direction	parallel I/O port
pio[6]	AH34	bi-direction	parallel I/O port
pio[7]	AH32	bi-direction	parallel I/O port
pio[8]	AJ35	bi-direction	Parallel I/O port, meanwhile, pio [8] has alternate function of UART2 clear-to-send.
pio[9]	AG31	bi-direction	Parallel I/O port, meanwhile, pio [9] has alternate function of UART2 request-to-send.
pio[10]	AL35	bi-direction	Parallel I/O port, meanwhile, pio [10] has alternate function of UART2 receiver data.
pio[11]	AJ33	bi-direction	Parallel I/O port, meanwhile, pio [11] has alternate function of UART2 transmitter data.
pio[12]	AJ31	bi-direction	Parallel I/O port, meanwhile, pio [12] has alternate function of UART1 clear-to-send.
pio[13]	AN35	bi-direction	Parallel I/O port, meanwhile, pio [13] has alternate function of UART1 request-to-send.
pio[14]	AH30	bi-direction	Parallel I/O port, meanwhile, pio [14] has alternate function of UART1 receiver data.
pio[15]	AM34	bi-direction	Parallel I/O port, meanwhile, pio [15] has alternate function of UART1 transmitter data.
pll_bp	K30	input	PLL bypass, when driven to VCC, this active high input set the PLL in bypass mode. The device is then directly clocked by the external clock. When grounded, the device is clocked trough the PLL.
pll_min[0]	G33	input/pull-down	PLL clock frequency multiplier input, LSB.
pll_min[1]	F28	input/pull-down	PLL clock frequency multiplier input.
pll_min[2]	A33	input/pull-down	PLL clock frequency multiplier input.
pll_min[3]	E29	input/pull-down	PLL clock frequency multiplier input, MSB.
pll_out	E35	output	PLL clock output. Notice: this signal is

pin name	pin number	input/output	description
			only used to test, not for control.
pll_rst	C29	input	PLL reset. Notice: this signal must meet 1KΩ resistor pull-down.
ramoen[0]	M34	output	RAM output enable, this active low signal provides the output enable for the first RAM bank.
ramoen[1]	N33	output	RAM output enable, this active low signal provides the output enable for the second RAM bank.
ramoen[2]	P32	output	RAM output enable, this active low signal provides the output enable for the third RAM bank.
ramoen[3]	R31	output	RAM output enable, this active low signal provides the output enable for the fourth RAM bank.
ramoen[4]	R33	output	RAM output enable, this active low signal provides the output enable for the fifth RAM bank.
ramsn[0]	H34	output	RAM chip-select, this active low output provides the chip-select signal for the first RAM bank.
ramsn[1]	L31	output	RAM chip-select, this active low output provides the chip-select signal for the second RAM bank.
ramsn[2]	K34	output	RAM chip-select, this active low output provides the chip-select signal for the third RAM bank.
ramsn[3]	M32	output	RAM chip-select, this active low output provides the chip-select signal for the fourth RAM bank.
ramsn[4]	L33	output	RAM chip-select, this active low output provides the chip-select signal for the fifth RAM bank.
read	J33	output	Read transaction, this active high output is asserted during read transaction on the memory bus. On the contrary, this active low output is asserted during write transaction on the memory bus. Notice: it's only for instructions, can’t serve as control.

pin name	pin number	input/output	description
rombsd[0]	AE5	input	the Prom bank size of 8-bit wide data bus，LSB.
rombsd[1]	AH2	input	The Prom bank size of 8-bit wide data bus.
rombsd[2]	AF4	input	The Prom bank size of 8-bit wide data bus.
rombsd[3]	AG3	input	the Prom bank size of 8-bit wide data bus，MSB.
romsn[0]	U35	output	PROM chip-select, this active low output provides the chip-select signal for PROM area.
romsn[1]	V30	output	PROM chip-select, this active low output provides the chip-select signal for PROM area.
iosn	V34	output	I/O select, this active low output provides the chip-select signal for the memory mapped I/O area.
oen	H32	output	Output enable, this active low output is asserted during read transactions on the memory bus. It’s used to PROM interface and memory mapped I/O.
writen	K32	output	Write enable, this active low output provides a write strobe during write transactions on the memory bus. It’s used to PROM interface and memory mapped I/O.
rwen[0]	T30	output	RAM write enable, this active low output provides a write strobe for data[31:24] byte.
rwen[1]	R35	output	RAM write enable, this active low output provides a write strobe for data[23:16] byte.
rwen[2]	T32	output	RAM write enable, this active low output provides a write strobe for data[15:8] byte.
rwen[3]	T34	output	RAM write enable, this active low output provides a write strobe for data[7:0] byte.
sdclk	AC31	output	SDRAM clock, this provides the SDRAM interface clock reference.
sdcsn[0]	W31	output	SDRAM chip select, this active low output provides the chip select signal for the first SDRAM bank.

pin name	pin number	input/output	description
sdcsn[1]	W33	output	SDRAM chip select, this active low output provides the chip select signal for the second SDRAM bank.
sddqm[0]	AB34	output	SDRAM data mask, this active low output provides the DQM signal for both SDRAM banks. This is for data [7:0].
sddqm[1]	AC35	output	SDRAM data mask, this active low output provides the DQM signal for both SDRAM banks. This is for data [15:8].
sddqm[2]	AB32	output	SDRAM data mask, this active low output provides the DQM signal for both SDRAM banks. This is for data [23:16].
sddqm[3]	AC33	output	SDRAM data mask, this active low output provides the DQM signal for both SDRAM banks. This is for data [31:24].
sdrasn	Y32	output	SDRAM row address strobe, this active low signal provides a common RAS for all SDRAM devices.
sdcasn	AA35	output	SDRAM column address strobe, this active low signal provides a common CAS for all SDRAM devices.
sdwen	Y34	output	SDRAM write strobe, this active low signal provides a common write strobe for all SDRAM devices.
scan_enable	G31	input/pull-down	Scan mode enable. Notice: this signal must meet 1KΩ resistor pull-down.
scan_mode	E1	input/pull-down	scan mode select, Notice: this signal must meet 1KΩ resistor pull-down.
bist_mode	C35	input/pull-down	Mbist mode enables. Notice: this signal must meet 1KΩ resistor pull-down.
skew[0]	AG1	input/pull-down	Clock tree skew. Notice: this signal must meet 1KΩ resistor pull-down.
skew[1]	AD6	input/pull-down	Clock tree skew. Notice: this signal must meet 1KΩ resistor pull-down.
tclk	T2	input/pull-down	JTAG test clock. Notice: this signal must meet 1KΩ resistor pull-down when JTAG is not in use.
tdi	T4	input/pull-up	JTAG test data input. Notice: this signal

pin name	pin number	input/output	description
			must meet 1KΩ resistor pull-up when JTAG is not in use.
tdo	R1	tri_state output	JTAG test data output.
tms	P2	input/pull-up	JTAG test mode select, this signal must meet 1KΩ resistor pull-up when JTAG is not in use.
trst	N1	input/pull-up	JTAG test reset, this active low signal provides a reset for JTAG test. Notice: this signal must meet 1KΩ resistor pull-down when JTAG is not in use.
timer0	Y4	output	timer0 output
timer1	Y2	output	timer1 output
wdogn	AB6	open-drain output	Watchdog time-out, this active low output is asserted when the watchdog expires.
rxd3	AB2	input/pull-up	UART3 data input
txd3	AC1	output	UART3 data output
rts3	AA5	output	UART3 request-to-send
cts3	AA3	input/pull-up	UART3 clear-to-send
pll_corevdd	D32	power	PLL AVDD18
pll_corevss	E33	power	PLL AGND18
pll_iovdd	D30	power	PLL AVDD33
pll_iovss	C31	power	PLL AGND33
vdd_core	A15	power	VDD18
vdd_core	A21	power	VDD18
vdd_core	A27	power	VDD18
vdd_core	D12	power	VDD18
vdd_core	E7	power	VDD18
vdd_core	F34	power	VDD18
vdd_core	K4	power	VDD18
vdd_core	L35	power	VDD18
vdd_core	R5	power	VDD18
vdd_core	U31	power	VDD18
vdd_core	AA1	power	VDD18
vdd_core	AD4	power	VDD18
vdd_core	AE35	power	VDD18
vdd_core	AJ1	power	VDD18
vdd_core	AJ5	power	VDD18
vdd_core	AK34	power	VDD18

pin name	pin number	input/output	description
vdd_core	AL15	power	VDD18
vdd_core	AM10	power	VDD18
vdd_core	AP22	power	VDD18
vdd_core	AP28	power	VDD18
vdd_io	B18	power	VDD33
vdd_io	B24	power	VDD33
vdd_io	C1	power	VDD33
vdd_io	C13	power	VDD33
vdd_io	C27	power	VDD33
vdd_io	D4	power	VDD33
vdd_io	D8	power	VDD33
vdd_io	H30	power	VDD33
vdd_io	J35	power	VDD33
vdd_io	L3	power	VDD33
vdd_io	N35	power	VDD33
vdd_io	U5	power	VDD33
vdd_io	V6	power	VDD33
vdd_io	V32	power	VDD33
vdd_io	Y30	power	VDD33
vdd_io	AB4	power	VDD33
vdd_io	AF32	power	VDD33
vdd_io	AL3	power	VDD33
vdd_io	AL33	power	VDD33
vdd_io	AN7	power	VDD33
vdd_io	AN11	power	VDD33
vdd_io	AN29	power	VDD33
vdd_io	AP18	power	VDD33
vdd_io	AR25	power	VDD33
vss_core	A3	power	GND18
vss_core	B10	power	GND18
vss_core	D20	power	GND18
vss_core	F16	power	GND18
vss_core	F24	power	GND18
vss_core	H2	power	GND18
vss_core	J31	power	GND18
vss_core	N31	power	GND18
vss_core	R3	power	GND18
vss_core	U33	power	GND18

pin name	pin number	input/output	description
vss_core	Y6	power	GND18
vss_core	AD34	power	GND18
vss_core	AF2	power	GND18
vss_core	AF30	power	GND18
vss_core	AG5	power	GND18
vss_core	AL21	power	GND18
vss_core	AL25	power	GND18
vss_core	AN1	power	GND18
vss_core	AN9	power	GND18
vss_core	AP14	power	GND18
vss_io	A7	power	GND33
vss_io	A29	power	GND33
vss_io	C3	power	GND33
vss_io	C23	power	GND33
vss_io	D34	power	GND33
vss_io	F14	power	GND33
vss_io	F18	power	GND33
vss_io	G5	power	GND33
vss_io	L1	power	GND33
vss_io	M30	power	GND33
vss_io	P34	power	GND33
vss_io	U3	power	GND33
vss_io	W1	power	GND33
vss_io	W35	power	GND33
vss_io	AA33	power	GND33
vss_io	AC3	power	GND33
vss_io	AG33	power	GND33
vss_io	AK26	power	GND33
vss_io	AL7	power	GND33
vss_io	AL23	power	GND33
vss_io	AL31	power	GND33
vss_io	AM12	power	GND33
vss_io	AM18	power	GND33
vss_io	AN3	power	GND33
N/C	A35	—	N/C
N/C	B32	—	N/C
N/C	B34	—	N/C
N/C	C33	—	N/C

pin name	pin number	input/output	description
N/C	E5	—	N/C
N/C	E31	—	N/C
N/C	F6	—	N/C
N/C	F10	—	N/C
N/C	F12	—	N/C
N/C	F22	—	N/C
N/C	F30	—	N/C
N/C	F32	—	N/C
N/C	G7	—	Alumina coat, this signal must be grounded.
N/C	G9	—	N/C
N/C	G11	—	N/C
N/C	G13	—	N/C
N/C	G15	—	N/C
N/C	G17	—	N/C
N/C	G19	—	N/C
N/C	G21	—	N/C
N/C	G23	—	N/C
N/C	G25	—	N/C
N/C	G27	—	N/C
N/C	G29	—	N/C
N/C	H4	—	N/C
N/C	H6	—	N/C
N/C	J7	—	N/C
N/C	J29	—	N/C
N/C	L7	—	N/C
N/C	L29	—	N/C
N/C	N7	—	N/C
N/C	N29	—	N/C
N/C	P30	—	N/C
N/C	R7	—	N/C
N/C	R29	—	N/C
N/C	T6	—	N/C
N/C	U7	—	N/C
N/C	U29	—	N/C
N/C	W7	—	N/C
N/C	W29	—	N/C
N/C	AA7	—	N/C

pin name	pin number	input/output	description
N/C	AA29	—	N/C
N/C	AA31	—	N/C
N/C	AB30	—	N/C
N/C	AC7	—	N/C
N/C	AC29	—	N/C
N/C	AE7	—	N/C
N/C	AE29	—	N/C
N/C	AF6	—	N/C
N/C	AG7	—	N/C
N/C	AG29	—	N/C
N/C	AJ7	—	N/C
N/C	AJ9	—	N/C
N/C	AJ11	—	N/C
N/C	AJ13	—	N/C
N/C	AJ15	—	N/C
N/C	AJ17	—	N/C
N/C	AJ19	—	N/C
N/C	AJ21	—	N/C
N/C	AJ23	—	N/C
N/C	AJ25	—	N/C
N/C	AJ27	—	N/C
N/C	AJ29	—	Alumina coat, this signal must be grounded.
N/C	AK6	—	N/C
N/C	AK30	—	N/C
N/C	AK32	—	N/C
N/C	AL5	—	N/C
N/C	AM6	—	N/C
N/C	AN5	—	N/C
N/C	AN33	—	N/C
N/C	AP2	—	N/C
N/C	AP34	—	N/C
N/C	AR1	—	N/C
N/C	AR35	—	N/C

Note:

pull-up/pull-down: There's pull-up/pull-down in the pad of pin.

Table A1-2 Output drive characteristics

I_{OH}	-0.5mA	-4mA	-8mA	-12mA	-16mA	NA (open-drain bi-direction)	NA (open-drain output)	-4mA (tri-state output)
I_{OL}	1.5 mA	4mA	8mA	12mA	16mA	1.5mA (open-drain bi-direction)	4mA (open-drain output)	4mA (tri-state output)
Pin Name	pci_arb_gnt_n[6:0] pci_ad[31:0] pci_cbe_n[3:0] pci_par pci_frame_n pci_idry_n pci_trdy_n pci_stop_n pci_devsel_n pci_req_n pci_perr_n pci_rst_n	data[31:0] pardata[6:0] ramsn[4:0] ramoen[4:0] read romsn[1:0] iosn rts3 timer0 timer1 pll_out	address[27:0] oen writen rwen[3:0] sdcsn[1:0] sddqm[3:0] sdrasn sdcasn sdwen pio[15:0]	txd3 dutx duact	sdclk	pci_serr_n pci_inta_n	errorn wdogn	tdo

Appendix2 Electrical characteristics

2.1 DC characteristic

Table A2-1 DC characteristics

Parameter	Symbol	test conditions default(-55℃≤T_A≤125℃, 1.65V≤V_{DDD}≤1.95V, 1.65V≤V_{DDA}≤1.95V, 3.0V≤V_{DDDIO}≤3.6V, 3.0V≤V_{DDAIO}≤3.6V)		min	max	unit
high level output voltage	V_{OH}	V_{DDD}=1.65V, V_{DDA}=1.65V, V_{DDDIO}=3.0V, V_{DDAIO}=3.0V,	I_{OH}=-0.5 mA	2.7	—	V
			I_{OH}= -4, -8, -12, -16 mA	2.6	—	
low level output voltage	V_{OL}		I_{OL}=1.5 mA	—	0.3	V
			I_{OL}= 4, 8, 12, 16 mA	—	0.4	
high level input voltage	V_{IH}	*test all input pins*		2.0	—	
low level input voltage	V_{IL}			—	0.8	
high level input leakage current	I_{IH}	V_{DDD}= 1.95V, V_{DDA}= 1.95V, V_{DDDIO}= 3.6V, V_{DDAIO}= 3.6V, V_I= 3.6V		—	1	μA
high level input pull-up current	I_{IHPU}			—	5	μA
high level input pull-down current	I_{IHPD}			40	200	μA
low level input leakage current	$\lvert I_{IL} \rvert$	V_{DDD}=1.95V, V_{DDA}=1.95V, V_{DDDIO}=3.6V, V_{DDAIO}= 3.6V, V_I= 0V		—	1	μA
low level input pull-up current	$\lvert I_{ILPU} \rvert$			40	200	μA

Parameter	Symbol	test conditions default(-55℃≤T_A≤125℃, 1.65V≤V_{DDD}≤1.95V, 1.65V≤V_{DDA}≤1.95V, 3.0V≤V_{DDDIO}≤3.6V, 3.0V≤V_{DDAIO}≤3.6V)		min	max	unit
low level input pull-down current	$\lvert I_{ILPD} \rvert$			—	5	μA
output leakage current tri-state (high level applied)	I_{OZH}	V_{DDD}=1.95V, V_{DDA}=1.95V, V_{DDDIO}=3.6V, V_{DDAIO}= 3.6V, V_O=3.6V		—	1	μA
output leakage current tri-state (low level applied)	$\lvert I_{OZL} \rvert$	V_{DDD}=1.95V, V_{DDA}=1.95V, V_{DDDIO}=3.6V, V_{DDAIO}= 3.6V, V_O=0V		—	1	μA
standby current	$I_{DDD(SB)}$	V_{DDD}=1.95V V_{DDA}=1.95V V_{DDDIO}=3.6V V_{DDAIO}=3.6V clk and pci_clk are not active	V_{DDD}	—	10	mA
			V_{DDDIO}	—	5	
standard input capacitance	C_{IN}	T_A =25℃		—	12	pF
standard input/output capacitance	C_{IO}	T_A =25℃		—	12	pF
power consumption	$I_{DDD(dy)}$	f=100MHz, with zero load	Test point V_{DDD}	—	240	mA
			Test point V_{DDA}		10	
			Test point V_{DDDIO}		90	

2.2 AC characteristic

Table A2-2 AC characteristics

comment	parameter	test conditions default (SKEW[1:0]="00", -55℃≤T_A≤125℃, 1.65V≤V_{DDD}≤1.95V, 1.65V≤V_{DDA}≤1.95V, 3.0V≤V_{DDDIO}≤3.6V, 3.0V≤V_{DDAIO}≤3.6V)	min	max	unit
clk period with PLL disable	t_1	V_{DDD}＝1.65V,	10	—	ns

clk low and high pulse width - PLL disabled	t_2	V_{DDA}=1.65V, V_{DDDIO}=3.0V, V_{DDAIO}=3.0V, f=20MHz	4.5	—	ns
clk low and high pulse width - PLL enabled	t_{2p}		15	—	ns
SDCLK period	t_3		10	—	ns
SDCLK output delay - PLL disabled	t_4		2	7	ns
PLL setup time	t_5		—	10^7	ns
Reset pulse width	t_6		10* t_3	—	ns
address[27:0] output delay	t_{10}		1.5	11	ns
data[31:0] and pardata[6:0] output delay	t_{11}		2	13	ns
data[31:0] and pardata[6:0] setup time	t_{12}		4.5	—	ns
data[31:0] and pardata[6:0] hold time during load/fetch	t_{13}		0	—	ns
data[31:0] and pardata[6:0] hold time during write	t_{14}		2	13	ns
Oen output delay	t_{15}		2	11	ns
written output delay	t_{17}		1.5	11	ns
Romsn[1:0] output delay	t_{18}		2	13	ns
Ramsn[4:0] output delay	t_{19}		2	13	ns
Ramoen[4:0]output delay	t_{20}		2	13	ns
rwen[3:0] output delay	t_{21}		2	13	ns
iosn output delay	t_{22}		2	13	ns
Brdyn setup time	t_{23}		4	—	ns
Brdyn hold time	t_{24}		0	—	ns
sdcasn output delay	t_{25}		3	9	ns
sdcsn[1:0] output delay	t_{26}		2	8.5	ns
sdrasn output delay	t_{27}		2	8.5	ns
sdwen output delay	t_{28}		2	8.5	ns
sddqm[3:0] output delay	t_{29}		2	8.5	ns
pio[15:0]output delay	t_{31}		2.5	10	ns
pio[15:0]setup time	t_{32}		4.5	—	ns
pio[15:0] hold time during load	t_{33}		0	—	ns
pio[15:0] hold time during write	t_{34}		2.5	—	ns
Bexcn setup time	t_{35}		4	—	ns
Bexcn hold time	t_{36}		0	—	ns
Pci_clk period	t_{101}		30	—	ns
PCI_CLK low and high pulse width	t_{102}		14.5	—	ns

Pci_ad_n[31:0] and pci_cbe_n[3:0] output delay	t_{110}		4	12	ns
Pci_ad_n[31:0] and pci_cbe_n[3:0] setup time	t_{111}		6	—	ns
Pci_ad_n[31:0] and pci_cbe_n[3:0] hold time	t_{112}		0	—	ns
Pci_frame_n, pci_par_n, pci_perr_n, pci_serr_n, pci_stop_n, and pci_devsel_n output delay	t_{113}		4	11	ns
pci_irdy_nand pci_trdy_n output delay	t_{114}		4	11	ns
pci_req_n output delay	t_{115}		4	12	ns
Pci_frame_n, pci_par_n, pci_perr_n, pci_serr_n, pci_stop_n, pci_idsel_n, and pci_devsel_n setup time	t_{116}		7	—	ns
pci_irdy_nand pci_trdy_n setup time	t_{117}		7	—	ns
pci_gnt_in_n setup time	t_{118}		6	—	ns
Pci_frame_n, pci_par_n, pci_perr_n, pci_serr_n, pci_stop_n, pci_idsel_n, and pci_devsel_n setup time	t_{119}		0	—	ns
pci_irdy_nand pci_trdy_n setup time	t_{120}		0	—	ns
pci_gnt_in_n setup time	t_{121}		0	—	ns

2.3 Timing Diagrams

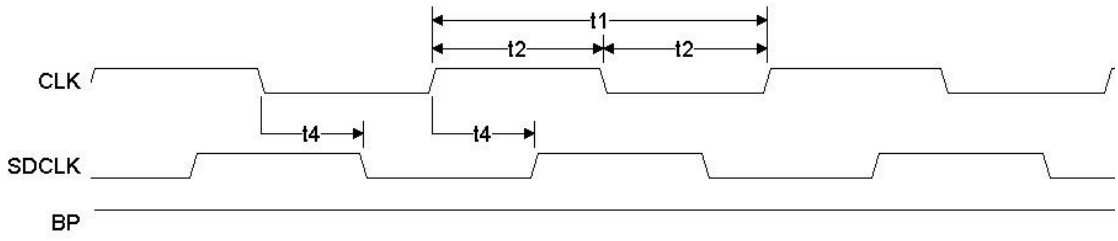


Figure A2-1 clock input without PLL

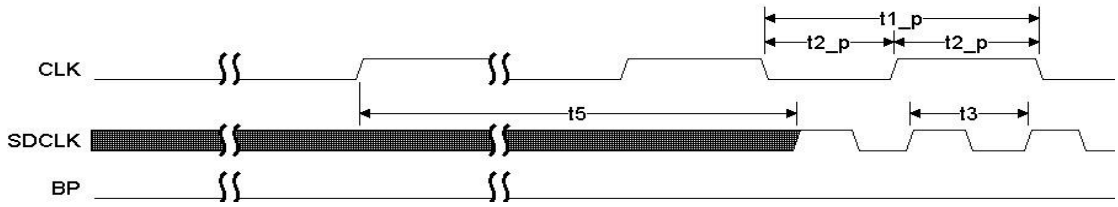


Figure A2-2 clock input with PLL

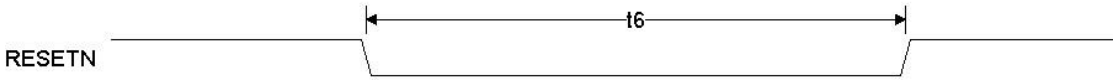


Figure A2-3 Reset sequence

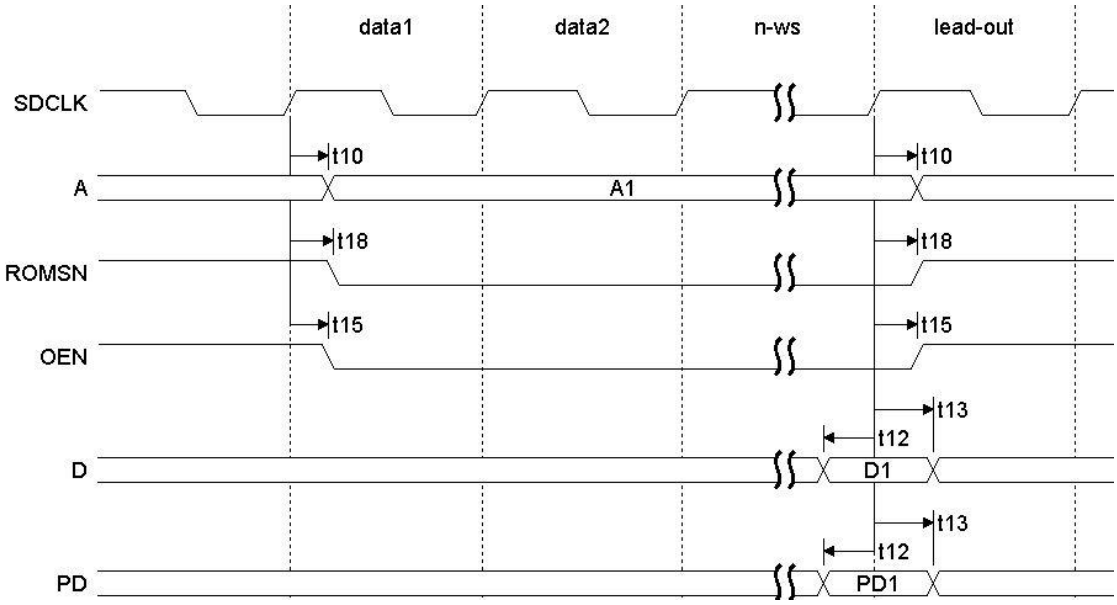


Figure A2-4 read from 32-bit PROM

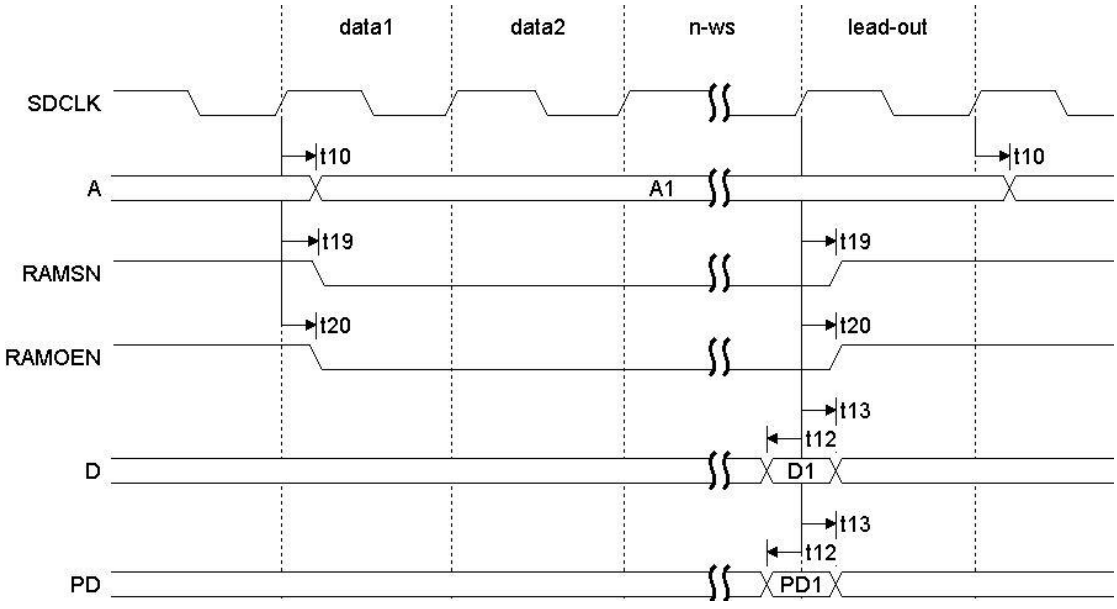


Figure A2-5 read from 32-bit SRAM

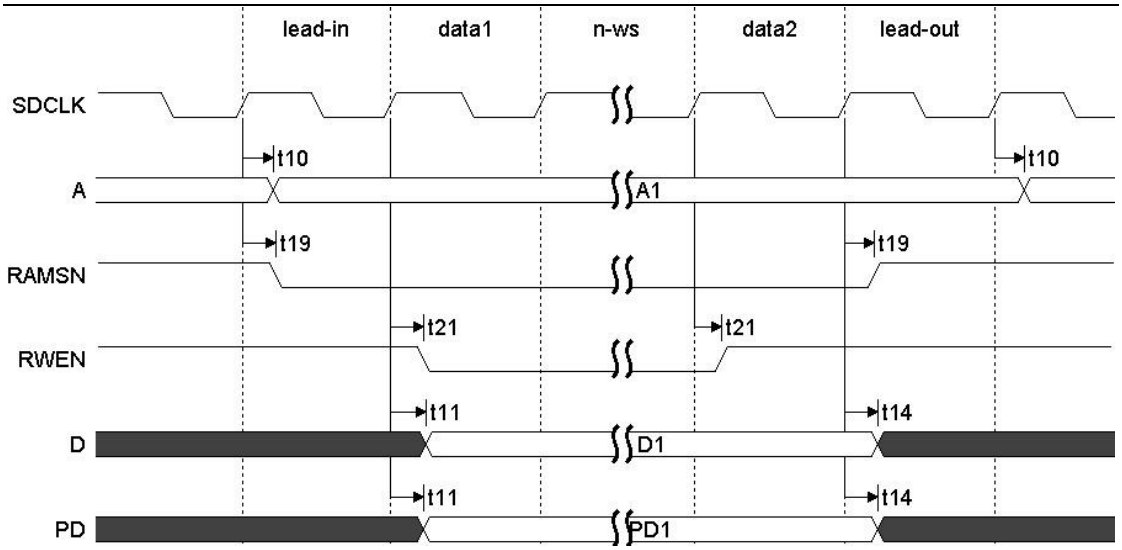


Figure A2-6 write to 32-bit SRAM

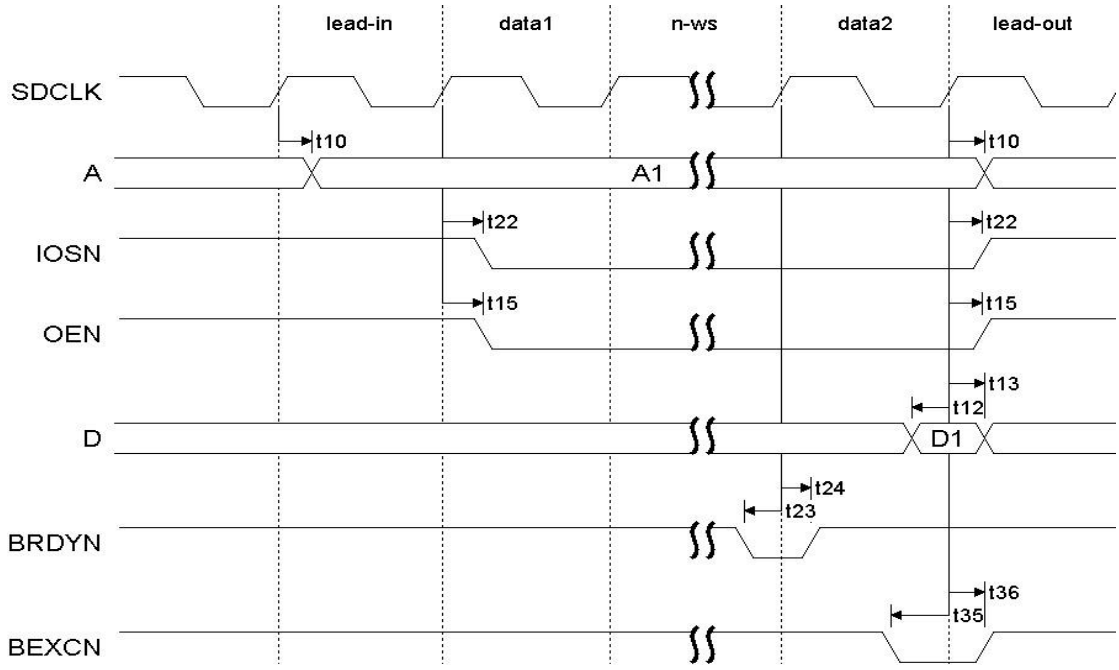


Figure A2-7 read from 32-bit I/O

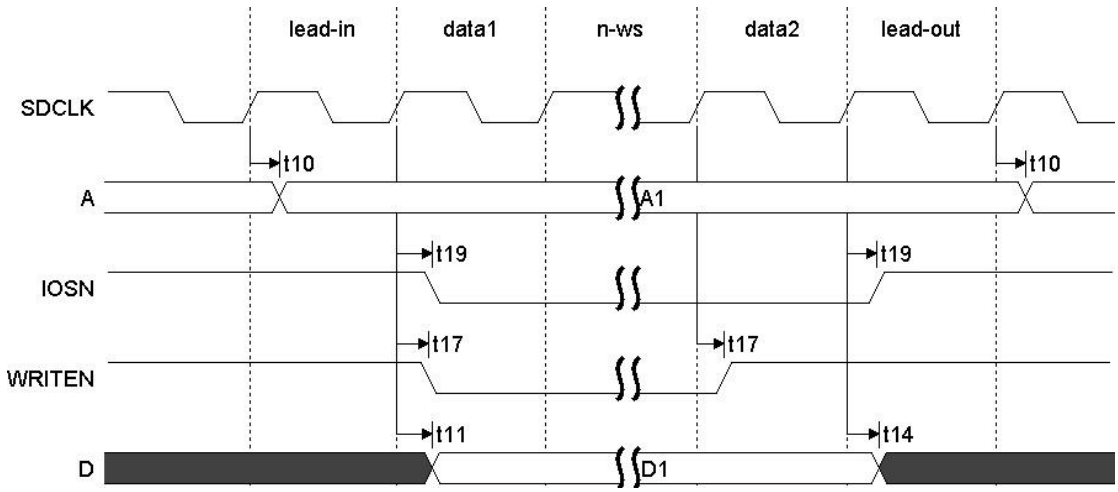


Figure A2-8 write to 32-bit I/O

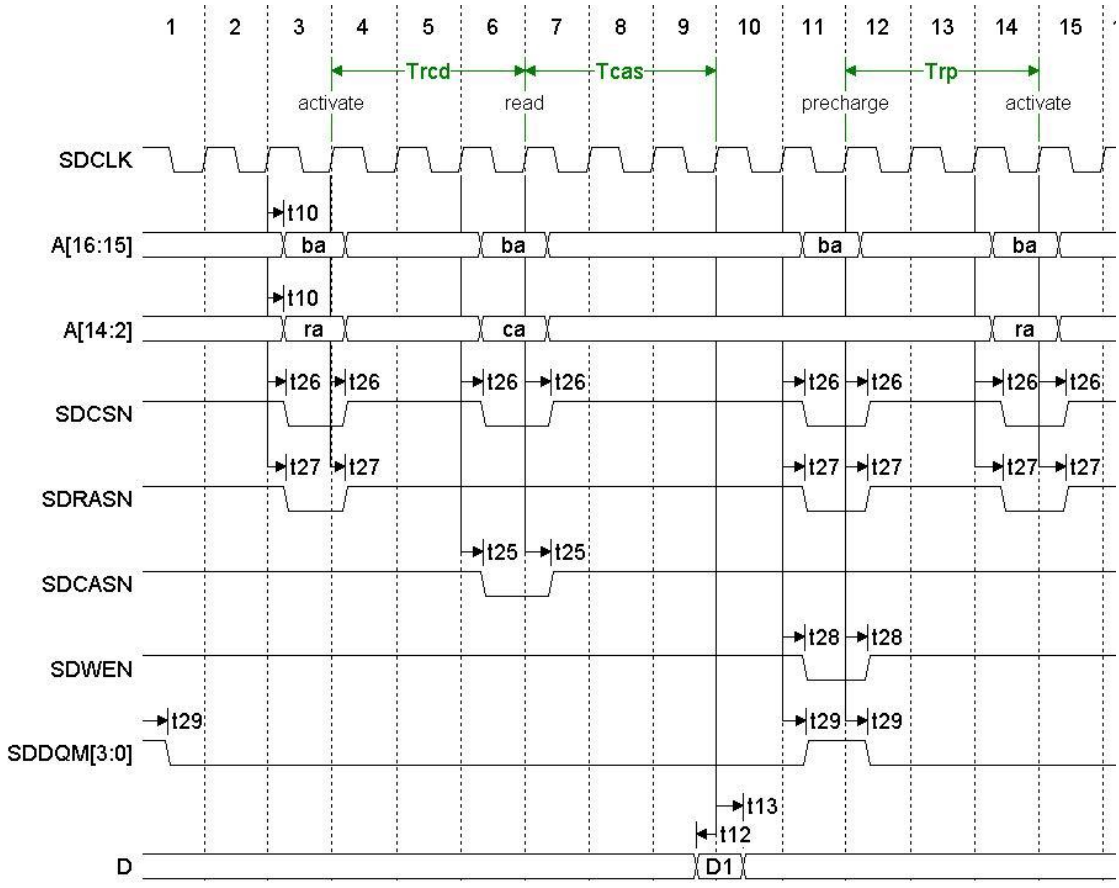


Figure A2-9　SDRAM read

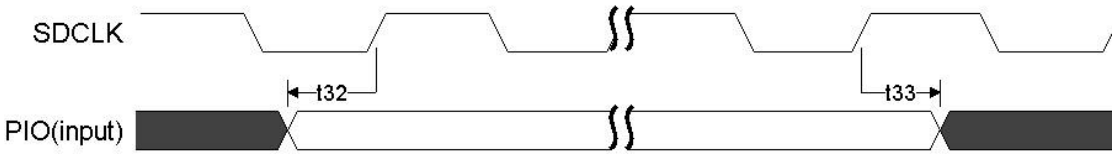


Figure A2-10　PIO input

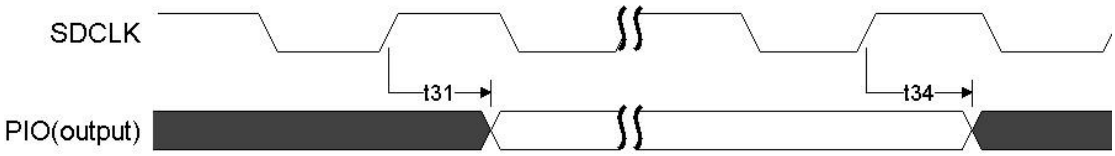


Figure A2-11　PIO output

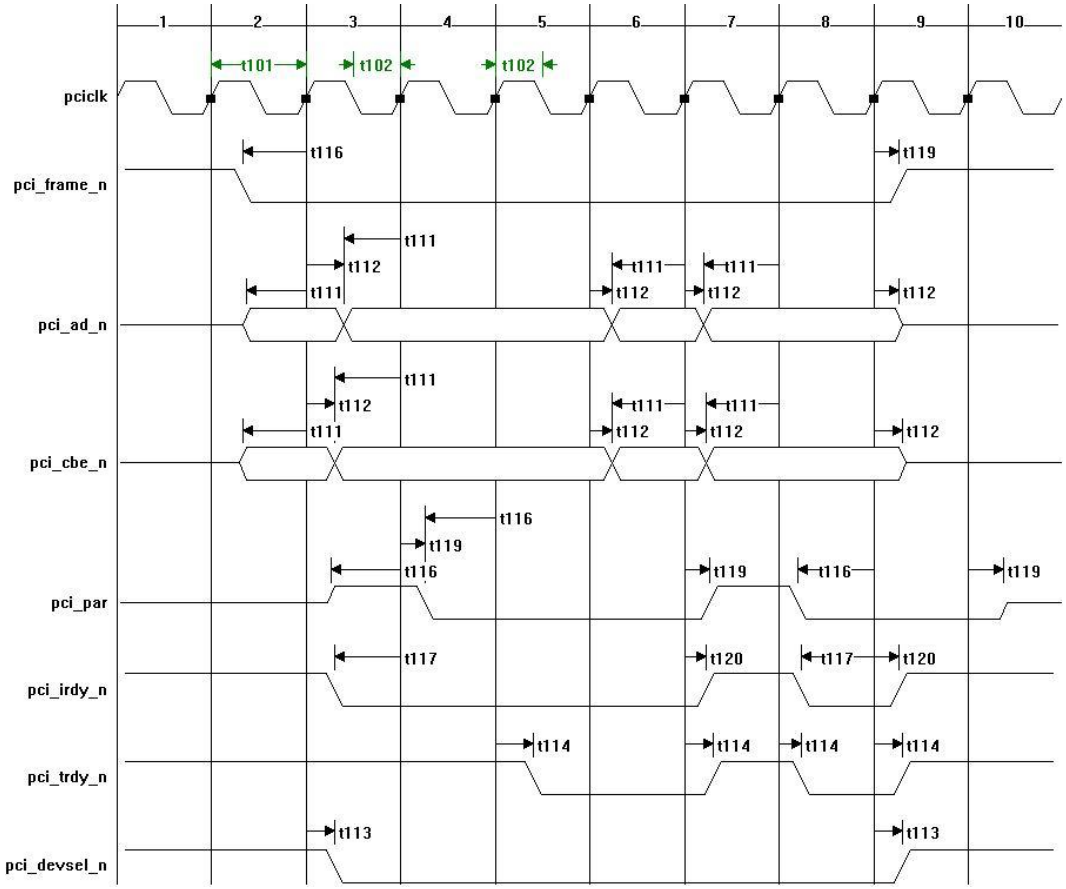


Figure A2-12　PCI write

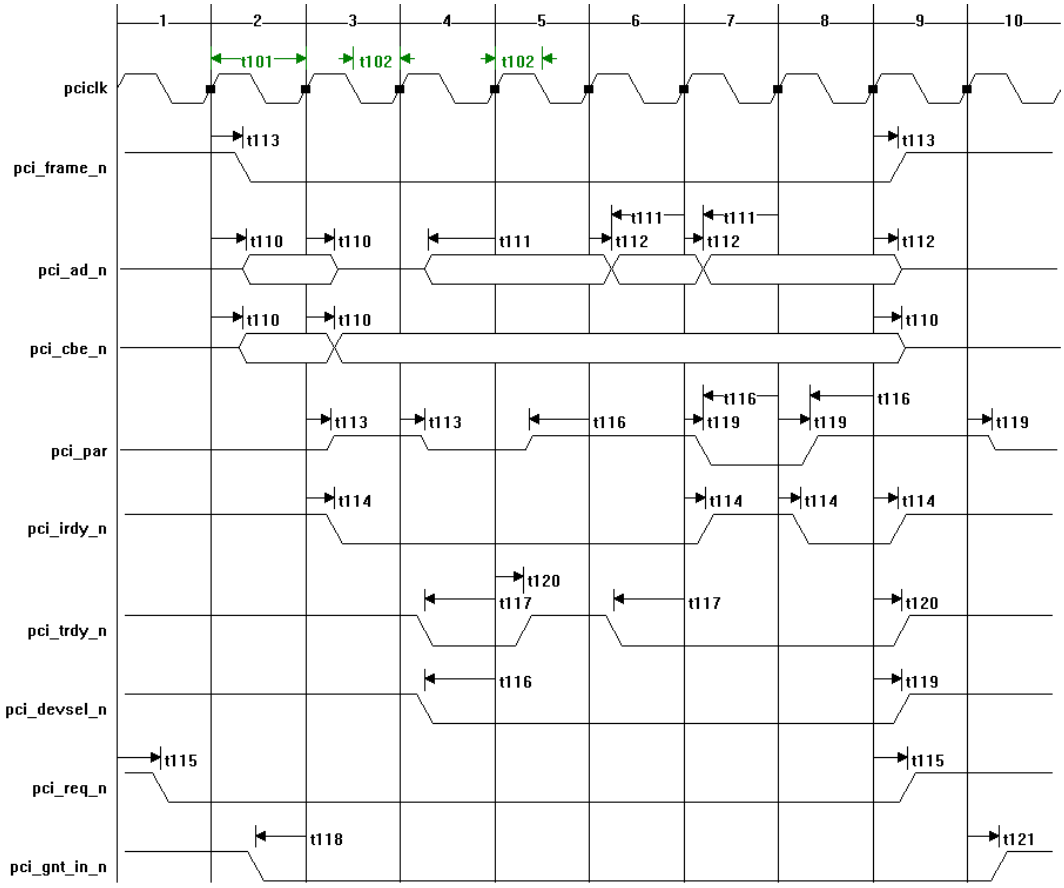


Figure A2-13 PCI read

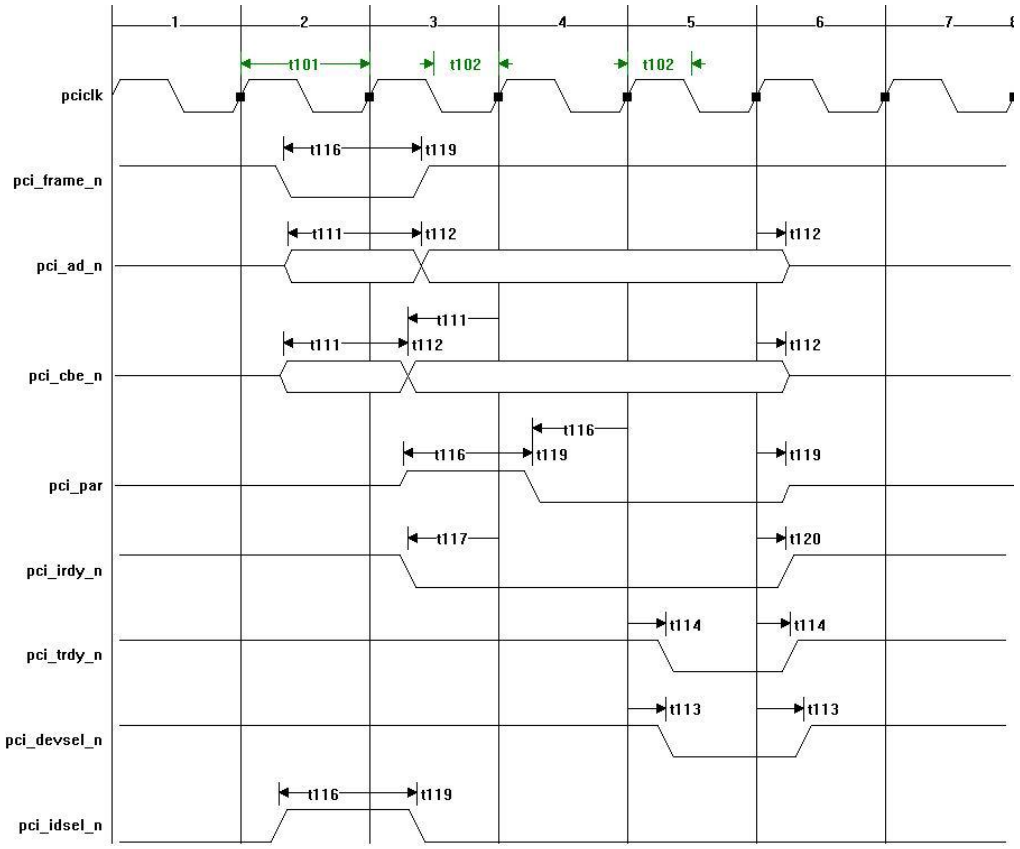


Figure A2-14　PCI configure write

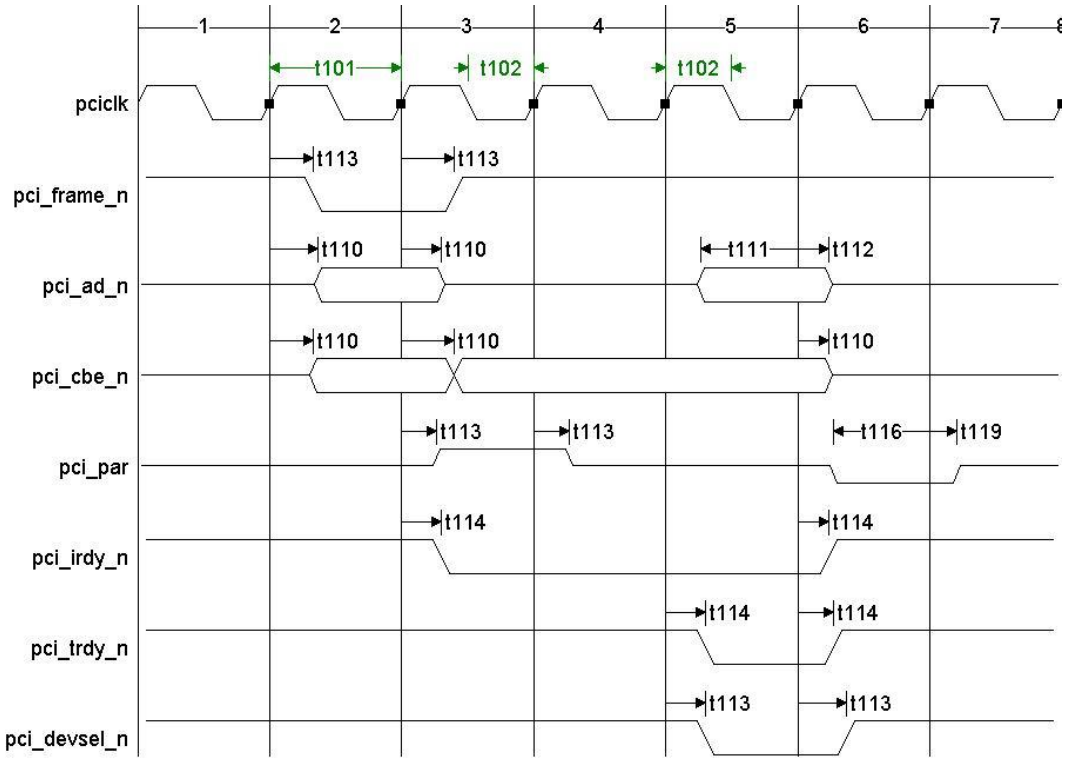


Figure A2-15 PCI configure read

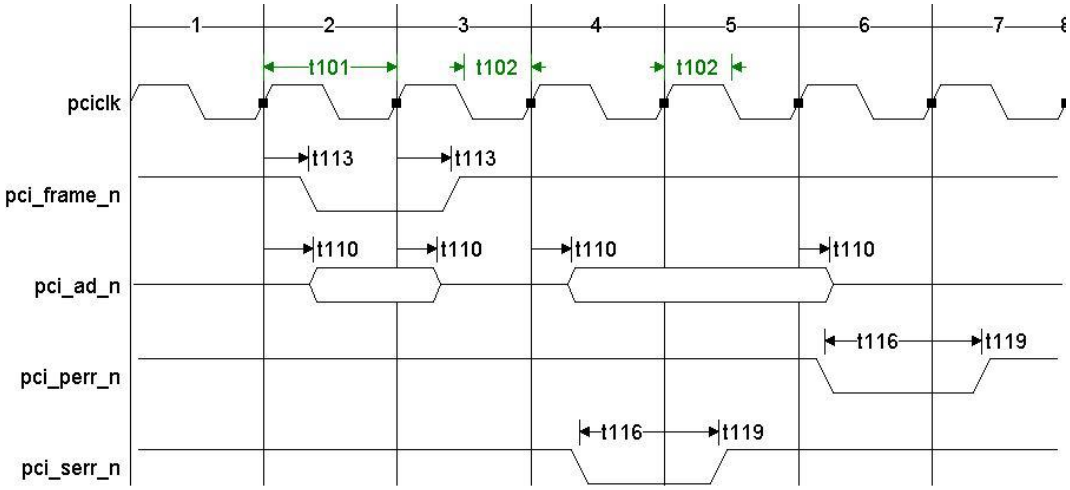


Figure A2-16 Pci_perr_n and pci_serr_n input

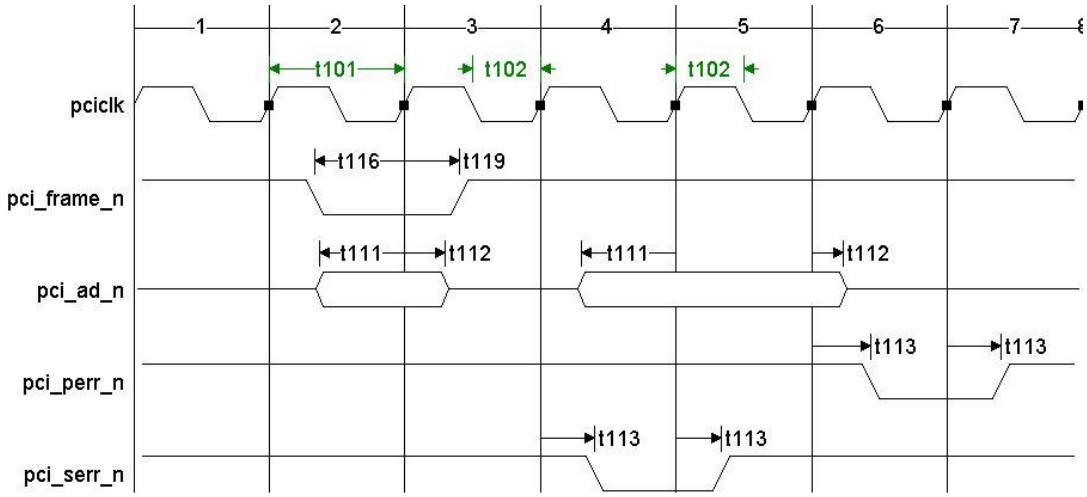


Figure A2-17　Pci_perr_n and pci_serr_n output

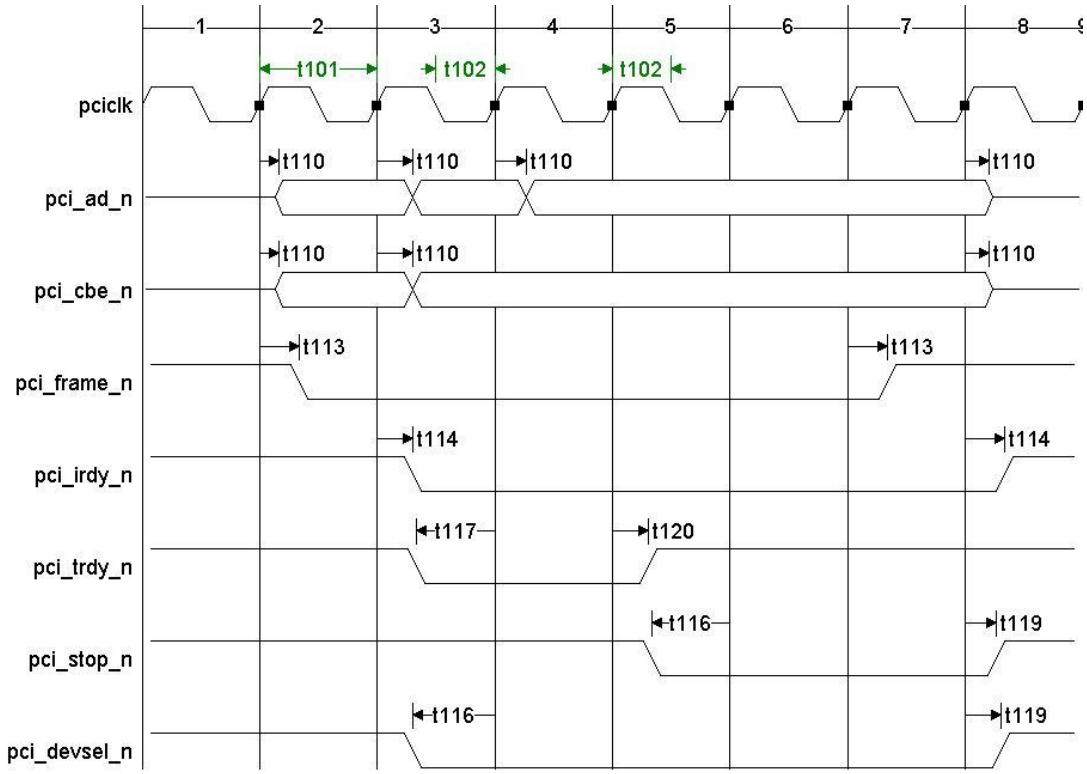


Figure A2-18　Pci_stop_n input

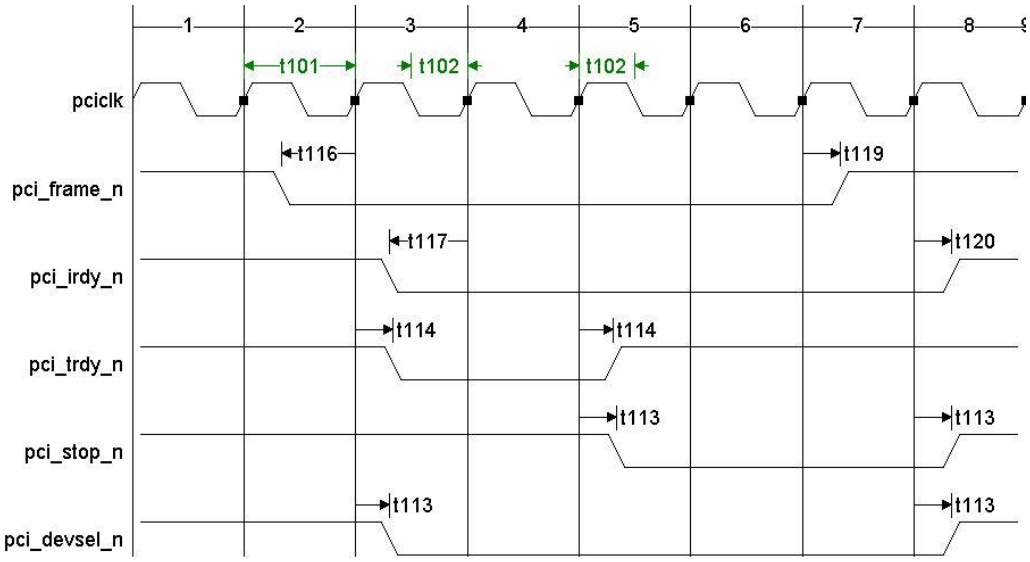


Figure A2-19　Pci_stop_n output

Appendix3 BM3803FMGRH application

BM3803 development board has functions below:

1. External 5V input;
2. BM3803, max frequency: 100MHz;
3. FLASH;
4. SRAM;
5. SDRAM;
6. 2-UART and DU;
7. CPCI interface compatible with PCIMG 2.0 R3.0;
8. GPIO interface.

Block Diagram of BM3803 board:

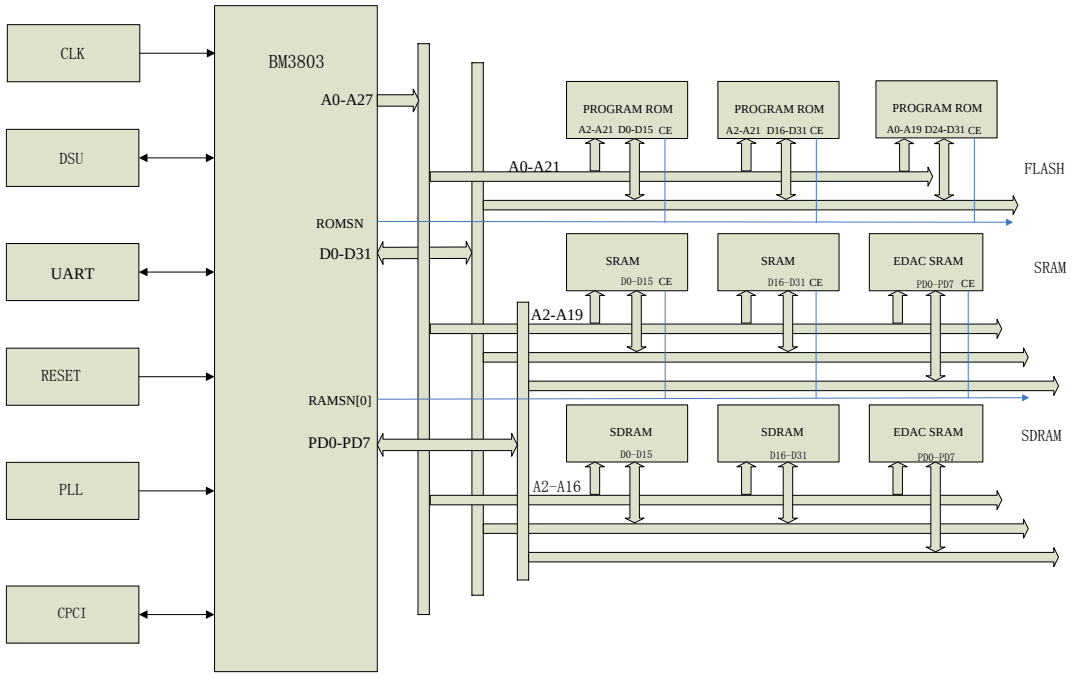


Figure A3-1 Block Diagram of BM3803 board

Description as follows:

1. BM3803 board supplies 3.3V and 1.8V power that provides to CPU and memory.
2. The CPU clock is driven either directly by an external oscillator or by the internal PLL:

- When BYPASS is driven high, the CPU clock is driven directly by the external oscillator: $f_{cpuclk} = f_{clk}$ =10MHz
- When BYPASS is driven low, the CPU clock is driven by the internal PLL :

$$f_{cpuclk} = f_{clk} * M$$

Dial switch is used to select the value of M:

Table A3-1 Dial switch is used to select the value of M

M	M[3]	M[2]	M[1]	M[0]
1	0	0	0	1

2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Note: “1” means connecting high level, “0” means connecting low level;

The M-value is best 10, at this point; the CPU frequency is 100MHz.

3. DU interface

The switch determines the two work states of CPU:

- reset mode

When DUEN and DUBRE are both driven low, the processor runs the programmer in FLASH after the end of the system reset.

- debug mode

When DUEN and DUBRE are both driven high, the processor runs the first four instructions to enter debug mode after the end of the system reset.

4. PROM

PROM interface is divided into two Banks, each Bank size is 256Mbyte, supporting general Flash chip, e.g. AM29LV160D.

5. SRAM

SRAM interface is divided into five Banks; each Bank size is programmable, supporting general SRAM chip, e.g. CY7C1041.

6. SDRAM

SDRAM interface is divided into two Banks; each Bank size is 256Mbyte,

supporting general SDRAM chip, e.g. HY57V561620.

7. CPCI interface

The PCI interface supports HOST mode and GUEST mode compatible with PCIMG 2.0 R3.0.

8. GPIO interface

GPIO interface consists in a 32-bit wide I/O port with alternate facilities. The interface is based on bi-directional I/O ports. The port is split in two parts, with the lower 16-bits accessible by the parallel IO pads and the upper 16-bits via the data bus.

9. UART interface

BM3803 implements three UARTS. The two Uarts are defined as alternate functions of the GPIO.

Appendix4 Registers Description

Register legend

Bit Number	31	30	29	28	27	26	25	24	23	...	...	...	...	9	8	7	6	5	4	3	2	1	0
Field name	bit	reserved			field										field				field				
Access type	r = read access				r/w = read and write access										w = write access				r/w = read and write access				
Default value after reset	0	0	1	1	x = undefined or non affected by reset										x	x	0	x	0	1	0	0	1

Integer Unit Registers

Processor State Register- PSR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMPL				VER				N	Z	V	C	reserved						EC	EF	PIL				S	PS	ET	reserved		CWP		
r				r				r/ w				r						r	r/w	r/ w				r/ w			r		r/w		
0				1	0	0	0	xxxx				0	0	1	0	0	0	0	x	xxxx				1	x	0	00		xxx		

impl: Implementation or class of implementations of the architecture.

ver: Identify one or more particular implementations or is a readable and writable state field whose properties are implementation-dependent.

n: Indicates whether the ALU result was negative for the last instruction modifying icc field.1 = negative; 0 = not negative.

z: Indicates whether the ALU result was zero for the last instruction modifying icc field. 1 = zero; 0 = not zero.

v: Indicates whether the ALU result was within the range of (was representable in) 32-bit 2's complement notation for the last instruction that modified the icc field. 1 = overflow; 0 = no overflow.

c: Indicates whether a 2's complement carry out (or borrow) occurred for the last instruction that modified the icc field. Carry is set on addition if there is a carry out of bit 31. Carry is set on subtraction if there is borrow into bit 31. 1 = carry; 0 = no carry.

ec: Determines whether the implementation-dependent coprocessor is enabled. If disabled, a coprocessor instruction will trap. 1 = enabled, 0 = disabled. If an implementation does not support a coprocessor in hardware, PSR.EC should always read as 0 and writes to it should be ignored.

ef: Determines whether the FPU is enabled. If disabled, a floating-point instruction will trap. 1 = enabled, 0 = disabled. If an implementation does not support hardware FPU, PSR.EF should always read as 0 and writes to it should be ignored.

pil: Identify the interrupt level above which the processor will accept an interrupt.

s: Determines whether the processor is in supervisor or user mode. 1 = supervisor mode, 0 = user mode.

ps: Contains the value of the S bit at the time of the most recent trap.

et: Determines whether traps are enabled. A trap automatically resets ET to 0. When ET=0, an interrupt request is ignored and an exception trap causes the IU to halt execution, which typically results in a reset trap that resumes execution at address 0. 1 = traps enabled, 0 = traps disabled.

cwp: Comprise the current window pointer, a counter that identifies the current window into the r registers. The hardware decrements the CWP on traps and SAVE instructions, and increments it on RESTORE and RETT instructions (modulo NWINDOWS).

Window Invalid Mask - WIM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																								Windows							
																								7	6	5	4	3	2	1	0
r																								r/w							
0																								x							

windows: Indicated whether the window is a ‘valid’ or an ‘invalid’ one. ‘0’: valid; ‘1’: invalid.

The WIM can be read by the privileged RDWIM instruction and written by the WRWIM instruction.

Trap Base Address - TBR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TBA																				TT								ZERO			
r/w																				r								r			
x																				x								0000			

tba: Trap Base Address. This field contains the most-significant 20 bits of the trap table address.

tt: Trap Type. This eight-bit field is written by the hardware when a trap occurs and retains its value until the next trap. It provides an offset into the trap table.

Y Register - Y

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Y																															
r/w																															
x																															

The Y register can be read and written with the RDY and WRY instructions.

Program Counters - PC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC																															
r/w																															
0																															

The 32-bit PC contains the address of the instruction currently being executed by the IU. When a trap occurs, the PC address is saved in the local register (%l1). When returning from trap, %l1 value is copied back to PC.

New Program Counters - nPC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
nPC																															
r/w																															
0x4																															

The nPC holds the address of the next instruction to be executed (assuming a trap does not occur).When a trap occurs, the nPC address is saved in the local register (%l2). When returning from trap, %l2 value is copied back to nPC.

Register File Protection Control Register, %asr16

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB			reserved						TCB							CNT															
r/w			r						r/w							r															
xxx			0						x							x															

cb: EDAC control bits, “000/001/011/100/110/111”: count the error; “010”: clear the error; “101”: make the data or check bits error.

tcb: make the check bits error.

cnt: Error counters. Incremented for each corrected error.

This register is accessed using the RDASR/WRASR instructions.

Table 4-1Window Registers

Type	Name	Definition
in	i7	return address
	i6	frame pointer
	i5-i0	incoming parameter register i5-i0
local	l7-l3	local register l7-l3
	l2	nPC (for RETT)
	l1	PC (for RETT)
	l0	local register 0
out	o7	temp
	o6	stack pointer
	o5-o0	outgoing parameter register o5-o0
global	g7-g0	global register g7-g0, global register 0 - always 0x00000000

Register File Protection Control Register, %asr17

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DCB																															
r/w																															
x																															

dcb: make the data bits error.

Watch Point Address Registers, (%asr24, %asr26, %asr28, %asr30)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WADDR																														reserved	IF
r/w																														r	r/w
x																														0	0

waddr: Defines the addresses range to be watched.

if: Enable hit generation on instruction fetch. 0 = disabled; 1 = enabled.

These registers are accessed using the RDASR/WRASR instructions

Watch Point Mask registers, (%asr25, %asr27, %asr29, %asr31)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WMASK																														DL	DS
r/w																														r/w	r/w
x																														0	0

wmask: Defines which bits are to be compared to waddr. 0 = comparison disabled; 1 = comparison enabled.

dl: Enable hit generation on data load. 0 = disabled; 1 = enabled.

ds: Enable hit generation on data store. 0 = disabled; 1 = enabled.

These registers are accessed using the RDASR/WRASR instructions

Floating Point Unit

FPU Status register - FSR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RD		reserved		TEM					NS	reseerved		VER			FTT			reserved		FCC		AEXC					CEXC				
r/w		r		r/w					r	r		r			r			r		r/w		r/w					r/w				
xx		00		xxxxx					0	00		000			xxx			00		00		xxxxx					xxxxx				

rd: Rounding Direction.

Defines the rounding direction used by the BM3803 FPU during a floating-point arithmetic operation.

tem: Trap Enable Mask.

tem field enables traps caused by FPops. These bits are ANDed with the bits of the cexc (current exception field) to determine whether to force a floating-point exception to IU. All trap enable fields correspond to the similarly named bit in the cexc field.

0 = trap disabled

1 = trap enabled

ns: Causes the FPU to produce implementation-defined results that may not correspond to ANSI/IEEE Standard 754-1985. For instance, to obtain higher performance, implementations may convert a subnormal floating-point operand or result to zero when NS is set.

ver: Identify one or more particular implementations of the FPU architecture. For each SPARC IU implementation there may be one or more FPU implementations, or none. This field identifies the particular FPU implementation present.

ftt: Identify floating-point exception trap types. when floating point exception occurs, the ftt field encodes the type of floating-point exception until an STFSR or another FPop is executed.

fcc: Contain the FPU condition codes. These bits are updated by floating-point compare instructions (FCMP and FCMPE). They are read and written by the STFSR and LDFSR instructions, respectively. FBfcc bases its control transfer on this field.

aexc: Accumulate IEEE floating-point exceptions while fp_exception traps are disabled using the TEM field. After an FPop completes, the TEM and cexc fields are logically anded together. If the result is nonzero, an fp_exception trap is generated; otherwise, the new cexc field is or'd into the aexc field. Thus, while traps are masked, exceptions are accumulated in the aexc field.

cexc: Indicate that one or more IEEE floating-point exceptions were generated by the most recently executed FPop instruction. The absence of an exception causes the corresponding bit to be cleared.

Table 4-2 FSR Trap Type（FTT）

FTT	Trap Type	Description
0	None	No trap
1	IEEE_exception	An IEEE_754_exception floating-point trap type indicates that a floating-point exception occurred that conforms to the ANSI/IEEE Standard 754-1985. The exception type is encoded in the cexc field.
4	sequence_error	A sequence_error indicates one of three abnormal error conditions in the FPU, all caused by erroneous supervisor software: - An attempt was made to execute a floating-point instruction when the FPU was not able to accept one. This type of sequence_error arises from a logic error in supervisor software that has caused a previous floating-point trap to be incompletely serviced (for example, the floating-point queue was not emptied after a previous floating-point exception). - An attempt was made to execute a STDFQ instruction when the floating-point deferred-trap queue (FQ) was empty, that is, when FSR.qne = 0. (Note that generation of sequence_error is recommended, but not required in this case).

Table 4-3 FSR Floating Point Condition Code（FCC）

FCC Description

FCC	Description
0	f rs1 = f rs2
1	f rs1 < f rs2
2	f rs1 > f rs2
3	f rs1 ? f rs2 indicates an unordered relation, which is true if either f rs1 or f rs2 is a signaling NaN or quiet NaN

Note: f_{rs1} and f_{rs2} correspond to the single or double values in the f registers specified by an instruction's rs1 and rs2 fields. Note that fcc is unchanged if FCMP or FCMPE generates an IEEE_exception trap.

The current and accrued exception fields and the trap enable mask assume the following definitions of the floating-point exception conditions.

Table 4-4 Floating Point Exception Fields

Aexc Mnemonic	Cexc Mnemonic	Name	Description
nva	nvc	Invalid	An operand is improper for the operation to be performed. 1 = invalid operand, 0 = valid operand(s). Examples : 0 ÷0 and ∞-∞ are invalid.
ofa	ofc	Overflow	The rounded result would be larger in magnitude than the largest normalized number in the specified format. 1 = overflow, 0 = no overflow.
ufa	ufc	Underflow	The rounded result is inexact and would be smaller in magnitude than the smallest normalized number in the indicated format. 1 = underflow, 0 = no underflow. Underflow is never indicated when the correct unrounded result is zero. if UFM=0 : The ufc and ufa bits will be set if the correct unrounded result of an operation is less in magnitude than the smallest normalized number and the correctly-rounded result is inexact. These bits will be set if the correct unrounded result is less than the smallest normalized number, but the correct rounded result is the smallest normalized number. nxc and nxa are always set as well. if UFM=1 : An IEEE_exception trap will occur if the correct unrounded result of an operation would be smaller than the smallest normalized number. A trap will occur if the correct unrounded result would be smaller than the smallest normalized number, but the correct rounded result would be the smallest normalized number.
dza	dzc	Div_by_zero	X÷0, where X is subnormal or normalized. Note that 0 ÷0 does not set the dzc bit. 1 = division-by-zero, 0 = no division-by-zero.
nxa	nxc	Inexact	1 = inexact result, 0 = exact result.

Memory Interface Registers

Memory Configuration Register—MCFG1（0x80000000）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved			IOWDH		BRDY	BEXC	reserved	IOWS				IOEN	reserced		PRWWS					PRWEN	reserved		PRWDH	reserved			PRRWS					
r			r/w		r/w	r/w	r	r/w				r/w	r		r/w					r/w	r		r/w	r			r/w					
000			xx		0	0	0	1111				0	00		11111					0	0		xx	000			11111					

iowdh: I/O bus width.

Defines the data with of the I/O area (“00”=8, “01”=16, “10”=32).

brdy: Bus ready enable for I/O area

bexc: Bus error enable.

iows: I/O waitstates.

Defines the number of waitstates during I/O accesses (“0000”=0, “0001”=1, “0010”=2, ..., “1111”=15).

ioen: I/O read and write enable.

prwws: Prom write waitstates.

Defines the number of waitstates during prom write cycles (“00000”=0, “00001”=2, ... “11111”=31).

prwen: Prom write enable.

If set, enables write cycles to the prom area.

prwdh: Prom width.

Defines the data with of the prom area (“00”=8, “01”=16, “1x”=32).

prrws: Prom read waitstates.

Defines the number of waitstates during prom read cycles (“00000”=0, “00001”=2, ... “11111”=31).

During power-up, the prom width (bits [9:8]) are set with value on PIO[1:0] inputs. The prom waitstates fields are set to 15 (maximum).

Note: The last 25% of the prom bank size are used to store EDAC checksums when EDAC is enabled in 8 bit mode.

Memory Configuration Register 2－MCFG2（0x80000004）

31	30	29 28 27	26	25 24 23	22 21	20 19	18 17 16 15	14	13	12 11 10 9	8	7	6	5 4	3 2 1 0
SDREF	TRP	TRFC	SDRCAS	SDRBS	SDRCLS	SDRCMD	SRWWS	SDREN	SRDIS	SRBS	reserved	BRDY	RMW	SRWDH	SRRWS
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r	r/w	r/w	r/w	r/w
0	1	1 1 1	1	0 0 0	1 0	0 0	1 1 1 1	0	0	x x x x	0	x	x x	x x	1 1 1 1

sdref: SDRAM refresh.

If set, the SDRAM refresh will be enabled.

trp: SDRAM t_{RP} timing.

t_{RP} will be equal to 2 or 3 system clocks (0/1).

trfc: SDRAM t_{RFC} timing.

t_{RFC} will be equal to (3 + field-value) system clocks.

sdrcas: SDRAM CAS delay.

Selects 2 or 3 cycles CAS delay (0/1). When changed, a LOAD-COMMAND - REGISTER command must be issued at the same time. Also sets RAS/CAS delay (tRCD).

sdrbs: SDRAM banks size.

Defines the banks size for SDRAM chip selects: “000”=4 Mbyte,
“001”=8 Mbyte,
“010”=16 Mbyte
....
“111”=512 Mbyte.

sdrcls: SDRAM column size.

“00”=256 when sdrbs = “111”
“01”=512 when sdrbs = “111”
“10”=1024 when sdrbs = “111”
“11”=4096 when sdrbs = “111”
“2048 otherwise“

sdrcmd:SDRAM command.

Writing a non-zero value will generate an SDRAM command:
“01”=PRECHARGE,
“10”=AUTO-REFRESH,
“11”=LOAD-COMMAND-REGISTER.
The field is reset after command has been executed.

srwws: SRAM write waitstates.

Defines the number of waitstates during SRAM write cycles (“0000”=0,“0001”=1,“0010”=2,…, “1111”=15).

sdren: SDRAM enable.

If set, the SDRAM controller will be enabled.

srdis: SRAM disable.

If set, the SRAM controller will be disabled.

srbs: SRAM bank size.

Defines the size of each ram bank (“0000”=8 Kbyte, “0001”=16 Kbyte... “1111”=256 Mbyte).

brdy: Bus ready enable. For SRAM BANK[4].

rmw: Read-modify-write.

Enable read-modify-write cycles on sub-word writes to 16- and 32-bit areas with common write strobe (no byte write strobe).

srwdh: SRAM bus width.

Defines the data with of the SRAMarea ("00"=8, "01"=16, "1X"= 32).

srrws: SRAM read waitstates.

Defines the number of waitstates during SRAM read cycles ("0000"=0, "0001"=1, "0010"=2,…,"1111"=15).

Memory Configuration Register 3 - MCFG3（0x80000008）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved					SDRRV															reserved											
r					r/w															r											
0					x															0											

sdrrv: SDRAM refresh counter reload value.

The period between each AUTO-REFRESH command is calculated as follows:

tREFRESH = ((reload value) + 1) / SYSCLK

MECFG 1（0x80000100）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SR1EEN	SR2EEN	SR3EEN	SR4EEN	SR5EEN	ROMEEN	re			EWB2	ERB	EWB1	SDEEN	EWB0	PRBS				re				TCBAREA		re	TCB							
r/w	r/w	r/w	r/w	r/w	r/w	r			r/w	r/w	r/w	r/w	r/w	r/w				r				r/w		r	r/w						
0	0	0	0	0	0	0			1	0	0	0	0	0				0				11		0	x						

sr1een: SRAM bank1 EDAC enable

sr2een: SRAM bank2 EDAC enable

sr3een: SRAM bank3 EDAC enable

sr4een: SRAM bank4 EDAC enable

sr5een: SRAM bank5 EDAC enable

romeen: PROM EDAC enable

ewb2: EDAC write bypass

erb: EDAC read bupass

ewb1: EDAC write bypass enable

sdeen: SDRAM EDAC enable

ewb0: EDAC write bypass enable

prba: PROM bank size （0000 = 8 Kbyte, 0001 = 16 Kbyte … 1111 = 256 Mbyte）

tcbarea: TCB area

tcb: test check bit

MECFG 2（0x80000104）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DR																															
r/w																															
x																															

DR: Data Reversal data bit reverse enable

MECFG 3（0x80000108）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																									PR						
r/w																									r/w						
0																									x						

PR: Pardata Reversal parity bit reverse enable

MECFG 4（0x8000010C）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MCTRL																															
r																															
0																														1	1

mctrl:edition

Write Protection Register 1 – WPR1（0x8000001C）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	BP	TAG															MASK														
r/w	r/w	r/w															r/w														
x	x	x															x														

en: Enable.

If set, enables the write protect unit

bp: Block protect

If set, selects block protect mode

tag: Address tag

This field is compared against address(29:15)

mask: Address mask

This field contains the address mask

Write Protection Register 2 - WPR2（0x80000020）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	BP	TAG															MASK														
r/w	r/w	r/w															r/w														
x	x	x															x														

en: Enable.

If set, enables the write protect unit

bp: Block protect

If set, selects block protect mode

tag: Address tag

This field is compared against address(29:15)

mask: Address mask

This field contains the address mask

System Registers

Product Configuration Register – PCR（0x80000024）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MMU	DSU	SDRCTRL	WTPNB			IMAC	NWINDOWS					ICSZ			ILSZ		DCSZ			DLSZ		DIVINST	MULINST	WDOG	MEMSTAT	FPU		PCI		WPRT	
r	r	r	r			r	r					r			r		r			r		r	r	r	r	r		r		r	
0	1	1	1	0	0	1	0	0	1	1	1	0	1	1	1	1	0	1	1	1	0	1	1	1	1	0	1	1	1	0	1

mmu: ‘0’, no MMU

du: Debug unit

‘0’=disabled

‘1’=present

sdrctrl: SDRAM controller present

‘0’=disabled

‘1’=present

wtpnt: Number of implemented watchpoints (0 - 4)

imac: UMAC/SMAC instruction implemented

nwindows: Number of register windows.

The implemented number of SPARC register windows - 1

icsz: Instruction cache size.

The size (in Kbytes) of the instruction cache.

ilsz: Instruction cache line size.

The line size (in 32-bit words) of each line.

dcsz: Data cache size.

The size (in kbytes) of the data cache.

dlsz: Data cache line size.

The line size (in 32-bit words) of each line.

divinst: UDIV/SDIV instruction implemented

mulinst: UMUL/SMUL instruction implemented

wdog: ‘1’,watchdog

memstat: Memory status and failing address register present

fpu: FPU type

pci: PCI core type

wprt: Write protection type

AHB Fail Address Register – FAILAR（0x8000000C）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FA																															
r																															
x																															

fa: Failing Address

This field contents the address of the access that triggered an error response. This register is updated each time an error occurs in the internal bus.

AHB Fail Status Register – FAILSR（0x80000010）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																							NE	RW	HMAST				HSIZE		
r																							r/w	r	r				r		
0																							0	x	xxxx				xxx		

ne: error state

ne: New error.

Set when a new error occurred.

After an error, this bit has to be reset by software

rw: Read/Write.

This bit is set if the failed access was a read cycle, otherwise it is cleared.

hmaster: AHB master.

This field contains the HMASTER[3:0] of the failed access.

hsize: Transfer Size.

This filed contains the HSIZE[2:0] of the failed transfer.

Caches Register

Cache Control Register – CCR（0x80000014）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DRP		IRP		ISTS		DSTS		DS	FD	FI	R				IB	IP	DP	R								DF	IF	DCS		ICS	
r		r		r		r		r/w	w	w	r				r/w	r	r	r								r/w	r/w	r/w		r/w	
11		11		11		01		0	0	0	0000				0	0	0	0								x	x	00		00	

drp: Data cache replacement policy:‘11’ - Least Recently Used (LRU)

irp: Instruction cache replacement policy:‘11’ - Least Recently Used (LRU)

ists: Instruction cache associativity.
Number of sets in the instruction cache‘11’ - 4 way associative

dsts: Data cache associativity.
Number of sets in the data cache‘01’ - 2 way associative

ds: Data cache snoop enable
If set, will enable data cache snooping.

fd: Flush data cache
If set, will flush the instruction cache. Always reads as zero.

fi: Flush Instruction cache
If set, will flush the instruction cache. Always reads as zero.

ib: Instruction burst fetch
This bit enables burst fill during instruction fetch.

ip : Instruction cache flush pending
This bit is set when an instruction cache flush operation is in progress.

dp: Data cache flush pending
This bit is set when a data cache flush operation is in progress.

df: Data Cache Freeze on Interrupt
If set, the data cache will automatically be frozen when an asynchronous interrupt is taken.

if: Instruction Cache Freeze on Interrupt
If set, the instruction cache will automatically be frozen when an asynchronous interrupt is taken.

dcs: Data Cache state
Set and Indicates the current data cache state
‘X0’ = disabled
‘01’ = frozen
‘11’ = enabled

ics: Instruction Cache state
Set and Indicates the current instruction cache state
‘X0’ = disabled
‘01’ = frozen
‘11’ = enabled.

CCR1（0x80000110）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VER			reserved															CMEEN			CNTRST	CPSEL		reserved				EPOS				
r			r															r/w			r/w	r/w		r				r/w			
0	0	1	0															0			0	0		0				0			

ver: Cache version

cmeen: Cache make error enable bit

cntrst: Cache parity counter clear bit

cpsel: 00-instruction cache data,01-instruction cache tag,10-data cache data,11-data cache tag

epos: High 4 bits of Cache parity make error

CCR2（0x80000114）

When making the wrong Cache enabled, and the target area for the Data zone:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EPOS																															
r/w																															
0																															

epos: make error low 32 bit

When making the wrong Cache enabled, and the target zone region Tag:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOCK	TAG																			VALID				保留							
r/w	r/w																			r/w				r							
0	0																			0				0							

lock: Cache lock bit

tag: Cache tag data bit

valid: invalid

CCR3（0x80000118）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DTAGCNT						DDATACNT										ITAGCNT						IDATACNT									
r						r										r						r									
0						0										0						0									

dtagcnt: Dcache tag error counter

ddatacnt: Dcache data error counter

itagcnt: Icache tag error counter

idatacnt: Icache data error counter

Timers

Timer 0 Counter Register – TIMC0（0x80000040）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								TIMERCNT0																							
r								r/w																							
0								x																							

timercnt0:Timer0 counter value

A read access gives the decounting value of the scalar.

Timer 0 Reload Register – TIMR0（0x80000044）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								TIMERLOAD0																							
r								r/w																							
0								x																							

timerload0:Timer 0 reload value

A write access programs the reload value of Timer 0 counter.

Timer 0 Control Register – TIMCTR0（0x80000048）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																											WTDEN	reserved	LD0	RL0	EN0
r																											r/w	r	r/w	r/w	r/w
0																											x	0	x	x	0

wtden:watchdog enable

ld0: Load counter

When written with ‘one’, will load the timer reload register into the timer counter register. Always reads as a ‘zero’.

rl0: Reload counter

If RL is set, then the counter will automatically be reloaded with the reload value after each underflow.

en0: Enable counter

Enables the timer when set.

Watchdog Register – WDG（0x8000004C）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								WDCNT																							
r								r/w																							
0								FFFFFF																							

wdcnt: Watchdog counter.

Timer 1 Counter Register – TIMC1（0x80000050）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								TIMERCNT1																							
r								r/w																							
0								x																							

timercnt1: Timer 1 counter value

A read access gives the decounting value of the scalar.

Timer 1 Reload Register – TIMR1（0x80000054）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								TIMERLOAD1																							
r								r/w																							
0								x																							

timerload1: Timer 1reload value

A write access programs the reload value of Timer 1 counter.

Timer 1 Control Register – TIMCTR1（0x80000058）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																													LD1	RL1	EN1
r																													r/w	r/w	r/w
0																													x	x	0

ld1: Load counter

When written with ‘one’, will load the timer reload register into the timer counter register. Always reads as a ‘zero’.

rl1: Reload counter

If RL is set, then the counter will automatically be reloaded with the reload value after each underflow.

en1: Enable timer

Enables the timer when set.

Prescalar Counter Register – SCAC（0x80000060）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																						SCALERLOAD									
r																						r/w									
0																						0x3FF									

scalarload: prescalar counter value

A read access gives the decounting value of the prescalar.

Prescalar Reload Register – SCAR（0x80000064）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																						SCALERLOAD									
r																						r/w									
0																						0x3FF									

scalarload: Prescalar reload value

UARTs Registers

UART 1 Data Register - UAD1（0x80000070）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																								RTD							
r																								r/w							
0																								x							

rtd: Received or Transmitted Data of UART1

Rtd field has 2 meanings:

A write access enables the sending of the written 8-bit data on UART 1.

A read access provides the received 8-bit data on UART1.

UART 1 Status Register - UAS1（0x80000074）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																									FE	PE	OV	BR	TH	TS	DR
r																									r/w	r/w	r/w	r/w	r	r	r
0																									0	0	0	0	1	1	0

fe: Framing error

Indicate that a framing error was detected.

pe: Parity error

Indicate that a parity error was detected.

ov: Overrun

Indicates that one or more character have been lost due to overrun.

br: Break received

Indicate that a BREAK has been received.

th: Transmitter hold register empty

Indicate that the transmitter hold register is empty.

ts: Transmitter shift register empty

Indicate that the transmitter shift register is empty.

dr: Data ready

Indicate that new data is available in the receiver holding register.

UART 1 Control Register - UAC1（0x80000078）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																							EC	LB	FL	PE	PS	TI	RI	TE	RE
r																							r/tw	r/tw	r/tw	r/tw	r/tw	r/tw	r/tw	r/tw	r/tw
0																							0	0	0	x	x	x	x	0	0

ec: External Clock

if set, the UART scalar will be clocked by PIO[3]

lb: Loop back

If set, loop back mode will be enabled.

fl: Flow control

If set, enables flow control using ctsn/rtsn.

pe: Parity enable

If set, enables parity generation and checking.

ps: Parity select

Selects parity polarity

“0” = even parity

“1” = odd parity

ti: Transmitter interrupt enable

If set, enable generation of transmitter interrupt.

ri: Receiver interrupt enable

If set, enable generation of receiver interrupt.

te: Transmitter enable

If set, enable the transmitter.

re: Receiver enable

If set, enable the receiver.

UART 1 Scalar Register - UASCA1（0x8000007C）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																				SRV											
r																				r/w											
0																				x											

SRV:scalar reload value

UART 2 Data Register - UAD2（0x80000080）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																								RTD							
r																								r/w							
0																								x							

rtd:　Received or Transmitted Data of UART2

Rtd field has 2 meanings :

A write access enables the sending of the written 8-bit data on UART 2.

A read access provides the received 8-bit data on UART2.

UART 2 Status Register - UAS2（0x80000084）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																									FE	PE	OV	BR	TH	TS	DR
r																									r/w	r/w	r/w	r/w	r	r	r
0																									0	0	0	0	1	1	0

fe: Framing error

Indicate that a framing error was detected.

pe: Parity error

Indicate that a parity error was detected.

ov: Overrun

Indicates that one or more character have been lost due to overrun.

br: Break received

Indicate that a BREAK has been received.

th: Transmitter hold register empty

Indicate that the transmitter hold register is empty.

ts: Transmitter shift register empty

Indicate that the transmitter shift register is empty.

dr: Data ready

Indicate that new data is available in the receiver holding register.

UART 2 Control Register - UAC2（0x80000088）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																							EC	LB	FL	PE	PS	TI	RI	TE	RE
r																							r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
0																							0	0	0	x	x	x	x	0	0

ec: External Clock

If set, the UART scalar will be clocked by PIO[3]

lb: Loop back

If set, loop back mode will be enabled.

fl: Flow control

If set, enables flow control using ctsn/rtsn.

pe: Parity enable

If set, enables parity generation and checking.

ps: Parity select

Selects parity polarity

“0” = even parity

“1” = odd parity

ti:Transmitter interrupt enable

If set, enable generation of transmitter interrupt.

ri: Receiver interrupt enable

If set, enable generation of receiver interrupt.

te: Transmitter enable

If set, enable the transmitter.

re: Receiver enable

If set, enable the receiver.

UART 2 Scalar Register - UASCA2（0x8000008C）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																				**SRV**											
r																				**r/w**											
0																				**x**											

SRV:scalar reload value.

UART3 receive/send data register（0x800000E0）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																								RTD							
r																								r/w							
0																								x							

rtd: Received or Transmitted Data of UART3

Rtd field has 2 meanings :

A write access enables the sending of the written 8-bit data on UART 3.

A read access provides the received 8-bit data on UART3.

UART 3 Status Register – UAS3（0x800000E4）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																									FE	PE	OV	BR	TH	TS	DR
r																									r/w	r/w	r/w	r/w	r	r	r
0																									0	0	0	0	1	1	0

fe: Framing error

Indicate that a framing error was detected.

pe: Parity error

Indicate that a parity error was detected.

ov: Overrun

Indicate that one or more character have been lost due to overrun.

br: Break received

Indicate that a BREAK has been received.

th: Transmitter hold register empty

Indicate that the transmitter hold register is empty.

ts: Transmitter shift register empty

Indicate that the transmitter shift register is empty.

dr: Data ready

Indicate that new data is available in the receiver holding register.

UART 3 Control Register – UAC3（0x800000E8）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																							EC	LB	FL	PE	PS	TI	RI	TE	RE
r																							r/fw	r/fw	r/fw	r/fw	r/fw	r/fw	r/fw	r/fw	r/fw
0																							0	0	0	x	x	x	x	0	0

ec: External Clock

If set, the UART scalar will be clocked by PIO[3]

lb: Loop back

If set, loop back mode will be enabled.

fl: Flow control

If set, enables flow control using ctsn/rtsn.

pe: Parity enable

If set, enables parity generation and checking.

ps: Parity select

Selects parity polarity

“0” = even parity

“1” = odd parity

ti: Transmitter interrupt enable

If set, enable generation of transmitter interrupt.

ri: Receiver interrupt enable

If set, enable generation of receiver interrupt.

te: Transmitter enable

If set, enable the transmitter.

re: Receiver enable

If set, enable the receiver.

UART 3 Scalar Register – UASCA3（0x800000EC）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																				**SRV**											
r																				**r/w**											
0																				**x**											

SRV:scalar reload value.

Interrupt Registers

Interrupt Mask and Priority Register – ITMP（0x80000090）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILEVEL															Reserved	IMASK															Reserved
r/w															r	r/w															r
x															x	0															x

ilevel[15:1]: Interrupt level

Indicate whether an interrupt belongs to priority level 1 (ILEVEL[n]=1) or level 0 (ILEVEL[n]=0).

imask[15:1]: Interrupt mask

Indicate whether an interrupt is masked or enabled

‘0’ = masked

‘1’ = enabled

Interrupt Pending Register – ITP（0x80000094）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																IPEND															reserved
r																r/w															r
x																x															x

ipend[15:1]: Interrupt pending

Indicate whether an interrupt is pending

“1” = interrupt pending

“0” = interrupt not pending

When the IU acknowledges the interrupt, the corresponding pending bit is automatically cleared.

Interrupt Force Register – ITF（0x80000098）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																IFORCE															reserved
r																r/w															r
x																x															x

iforce[15:1]: Interrupt force

Indicate whether an interrupt is being forced

"1" = interrupt forced

"0" = interrupt not forced

Interrupt can be forced by setting a bit in the interrupt force register. IU acknowledgement will clear the force bit.

Interrupt Clear Register – ITC（0x8000009C）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																ICLEAR															reserved
r																w															r
x																x															x

iclear[15:1]: Interrupt clear

If written with a '1', clear the corresponding bit(s) in the interrupt pending register.

A read returns zero.

General Purpose Interface Registers

I/O Port Data Register – IODAT（0x800000A0）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEDDAT								LOWDAT								IODATA															
r/w								r/w								r/w															
x								x								x															

meddat:connect to the data bus D[15:8]

lowdat:connect to the data bus D[7:0]

iodata:I/O port data

When read, returns the current value of the I/O port;

When written, value is driven on the I/O port signals (if enabled as Output)

I/O Port Direction Register – IODIR（0x800000A4）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved														MEDDIR	LOWDIR	IODIR[15：0]															
r														r/w	r/w	r/w															
0														0	0	0															

meddir: Defines the direction of D[15:8]

lowddir: Defines the direction of D[7:0]

iodir: I/O port direction

Define the direction of I/O ports[15:0]

‘1’ = output

‘0’ = input

I/O Port Interrupt Register（0x800000A8）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	LE	PL	ISEL					EN	LE	PL	ISEL					EN	LE	PL	ISEL					EN	LE	PL	ISEL				
r/w	r/w	r/w	r/w					r/w	r/w	r/w	r/w					r/w	r/w	r/w	r/w					r/w	r/w	r/w	r/w				
0	x	x	x					0	x	x	x					0	x	x	x					0	x	x	x				

en: Enable.

If set, the corresponding interrupt will be enabled, otherwise it will be masked.

le: Level/edge triggered.

If set, the interrupt will be edge-triggered, otherwise level sensitive.

pl: Polarity

If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge).

isel:I/O port select.

The value of this field defines which I/O port (0 - 31) should generate parallel I/O port interrupt .

PCI related registers

PCI bus reset register(PCI_RESET)（0x800000D0）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																												RESET			
r																												w			
x																												xxxx			

reset: PCI interface reset

PCI bus DMA write operation start address register(WDMA_PCI_ADDR)
（0xC0000000）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR																															
r/w																															
0																															

addr: DMA write operation on PCI bus target register buffer address

AHB bus DMA write operation start address register(WDMA_AHB_ADDR)
（0xC0000004）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR																															
r/w																															
0																															

addr: DMA write operation on PCI bus target register buffer address

DMA write operation control register(WDMA_CONTROL)（0xC0000008）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRANSFER SIZE																								COMMAND				PCI INTERRUPT	reserved	STOP TRANSFER	START TRANSFER
r/w																								r/w				r/w	r	r/w	r/w
0																								0000				0	0	0	0

transfer size: DMA transfer bytes,range from 1-0xFFFFFF

command: Transfer command on PCI bus when DMA write operation

PCI interrupt: Interrupt request on PCI bus when DMA write operation finished in simple bridge

stop transfer: force exit when dma write operation

start transfer: start DMA write operation

PCI bus DMA read operation start address register(RDMA_PCI_ADDR)（0xC0000020）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR																															
r/w																															
0																															

addr: DMA read operation on PCI bus target buffer address

AHB bus DMA read operation start address register(RDMA_AHB_ADDR)（0xC0000024）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR																															
r/w																															
0																															

addr: DMA read operation on AHB bus target buffer address

DMA read operation control register(RDMA_CONTROL)（0xC0000028）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRANSFER SIZE																								COMMAND				PCI INTERRUPT	reserved	STOP TRANSFER	START TRANSFER
r/w																								r/w				r/w	r	r/w	r/w
0																								0000				0	0	0	0

transfer size: DMA transfer bytes,range from 1-0xFFFFFF

command: Transfer command on PCI bus when DMA read operation

PCI interrupt: Interrupt request on PCI bus when DMA read operation finished in

simple bridge

stop transfer: Force exit when DMA read operation

start transfer: start DMA write read operation

CPU interrupt mask register（0xC0000040）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved			PCI SYSTEM ERROR	PCI INTERRUPT				reserved				PCI DOORBELL				reserved	AHB- PCI WINDOW			READ DMA				reserved	PCI- AHB WINDOW			WRITE DMA			
																	DISCARD	FETCH ERROR	POST ERROR	AHB ERROR	PARITY ERROR	ABORT	END		DISCARD	FETCH ERROR	POST ERROR	AHB ERROR	PARITY ERROR	ABORT	END
r/w			r/w	r\w				r/w				r/w				r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r	r/w	r/w	r/w	r/w	r/w	r/w	r/w
000			0	0000				0000				0000				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PCI system error: PCI system error on host bridge mode

PCI interrupt: PCI interrupt on host bridge mode

PCI DOORBELL: PCI DOORBELL bit on simple bridge mode

AHB-PCI window discard: AHB-PCI window discard

AHB-PCI window fetch error: AHB-PCI window fetch error

AHB-PCI window post error: AHB-PCI window post error

read DMA AHB error: Read DMA AHB error

read DMA parity error: Read DMA parity error

read DMA abort: Read DMA abort

read DMA end: Read DAM end

PCI-AHB window discard: PCI-AHB window discard

PCI-AHB window fetch error: PCI-AHB window fetch error

PCI-AHB window post error: PCI-AHB window post error

write DMA AHB error: Write DMA AHB error

write DMA parity error: Write DMA parity error

write DMA abort: Write DMA abort

write DMA end: Write DMA end

CPU interrupt state register（0xC0000044）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved			PCI SYSTEM ERROR	PCI INTERRUPT				reserved				PCI DOORBELL				reserved	AHB-PCI WINDOW			READ DMA				reserved	PCI-AHB WINDOW			WRITE DMA			
																	DISCARD	FETCH ERROR	POST ERROR	AHB ERROR	PARITY ERROR	ABORT	END		DISCARD	FETCH ERROR	POST ERROR	AHB ERROR	PARITY ERROR	ABORT	END
r			r/w					r				r/w				r	r/w							r	r/w						
000			000000					0000				0000				0	0							0	0						

PCI system error: PCI system error on host bridge mode
PCI interrupt: PCI interrupt on host bridge mode
PCI DOORBELL: PCI DOORBELL bit on simple bridge mode
AHB-PCI window discard: AHB-PCI window discard
AHB-PCI window fetch error: AHB-PCI window fetch error
AHB-PCI window post error: AHB-PCI window post error
read DMA AHB error: read DMA AHB error
read DMA parity error: read DMA parity error
read DMA abort: read DMA abort
read DMA end: read DMA end
PCI-AHB window discard: PCI-AHB window discard
PCI-AHB window fetch error: PCI-AHB window fetch error
PCI-AHB window post error: PCI-AHB window post error
write DMA AHB error: write DMA AHB error
write DMA parity error: write DMA parity error
write DMA abort: write DMA abort
write DMA end: write DMA end

CPU interrupt command register（0xC0000048）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rserved								CPU DOORBELL				reserved																			
r								w				r																			
0								0000				0																			

CPU DOORBELL: CPU DOORBELL bit on simple bridge

CPU state and version register（0xC000004C）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																TYPE				VERSION											
r																r				r											
0																xxxx				0x380											

type: PCI interface mode.1=32 bit simple bridge mode,0=32 bit host bridge mode.
version: PCI bridge version

PCI-AHB window non-prefetch range control register(PCIAHB_ADDR_NP)
（0xC0000070）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR																															EN
r/w																								r							r/w
0x800000																								0							1

ahb base address: PCI bus operation change address in simple bridge mode,PCI-AHB window base address in host bridge mode
windows enable: PCI bus operation change enable

PCI-AHB window prefetch range control register(PCIAHB_ADDR_PF)
（0xC0000074）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR																															EN
r/w																								r							r/w
0x400000																								0							1

ahb base address: PCI bus operation change address in simple bridge mode,PCI-AHB window base address in host bridge mode
windows enable: PCI bus operation change enable

PCI-AHB window timer discard register(PCIAHB_TIMER)（0xC0000078）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																				DISCARD TIMER											
r																				r/w											
0																				0x100											

discard timer: discard timer to prevent PCI bus lock

AHB-PCI window discard timer register(AHBPCI_TIMER)（0xC000007C）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																				DISCARD TIMER											
r																				r/w											
0																				0x100											

discard timer: discard timer to prevent PCI bus lock

PCI control register（0xC0000080）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																													MasterEN	MemoryEN	BridgePCIReady
r																													r	r	r
0																													x	x	x

MasterEN: Ability state of master device

MemoryEN: Memory address space enable

Bridge PCI Ready: On host bridge mode,before CPU access PCI bus configuration could wait the bit is set.After PCI reset the bit is cleared,after 2^{25} PCI clock cycles set to 1

PCI device and vendor ID register(PCI_DV)（0xC0000084）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DV																															
r/w																															
0x55551556																															

DV: PCI device and vendor information

PCI subsystem and subsystem vendor configure register(PCI_SUB)（0xC0000088）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SUBSYSTEM ID																SUBSYSTEM VENDOR ID															
r/w																r/w															
0																0															

subsystem id: PCI subsystem device information

subsystem vendor id: PCI subsystem vendor information

PCI classify and edition configuration register(PCI_CREV)（0xC000008C）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ClASS CODE																								REVISION ID							
r/w																								r/w							
0x040000																								0							

class code: PCI classified code

revision id: revision information

PCI arbiter state register(PCI_BROKEN)（0xC0000090）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																								STATUS							
r																								x							
0																								0							

status: master device status on PCI bus in host bridge mode

PCI-AHB window non-prefetch range config register(PCIAHB_SIZ_NP)
（0xC0000094）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIZ																															
r/w																															
0xFFFC00																								0							

siz: the size of non-prefetch space

PCI-AHB window prefetch range config register(PCIAHB_SIZ_PF)（0xC0000098）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIZ																															
r/w																															
0xFFFC00																								0							

siz: the size of prefetch space

PCI device and vendor Identification register(PCI config space 0x00)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DEVICE ID																VENDOR ID															
r																r															
0x5555																0x1556															

Device ID: Device ID,allocated by PCI device and Vendor ID Register（0xC0000084）

Vendor ID: Vendor ID,allocated by PCI device and Vendor ID Registe（0xC0000084）

PCI Status - Command Register(PCI config space 0x04)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STATUS																COMMAND															
DETECTED PARITY ERROR	SIGNALED SYSTEM ERROR	RECEIVED MASTER ABORT	RECEIVED TARGET ABORT	SINGALED TARGET ABORT	DEVSEL TIMING	MASTER DATA PARITY ERROR	FAST BACK-TO-BACK CAPABLE	reserved	reserved	reserved	INTERRUPT STATUS	reserved				reserved	INTERRUPT DISABLE	reserved	SERR ENABLE	reserved	PARITY ERROR RESPONSE	reserved	reserved	reserved	BUS MASTER	MEMORY SPACE	reserved				
r	r	r	r	r	r	r	r	r	r	r	r	r	r			r	r/w	r/w	r/w	r	r/w	r	r	r	r/w	r/w	r/w				
0	0	0	0	0	01	0	0	0	1	0	0	000				00000	0	0	0	0	0	0	0	0	0	0	0				

detected parity error: Parity error detected.

This bit must be set by the device whenever it detects a parity error

singaled system error: SERR asserted.

This bit must be set whenever the device asserts SERR*.

received master abort: received master abort

received target abort: received target abort

singaled target abort: singaled target abort

devesel timing: Devsel timing.

These bits encode the timing of DEVSEL*.

master data parity error: Master received/asserted PERR
fast back-to-back enable: Target supports fast back to back
interrupt status: interrupt status
interrupt disable: interrupt disable
serr enable: Enable SERR driver
parity error response: device's response to parity errors.
bus master: device's ability to act as a master on the PCI bus
memeory space: device's response to I/O Space accesses

PCI classify and edition configuration register(PCI_CREV) (PCI config space 0x08)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Class Code																								Revision ID							
r																								r							
0x040000																								0							

class code: The Class Code register is read-only and is used to identify the generic function of the device and, in some cases, a specific register-level programming interface.The register value is set by register 0xc000008c.
revision id: This register specifies a device specific revision identifier. The register value is set by register 0xc000008c.

PCI Latency Timer Register（PCI_LT）(PCI config space 0x0C)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																LATENCY TIMER								reserved							
r																r/w						r		r							
0																0						0		0							

latency timer: the value for latency timer in PCI bus clock unit

PCI Base Address Register 0 (BAR0) (PCI config space 0x10)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																															
r																															
0																															

PCI Base Address Register 1 (BAR1) (PCI config space 0x14)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																															
r																															
0																															

PCI Base Address Register 0 (BAR2) (PCI config space 0x18)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																															
r																															
0																															

PCI Base Address Register 3 (BAR3) (PCI config space 0x1C)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BASE ADDRESS																												PF	TYPE		MSI
r/w																								r				r	r		r
0																								0000				0	0		0

base address: Memory base address.Size is decided by PCIAHB_SIZ_NP

pf: the address of space is no prefetchable

type: 32 bits wide and can be mapped

msi: that base address mapes Memory Space

PCI Base Address Register 4 (BAR4) (PCI config space 0x20)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BASE ADDRESS																												PF	TYPE		MSI
r/w																								r				r	r		r
0																								0000				1	0		0

base address: Memory base address. Size is decided by PCIAHB_SIZ_PF.

pf: the address of space is prefetchable

type: 32 bits wide and can be mapped

msi: that base address mapes Memory Space

PCI Base Address Register 5(BAR5) (PCI config space 0x24)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BASE ADDRESS																												PF	TYPE		MSI
r/w																								r				r	r		r
0																								0000				0	0		0

base address: config register base address

pf: the address of space is no prefetchable

type: 32 bits wide and can be mapped

msi: that base address mapes Memory Space

PCI Cardbus CIS Pointer Register(PCI config space 0x28)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																															
r																															
0																															

PCI Subsystem Device and Vendor Config Register（PCI_SUB）(PCI config space 0x2C)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SUBSYSTEM ID																SUBSYSTEM VENDOR ID															
r																r															
0																0															

subsystem id: pci subsystem device id,set by register 0xC0000088

subsystem vendor id: pci subsystem vendor id,set by register 0xC0000088

PCI extend ROM Base Address Register(PCI config space 0x30)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																															
r																															
0																															

PCI Capabilities Pointer Register(PCI config space 0x34)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																															
r																															
0																								0x78							

PCI Reserved Register(PCI config space 0x38)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																															
r																															
0																															

PCI Interrupt Line And Interrupt Pin Register(PCI config space 0x3C)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																INTERRUPT PIN								INTERRUPT LINE							
r																r								r/w							
0																0x1								0							

interrupt pin: interrupt pin,0x1 indicates device uses pci_inta_n as interrupt output pin

interrupt line: interrupt line,which pin of interrupt controller connected to interrupt signal pci_inta_n

PCI Interrupt Mask Register (PCI config space 0x80)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								CPU DOORBELL				reserved											READ DMA END	reserved							WRITE DMA END
r								r/w				r											r/w	r							r/w
0								0				0											0	0							0

CPU DOORBELL: CPU DOORBELL interrupt enable in simple bridge mode

read DMA end: read DMA end interrupt enable in simple bridge mode

write DMA end: write DMA end interrupt enable in simple bridge mode

PCI interrupt status register (PCI config space 0x84)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rserved								CPU DOORBELL				reserved											READ DMA END	reserved							WRITE DMA END
r								r/w				r											r/w	r							r/w
0								0000				0											0	0							0

CPU DOORBELL: CPU DOORBELL valid in simple bridge mode

read DMA end: read DMA end in simple bridge mode

write DMA end: write DMA end in simple bridge mode

PCI interrupt command register(PCI config space 0x88)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												PCI DOORBELL				reserved																
r												w				r															
0												0000				0															

PCI doorbell: PCI doorbell valid in simple bridge mode

PCI status and version information register(PCI config space 0x8C)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																TYPE				VERSION											
r																r				r											
0																xxxx				0x380											

type: PCI interface mode.1=32 bits simple bridge mode,2=32 bits host bridge mode.

version: PCI bridge version

DU Registers

DU Control Register（0x90000000）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved			DCNT									RE	DR	LR	SS	PE	EE	EB	DM	DE	BZ	BX	BB	BN	BS	BW	BE	FT	BT	DM	TE
r			r/w									r/w	r/w	r/w	r/w	r	r	r	r	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
0			x									0	0	0	0	0	x	x	0	0	x	x	0	x	0	x	x	0	0	x	0

dcnt: Trace buffer delay counter

re: Reset error mode

If set, clear the error mode in the processor.

dr: Debug mode response

If set, the DU communication link will send a response word when the processor enters debug mode

lr: Link response

If set, the DU communication link will send a response word after AHB transfer.

ss: Single step

If set, the processor will execute one instruction and the return to debug mode

pe: Processor error mode

Return ‘1’ on read when processor is in error mode else return ‘0’.

ee: value of the external DUEN signal (read-only)

eb: value of the external DUBRE signal (read-only)

dm: Debug mode

Indicate when the processor has entered debug mode (read-only).

de: Delay counter enable

If set, the trace buffer delay counter will decrement for each stored trace. This bit is set automatically when a DU breakpoint is hit and the delay counter is not equal to zero.

bz: Break on error traps

If set, will force the processor into debug mode on all except the following traps: priviledged_instruction, fpu_disabled, window_overflow, window_underflow, asynchronous_interrupt, ticc_trap.

bx: Break on trap

If set, will force the processor into debug mode when any trap occurs.

bd: Break on DU breakpoint

If set, will force the processor to debug mode when a DU breakpoint is hit.

bn: Break now

Force processor into debug mode. If cleared, the processor will resume execution.

bs: Break on S/W breakpoint

If set, debug mode will be forced when an breakpoint instruction (ta 1) is executed

bw: Break on IU watchpoint

If set, debug mode will be forced on a IU watchpoint (trap 0xb).

be: Break on error (BE) - if set, will force the processor to debug mode

When the processor would have entered error condition (trap in trap).

ft: Freeze timers

If set, the scalar in the timer unit will be stopped during debug mode to preserve the time for the software application.

bt: Break on trace

If set, generate a DU break condition on trace freeze.

dm: Delay counter mode (DM). In mixed tracing mode, setting this bit will cause the delay counter to decrement on AHB traces. If reset, the delay counter will decrement on instruction traces

te: Trace enable.

Enable the trace buffer.

Trace Buffer Control Register（0x90000004）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved						TA	TI	reserved			AHB INDEX									reserved			INST INDEX								
r						r/w	r/w	r			r/w									r			r/w								
0						x	x	0			x									0			x								

ta: Trace AHB enable.

ti: Trace instruction enable.

AHB index: AHB trace index counter (AHB Index [8:0]).

inst index: Instruction trace index counter (Inst Index [8:0]).

DU Trace Time Tag Register（0x90000008）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DU TIME TAG VALUE																														
r	r/w																														
0	0																														

DU Time tag value: count value of DU trace time tag.

AHB break address Register 1（0x90000010）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BADD																														R	EN
r/w																														r	w
x																														0	0

badd: Breakpoint address

en: Enables break on executed instruction

AHB Break Mask Register 1（0x90000014）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BMA																														LD	ST
r/w																														r/w	r/w
x																														0	0

bma: Breaking Mask

ld: Enables break on AHB load

st: Enables break on AHB write

AHB Break Address Register 2（0x90000018）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BADD																														R	EN
r/w																														r	w
x																														0	0

badd: Breakpoint address

en: Enables break on executed instruction

AHB Break Mask Register 2（0x9000001c）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BMA																														LD	ST
r/w																														r/w	r/w
x																														0	0

bma: Breaking Mask

ld: Enables break on AHB load

st: Enables break on AHB write

DU UART Status Register（0x800000C4）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																									FE	R	OV	BR	TH	TS	DR
r																									r/w	r	r/w	r/w	r	r	r
0																									0	0	0	0	1	1	0

fe: Framing error

Indicate that a framing error was detected.

ov: Overrun

Indicates that one or more character have been lost due to overrun.

br:Break flag

th: Transmitter hold register empty

Indicate that the transmitter hold register is empty.

ts: Transmitter shift register empty

Indicate that the transmitter shift register is empty.

dr: Data ready

Indicate that new data is available in the receiver holding register.

DU UART Control Register（0x800000C8）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																														BL	EN
r																														r/w	r/w
0																														0	0

bl: Baud rate locked .

Is automatically set when the baud rate is locked.

en: Receive an send enable

If set, enables both the transmitter and receiver.

DU UART Scalar Reload Register（0x800000CC）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved														SRV																	
r														r/w																	
0														3FFFF																	

SRV: Scalar reload Value.　SRV = （（sysclk*10）/（baudrate*8） - 5）/10.

Appendix5 IU/FPU register file address table

Global register:

%g0	0x9002-0280
%g1	0x9002-0284
%g2	0x9002-0288
%g3	0x9002-028C
%g4	0x9002-0290
%g5	0x9002-0294
%g6	0x9002-0298
%g7	0x9002-029C

Window register:

cwp = 0

%o0	0x9002-0020	%l0	0x9002-0040	%i0	0x9002-0060
%o1	0x9002-0024	%l1	0x9002-0044	%i1	0x9002-0064
%o2	0x9002-0028	%l2	0x9002-0048	%i2	0x9002-0068
%o3	0x9002-002C	%l3	0x9002-004C	%i3	0x9002-006C
%o4	0x9002-0030	%l4	0x9002-0050	%i4	0x9002-0070
%o5	0x9002-0034	%l5	0x9002-0054	%i5	0x9002-0074
%o6	0x9002-0038	%l6	0x9002-0058	%i6	0x9002-0078
%o7	0x9002-003C	%l7	0x9002-005C	%i7	0x9002-007C

cwp = 1

%o0	0x9002-0060	%l0	0x9002-0080	%i0	0x9002-00A0
%o1	0x9002-0064	%l1	0x9002-0084	%i1	0x9002-00A4
%o2	0x9002-0068	%l2	0x9002-0088	%i2	0x9002-00A8
%o3	0x9002-006C	%l3	0x9002-008C	%i3	0x9002-00AC
%o4	0x9002-0070	%l4	0x9002-0090	%i4	0x9002-00B0
%o5	0x9002-0074	%l5	0x9002-0094	%i5	0x9002-00B4
%o6	0x9002-0078	%l6	0x9002-0098	%i6	0x9002-00B8
%o7	0x9002-007C	%l7	0x9002-009C	%i7	0x9002-00BC

cwp = 2

%o0	0x9002-00A0	%l0	0x9002-00C0	%i0	0x9002-00E0

%o1	0x9002-00A4	%l1	0x9002-00C4	%i1	0x9002-00E4
%o2	0x9002-00A8	%l2	0x9002-00C8	%i2	0x9002-00E8
%o3	0x9002-00AC	%l3	0x9002-00CC	%i3	0x9002-00EC
%o4	0x9002-00B0	%l4	0x9002-00D0	%i4	0x9002-00F0
%o5	0x9002-00B4	%l5	0x9002-00D4	%i5	0x9002-00F4
%o6	0x9002-00B8	%l6	0x9002-00D8	%i6	0x9002-00F8
%o7	0x9002-00BC	%l7	0x9002-00DC	%i7	0x9002-00FC

cwp = 3

%o0	0x9002-00E0	%l0	0x9002-0100	%i0	0x9002-0120
%o1	0x9002-00E4	%l1	0x9002-0104	%i1	0x9002-0124
%o2	0x9002-00E8	%l2	0x9002-0108	%i2	0x9002-0128
%o3	0x9002-00EC	%l3	0x9002-010C	%i3	0x9002-012C
%o4	0x9002-00F0	%l4	0x9002-0110	%i4	0x9002-0130
%o5	0x9002-00F4	%l5	0x9002-0114	%i5	0x9002-0134
%o6	0x9002-00F8	%l6	0x9002-0118	%i6	0x9002-0138
%o7	0x9002-00FC	%l7	0x9002-011C	%i7	0x9002-013C

cwp = 4

%o0	0x9002-0120	%l0	0x9002-0140	%i0	0x9002-0160
%o1	0x9002-0124	%l1	0x9002-0144	%i1	0x9002-0164
%o2	0x9002-0128	%l2	0x9002-0148	%i2	0x9002-0168
%o3	0x9002-012C	%l3	0x9002-014C	%i3	0x9002-016C
%o4	0x9002-0130	%l4	0x9002-0150	%i4	0x9002-0170
%o5	0x9002-0134	%l5	0x9002-0154	%i5	0x9002-0174
%o6	0x9002-0138	%l6	0x9002-0158	%i6	0x9002-0178
%o7	0x9002-013C	%l7	0x9002-015C	%i7	0x9002-017C

cwp = 5

%o0	0x9002-0160	%l0	0x9002-0180	%i0	0x9002-01A0
%o1	0x9002-0164	%l1	0x9002-0184	%i1	0x9002-01A4
%o2	0x9002-0168	%l2	0x9002-0188	%i2	0x9002-01A8
%o3	0x9002-016C	%l3	0x9002-018C	%i3	0x9002-01AC
%o4	0x9002-0170	%l4	0x9002-0190	%i4	0x9002-01B0
%o5	0x9002-0174	%l5	0x9002-0194	%i5	0x9002-01B4
%o6	0x9002-0178	%l6	0x9002-0198	%i6	0x9002-01B8
%o7	0x9002-017C	%l7	0x9002-019C	%i7	0x9002-01BC

cwp = 6

%o0	0x9002-01A0	%l0	0x9002-01C0	%i0	0x9002-01E0
%o1	0x9002-01A4	%l1	0x9002-01C4	%i1	0x9002-01E4
%o2	0x9002-01A8	%l2	0x9002-01C8	%i2	0x9002-01E8
%o3	0x9002-01AC	%l3	0x9002-01CC	%i3	0x9002-01EC
%o4	0x9002-01B0	%l4	0x9002-01D0	%i4	0x9002-01F0
%o5	0x9002-01B4	%l5	0x9002-01D4	%i5	0x9002-01F4
%o6	0x9002-01B8	%l6	0x9002-01D8	%i6	0x9002-01F8
%o7	0x9002-01BC	%l7	0x9002-01DC	%i7	0x9002-01FC

cwp = 7

%o0	0x9002-01E0	%l0	0x9002-0000	%i0	0x9002-0020
%o1	0x9002-01E4	%l1	0x9002-0004	%i1	0x9002-0024
%o2	0x9002-01E8	%l2	0x9002-0008	%i2	0x9002-0028
%o3	0x9002-01EC	%l3	0x9002-000C	%i3	0x9002-002C
%o4	0x9002-01F0	%l4	0x9002-0010	%i4	0x9002-0030
%o5	0x9002-01F4	%l5	0x9002-0014	%i5	0x9002-0034
%o6	0x9002-01F8	%l6	0x9002-0018	%i6	0x9002-0038
%o7	0x9002-01FC	%l7	0x9002-001C	%i7	0x9002-003C

FPU register

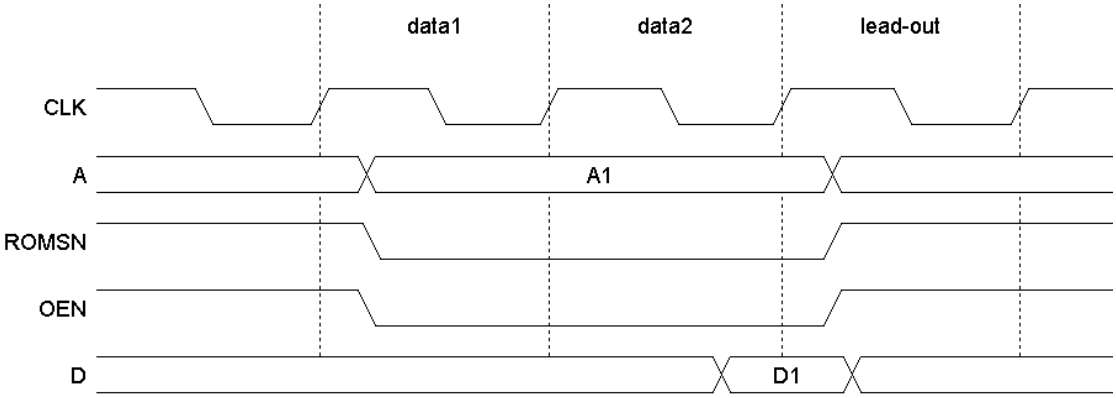
%f0	0x9002-0200	%f8	0x9002-0220
%f1	0x9002-0204	%f9	0x9002-0224
%f2	0x9002-0208	%f10	0x9002-0228
%f3	0x9002-020C	%f11	0x9002-022C
%f4	0x9002-0210	%f12	0x9002-0230
%f5	0x9002-0214	%f13	0x9002-0234
%f6	0x9002-0218	%f14	0x9002-0238
%f7	0x9002-021C	%f15	0x9002-023C

%f16	0x9002-0240	%f24	0x9002-0260
%f17	0x9002-0244	%f25	0x9002-0264
%f18	0x9002-0248	%f26	0x9002-0268
%f19	0x9002-024C	%f27	0x9002-026C
%f20	0x9002-0250	%f28	0x9002-0270
%f21	0x9002-0254	%f29	0x9002-0274

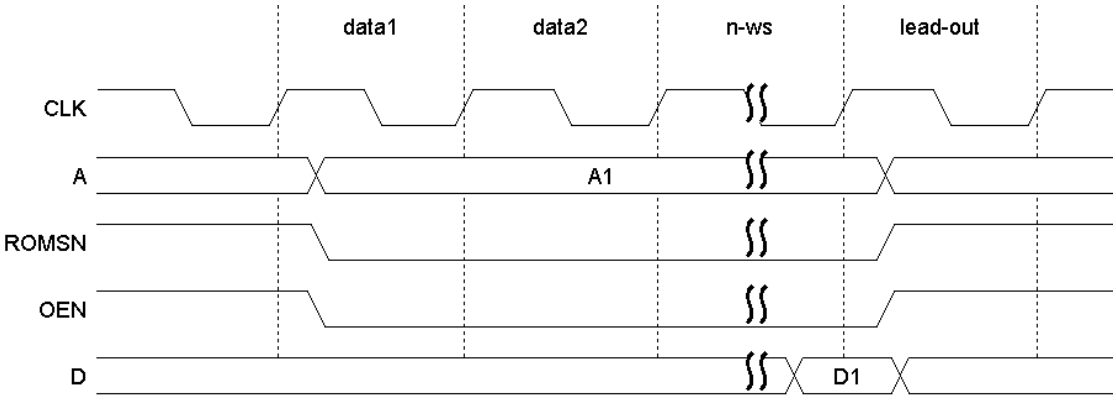
%f22	0x9002-0258	%f30	0x9002-0278
%f23	0x9002-025C	%f31	0x9002-027C

Appendix6 PROM and SDRAM timing

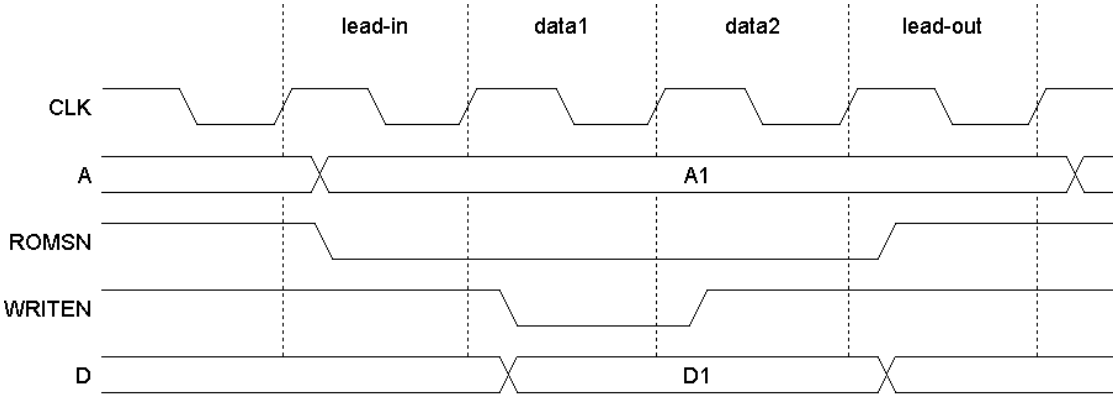
1. PROM read (0 wait cycle)



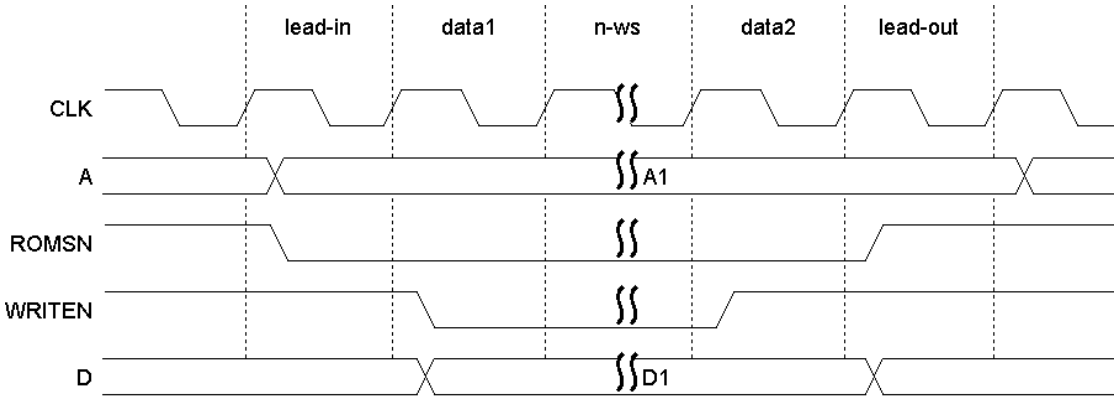
2. PROM read (n wait cycles)



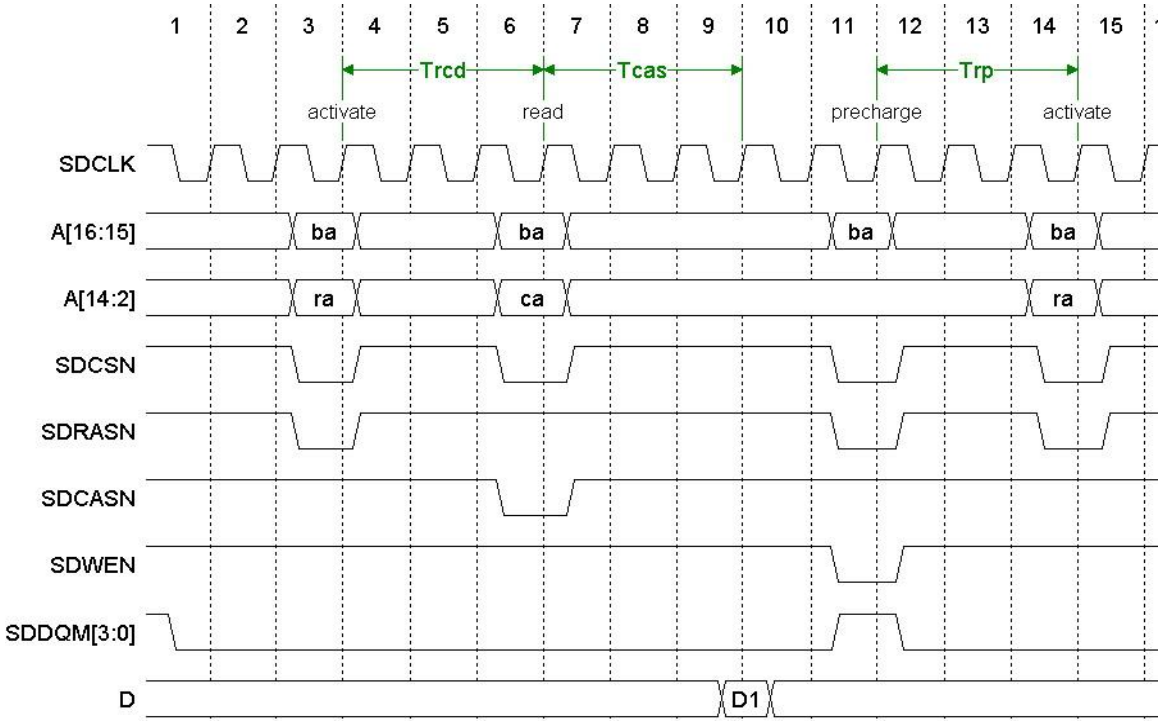
3. PROM write (0 wait cycle)



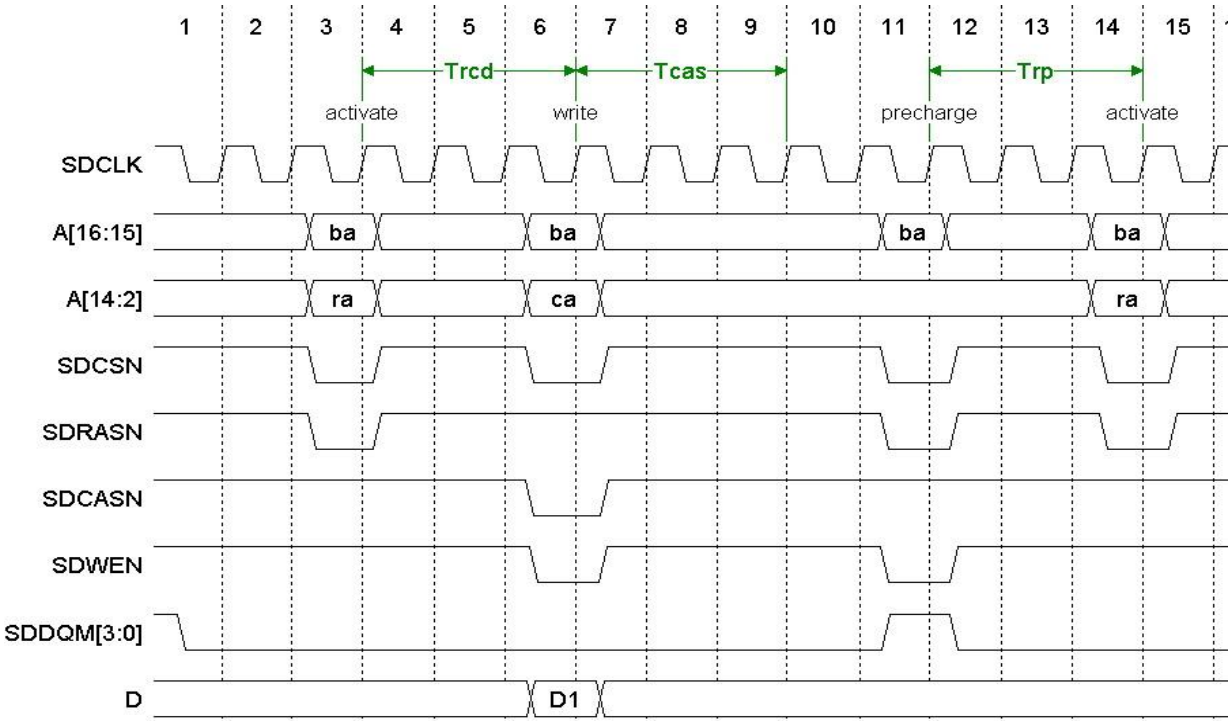
4. PROM write (n wait cycles)



5. SDRAM read (Burst length = 1; CL = 3)



6. SDRAM write (Burst length = 1; CL = 3)


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Trcd
Tcas
Trp
activate
write
precharge
activate
SDCLK
A[16:15]
A[14:2]
SDCSN
SDRASN
SDCASN
SDWEN
SDDQM[3:0]
D
ba
ba
ba
ba
ra
ca
ra
D1

Appendix7 Software exploitation criterion

7.1 BM3803 register introduction

7.1.1 register file introduction

BM3803 provides window register, globe integer register and floating point register.

Window register(%i0~%i7, %l0~%l7, %o0~%o7)

Window register contains in register, local register and out register.

The in and out registers are used primarily for passing parameters to subroutines and receiving results from them, and keeping track of the memory stack. When a procedure is called, the caller's outs become the callee's in register.

One of a procedure's out registers (%o6) is used as its stack pointer, %sp. It points to an area in which the system can store %r16... %r31(%l0... %l7and %i0... % i 7) when the register file overflows (window_overflow trap), and is used to address most values located on the stack.A trap2 can occur at any time, which may precipitate a subsequent window_overflow trap, during which the contents of the user's register window at the time of the original trap are spilled to the memory to which its %sp points.

The locals are used for automatic variables, and for most temporary values.

Global register(%g0~%g7)

Unlike the ins, locals, and outs, the globals are not part of any register window. The globals are a set of eight registers with global scope, like the register sets of more traditional processor architectures. The globals (except %g0) are conventionally assumed to be volatile across procedure calls. However, if they were used on a per-procedure basis and expected to be nonvolatile across procedure calls, either the caller or the callee would have to take responsibility for saving and restoring their contents. Global register %g0 has a "hardwired" value of zero. It always reads as zero, and writes to it have no effect. The global registers, other than %g0, can be used for temporaries, global variables, or global pointers — either user variables, or values

maintained as part of the program's execution environment.

Floating point register(%f0~%f31)

BM3803 has 32 32-bit floating point registers. Like the global registers, the floating-point registers must be managed by software. Compilers use the floating-point registers for user variables and compiler temporaries and return floating-point function values in them.

7.1.2 Special register introduction

Processor State Register, PSR

The 32-bit PSR contains various fields that control the processor and hold status information. It can be modified by the SAVE, RESTORE, Ticc, and RETT instructions, and by all instructions that modify the condition codes. The privileged RDPSR and WRPSR instructions read and write the PSR directly.

impl	*ver*	*icc*	reserved	EC	EF	PIL	S	PS	ET	CWP
31:28	27:24	23:20	19:14	13	12	11:8	7	6	5	4:0

Figure A 7-1 PSR

The PSR provides the following fields:

PSR_implementation (impl): Bits 31 through 28 are hardwired to identify an implementation or class of implementations of the architecture.

PSR_version(ver): Bits27 through 24 are implementation-dependent. The ver field is either hardwired to identify one or more particular implementations or is a readable and writable state field whose properties are implementation-dependent.

PSR_integer_cond_codes(icc): Bits 23 through 20 are the IU's condition codes. These bits are modified by the arithmetic and logical instructions whose names end with the letters cc (e.g., ANDcc), and by the WRPSR instruction. The Bicc and Ticc instructions cause a transfer of control based on the value of these bits, which are defined as follows:

n	*z*	*v*	*c*
23	22	21	20

Figure A 7-2 Integer Condition Codes (icc) Fields of the PSR

PSR_negative(n): Bit 23 indicates whether the 32-bit 2's complement ALU result

was negative for the last instruction that modified the icc field. 1 = negative, 0 = not negative.

PSR_zero(z): Bit 22 indicates whether the 32-bit ALU result was zero for the last instruction that modified the icc field. 1 = zero, 0 = nonzero.

PSR_overflow(v): Bit 21 indicates whether the ALU result was within the range of (was representable in) 32-bit 2's complement notation for the last instruction that modified the icc field. 1 = overflow, 0 = no overflow.

PSR_carry(c): Bit 20 indicates whether a 2's complement carry out (or borrow) occurred for the last instruction that modified the icc field. Carry is set on addition if there is a carry out of bit 31. Carry is set on subtraction if there is borrow into bit 31. 1 = carry, 0 = no carry.

PSR_reserved: Bits 19 through 14 are reserved. When read by a RDPSR instruction, these bits deliver zeros. For future compatibility, supervisor software should only issue WRPSR instructions with zero values in this field.

PSR_enable_coprocessor(EC): Bit 13 determines whether the implementation-dependent coprocessor is enabled. If disabled, a coprocessor instruction will trap. 1 = enabled, 0 = disabled. If an implementation does not support a coprocessor in hardware, PSR.EC should always read as 0 and write to it should be ignored.

PSR_enable_floating- point(EF): Bit 12 determines whether the FPU is enabled. If disabled, a floating-point instruction will trap. 1 = enabled, 0 = disabled.

PSR_proc_interrupt_level(PIL): Bits 11 (the most significant bit) through 8 (the least significant bit) identify the interrupt level above which the processor will accept an interrupt.

PSR_supervisor(S): determines whether the processor is in supervisor or user mode. 1 = supervisor mode, 0 = user mode.

PSR_previous_supervisor(PS): contains the value of the S bit at the time of the most recent trap.

PSR_enable_traps(ET): determines whether traps are enabled. A trap automatically resets ET to 0. When ET=0, an interrupt request is ignored and an exception trap causes the IU to halt execution, which typically results in a reset trap that resumes execution at address 0. 1 = traps enabled, 0 = traps disabled.

PSR_current_window_pointer(CWP): (the MSB) through 0 (the LSB) comprise

the current window pointer, a counter that identifies the current window into the r registers. The hardware decrements the CWP on traps and SAVE instructions, and increments it on RESTORE and RETT instructions (modulo NWINDOWS).

Window Invalid Mask Register, WIM

The Window Invalid Mask register (WIM) is controlled by supervisor software and is used by hardware to determine whether a window overflow or underflow trap is to be generated by a SAVE, RESTORE, or RETT instruction.

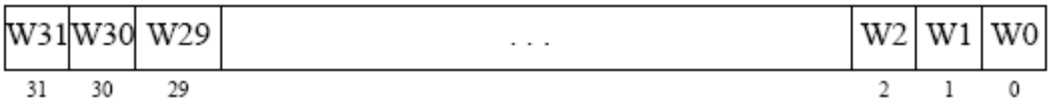


Figure A 7-3 WIM

BM3803 has 8 register windows, so the low 8-bit of this register is valid.

When a SAVE, RESTORE, or RETT instruction executes, the current value of the CWP is compared against the WIM. If the SAVE, RESTORE, or RETT instruction would cause the CWP to point to an “invalid” register set, that is, one whose corresponding WIM bit equals 1 (WIM[CWP] = 1), a window_overflow or window_underflow trap is caused.

The WIM can be read by the privileged RDWIM instruction and written by the WRWIM instruction. Bits corresponding to unimplemented windows read as zeroes and values written to unimplemented bits are unused. A WRWIM with all bits sets to 1, followed by a RDWIM, yields a bit vector in which the implemented windows (and only the implemented windows) are indicated by 1’s.

Trap Base Register, TBR

The Trap Base Register (TBR) contains three fields that together equal the address to which control is transferred when a trap occurs.

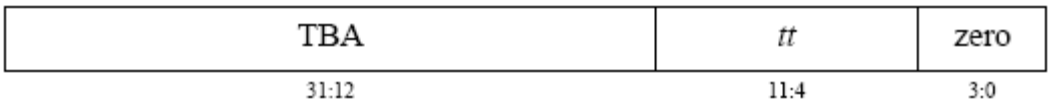


Figure A 7-4 TBR

The TBR provides the following fields:

TBR_trap_base_address (TBA): Bits 31 through 12 are the trap base address, which is established by supervisor software. It contains the most-significant 20 bits of the trap table address. The TBA field is written by the WRTBR instruction.

TBR_trap_type(tt): Bits 11 through 4 comprise the trap type (tt) field. This 8-bit field is written by the hardware when a trap occurs, and retains its value until the next trap. It provides an offset into the trap table. The WRTBR instruction does not affect the tt field.

TBR_zero(z): Bits 3 through 0 are zeroes. The WRTBR instruction does not affect this field. For future compatibility, supervisor software should only issue a WRTBR instruction with a zero value in this field.

Multiply/Divide Register, Y

The 32-bit Y register contains the most significant word of the double-precision product of an integer multiplication, as a result of either an integer multiply (SMUL, SMULcc, UMUL, UMULcc) instruction, or of a routine that uses the integer multiply step (MULScc) instruction. The Y register also holds the most significant word of the double-precision dividend for an integer divide (SDIV, SDIVcc, UDIV, UDIVcc) instruction. The Y register can be read and written with the RDY and WRY instructions.

Program Counters, PC, nPC

The 32-bit PC contains the address of the instruction currently being executed by the IU. The nPC holds the address of the next instruction to be executed (assuming a trap does not occur).

For a delayed control transfer, the instruction that immediately follows the transfer instruction is known as the delay instruction. This delay instruction is executed (unless the control transfer instruction annuls it) before control is transferred to the target. During execution of the delay instruction, the nPC points to the target of the control transfer instruction, while the PC points to the delay instruction.

The PC is read by a CALL or JMPL instruction. The PC and nPC are written to two local registers during a trap.

Ancillary State Registers, ASR

BM3803 inplements 11 ASR’s, the number is 16, 17, 18, 24~31.

ASR16, 17 are used for IU admit error, ASR18 is used for muliplication/addition operation, ASR24~31 are used for hardware watchpoint.

ASR’s numbered 1-15 are reserved for future use by the architecture and should not be referenced by software.

An ASR is read and written with the RDASR and WRASR instructions. A

read/write ASR instruction is privileged if the accessed register is privileged.

Floating-point State Register FSR

The FSR register fields contain FPU mode and status information. The FSR is read and written by the STFSR and LDFSR instructions.

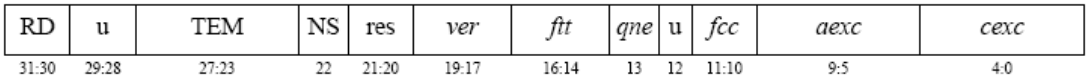


Figure A 7-5 FSR

FSR_rounding_direction(RD): Bits 31 and 30 select the rounding direction for floating-point results according to ANSI/IEEE Standard 754-1985.

Table A 7-1 Rounding Direction (RD) Field of FSR

RD	Round Toward:
0	Nearest (even, if tie)
1	0
2	+∞
3	−∞

FSR_unused(u): Bits 29, 28, and 12 are unused. For future compatibility, software should only issue a LDFSR instruction with zero values in these bits.

FSR_trap_enable_mask(TEM):Bits 27 through 23 are enable bits for each of the five floating-point exceptions that can be indicated in the current_exception field (cexc). If a floating-point operate instruction generates one or more exceptions and the TEM bit corresponding to one or more of the exceptions is 1, an fp_exception trap is caused. A TEM value of 0 prevents that exception type from generating a trap.

FSR_nonstandard_fp (NS): Bit 22, when set to 1, causes the FPU to produce implementation-defined results that may not correspond to ANSI/IEEE Standard 754-1985. For instance, to obtain higher performance, implementations may convert a subnormal floating-point operand or result to zero when NS is set.

FSR_reserved(res): Bits 21 and 20 are reserved. When read by an STFSR instruction, these bits deliver zeroes. For future compatibility, software should only issue LDFSR instructions with zero values in these bits.

FSR_version(ver): Bits 19 through 17 identify one or more particular implementations of the FPU architecture.

FSR_floating-point_trap_type(ftt):Bits 16 through 14 identify floating-point exception trap types. After a floating-point exception occurs, the ftt field encodes the

type of floating-point exception until an STFSR or another FPop is executed.

The ftt field can be read by the STFSR instruction. An LDFSR instruction does not affect ftt.

This field encodes the exception type according to Table A7-2.

Table A 7-2 Floating-point Trap Type (ftt) Field of FSR

ftt	Trap Type
0	None
1	IEEE_754_exception
2	unfinished_FPop
3	unimplemented_FPop
4	sequence_error
5	hardware_error
6	invalid_fp_register
7	*reserved*

The sequence_error and hardware_error trap types are not expected to arise in the normal course of computation. They are essentially unrecoverable, from the point of view of user applications.

In contrast, IEEE_754_exception, unfinished_FPop, and unimplemented_FPop are expected to arise occasionally in the normal course of computation and must be recoverable by supervisor software. When a floating-point trap occurs (as observed by a user signal (trap) handler):

1. The value of aexc is unchanged;

2. The value of cexc is unchanged, except that on an IEEE_754_exception exactly one bit corresponding to the trapping exception will be set. Unfinished_FPop, unimplemented_FPop, and sequence_error floating point exceptions do not affect cexc;

3. The source f registers are unchanged (usually implemented by leaving the destination f register unchanged);

4. The value of fcc is unchanged.

The foregoing describes the result seen by a user signal handler if an IEEE exception is signaled, either immediately from an IEEE_754_exception or after recovery from an unfinished_FPop or unimplemented_FPop. In either case, cexc as seen by the trap handler will reflect the exception causing the trap. In the cases of

unfinished_FPop and unimplemented_FPop traps that don't subsequently generate IEEE exceptions, the recovery software is expected to define cexc, aexc, and either the destination f register or fcc, as appropriate.

ftt=IEEE_754_exception:An IEEE_754_exception floating-point trap type indicates that a floating-point exception occurred that conforms to the ANSI/IEEE Standard 754-1985. The exception type is encoded in the cexc field. Note that aexc, fcc, and the destination f register are not affected by an IEEE_754_exception trap.

ftt=unfinished_FPop:An unfinished_FPop indicates that an implementation's FPU was unable to generate correct results or exceptions as defined by ANSI/IEEE Standard 754-1985. In this case, the cexc field is unchanged.

ftt=unimplemented_FPop:An unimplemented_FPop indicates that an implementation's FPU decoded an FPop that it does not implement. In this case, the cexc field is unchanged.

ftt=sequence_error:A sequence_error indicates one of three abnormal error conditions in the FPU, all caused by erroneous supervisor software:

— An attempt was made to execute a STDFQ instruction on an implementation without a floating-point deferred-trap queue (FQ);

— An attempt was made to execute a floating-point instruction when the FPU was not able to accept one. This type of sequence_error arises from a logic error in supervisor software that has caused a previous floating-point trap to be incompletely serviced (for example, the floating-point queue was not emptied after a previous floating-point exception).;

— An attempt was made to execute a STDFQ instruction when the floating-point deferred-trap queue (FQ) was empty, that is, when FSR.qne = 0. (Note that generation of sequence_error is recommended, but not required in this case)

ftt=hardware_error:A hardware_error indicates that the FPU detected a catastrophic internal error, such as an illegal state or a parity error on an f register access.

If a hardware_error occurs during execution of user code, it may not be possible to recover sufficient state to continue execution of the user application.

ftt=invalid_fp_register:An invalid_fp_register trap type indicates that one (or more) operands of an FPop are misaligned, that is, a double-precision register number is not 0 mod 2, or a quadruple-precision register number is not 0 mod 4. It is

recommended that implementations generate an fp_exception trap with FSR.ftt = invalid_fp_register in this case, but an implementation may choose not to generate a trap.

FSR_FQ_not_empty(qne):Bit 13 indicates whether the optional floating-point deferred-trap queue (FQ) is empty after a deferred fp_exception trap or after a store double floating-point queue (STDFQ) instruction has been executed. If qne = 0, the queue is empty; if qne = 1, the queue is not empty.

The qne bit can be read by the STFSR instruction. The LDFSR instruction does not affect qne. However, executing successive STDFQ instructions will (eventually) cause the FQ to become empty (qne = 0). If an implementation does not provide an FQ, this bit reads as zero. Supervisor software must arrange for this bit to always read as zero to user mode software.

FSR_fq_condition_codes(fcc):Bits 11 and 10 contain the FPU condition codes. These bits are updated by floating-point compare instructions (FCMP and FCMPE). They are read and written by the STFSR and LDFSR instructions, respectively. FBfcc bases its control transfer on this field.

In the following table, f_{rs1} and f_{rs2} correspond to the single, double, or quad values in the f registers specified by an instruction's rs1 and rs2 fields. The question mark (?) indicates an unordered relation, which is true if either f rs1 or f rs2 is a signaling NaN or quiet NaN. Note that fcc is unchanged if FCMP or FCMPE generates an IEEE_754_exception trap.

Table A 7-3 Floating-point Condition Codes (fcc) Field of FSR

fcc	Relation
0	$f_{rs1} = f_{rs2}$
1	$f_{rs1} < f_{rs2}$
2	$f_{rs1} > f_{rs2}$
3	f_{rs1} ? f_{rs2} (unordered)

FSR_accrued_exception(aexc): Bits 9 through 5 accumulate IEEE_754 floating-point exceptions while fp_exception traps are disabled using the TEM field. See Figure 4-10. After an FPop completes, the TEM and cexc fields are logically and'd together. If the result is nonzero, an fp_exception trap is generated; otherwise, the new cexc field is or'd into the aexc field. Thus, while traps are masked, exceptions

are accumulated in the aexc field.

FSR_current_exception(cexc): Bits 4 through 0 indicate that one or more IEEE_754 floating-point exceptions were generated by the most recently executed FPop instruction. The absence of an exception causes the corresponding bit to be cleared.

The cexc bits are set as described in section 4.4.2 by the execution of an FPop that either does not cause a trap or causes an fp_exception trap with FSR.ftt = IEEE_754_exception. It is recommended that an IEEE_754_exception which traps should cause exactly one bit in FSR.cexc to be set, corresponding to the detected IEEE 754 exception. If the execution of an FPop causes a trap other than an fp_exception due to an IEEE 754 exception, FSR.cexc is left unchanged.

Floating-Point Exception Fields

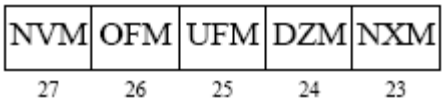


Figure A 7-6 Trap Enable Mask (TEM) Fields of FSR

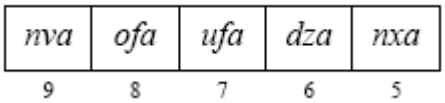


Figure A 7-7 Accrued Exception Bits (aexc) Fields of FSR

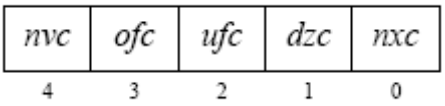


Figure A 7-8 Current Exception Bits (cexc) Fields of FSR

FSR_invalid(nvc, nva): An operand is improper for the operation to be performed. For example, 0 ÷0, and ∞.∞are invalid. 1 = invalid operand, 0 = valid operand(s).

FSR_overflow(ofc, ofa): The rounded result would be larger in magnitude than the largest normalized number in the specified format. 1 = overflow, 0 = no overflow.

FSR_underflow(ufc, ufa): The rounded result is inexact and would be smaller in magnitude than the smallest normalized number in the indicated format. 1 = underflow, 0 = no underflow.

Underflow is never indicated when the correct unrounded result is zero.

Otherwise:

if UFM=0: The ufc and ufa bits will be set if the correct unrounded result of an operation is less in magnitude than the smallest normalized number and the correctly-rounded result is inexact. These bits will be set if the correct unrounded result is less than the smallest normalized number, but the correct rounded result is the smallest normalized number. nxc and nxa are always set as well.

If UFM=1: An IEEE-754_exception trap will occur if the correct unrounded result of an operation would be smaller than the smallest normalized number. A trap will occur if the correct unrounded result would be smaller than the smallest normalized number, but the correct rounded result would be the smallest normalized number.

FSR_division-by-zero(dzc, dza): X ÷ 0, where X is subnormal or normalized. Note that 0 ÷ 0 does not set the dzc bit. 1 = division-by-zero, 0 = no division-by-zero.

FSR_inexact(nxc, nxa): The rounded result of an operation differs from the infinitely precise correct result. 1 = inexact result, 0 = exact result.

7.1.3 Register reccomand use

Table A7-4 Register reccomand use

in	%i7	(%r31)	return address− 8
	%fp, %i6	(%r30)	frame pointer
	%i5	(%r29)	input parameter 6
	%i4	(%r28)	input parameter 5
	%i3	(%r27)	input parameter 4
	%i2	(%r26)	input parameter 3
	%i1	(%r25)	input parameter 2
	%i0	(%r24)	input parameter 1/return value to caller
local	%l7	(%r23)	local variable 7
	%l6	(%r22)	local variable 6
	%l5	(%r21)	local variable 5
	%l4	(%r20)	local variable 4
	%l3	(%r19)	local variable 3
	%l2	(%r18)	local variable 2
	%l1	(%r17)	local variable 1
	%l0	(%r16)	local variable 0
out	%o7	(%r15)	temporary value / address of CALL instruction

	%sp, %o6	(%r14)	stack pointer
	%o5	(%r13)	output parameter 6
	%o4	(%r12)	output parameter 5
	%o3	(%r11)	output parameter 4
	%o2	(%r10)	output parameter 3
	%o1	(%r9)	output parameter 2
	%o0	(%r8)	outgoing param 1/ return value from callee
global	%g7	(%r7)	global variable 7 (SPARC ABI:reserved)
	%g6	(%r6)	global variable 6 (SPARC ABI:reserved)
	%g5	(%r5)	global variable 5 (SPARC ABI:reserved)
	%g4	(%r4)	global variable 4 (SPARC ABI:globalregvariable)
	%g3	(%r3)	global variable 3 (SPARC ABI:globalregvariable)
	%g2	(%r2)	global variable 2 (SPARC ABI:globalregvariable)
	%g1	(%r1)	temporary value
	%g0	(%r0)	0
state	%y		Y register (used in multiplication/division)
	(*icc* field of %psr)		Integer condition codes
	(*fcc* field of %fsr)		Floating-point condition codes
	(ccc field of %csr)		Coprocessor condition codes
floating point	%f31		floating-point value
	:		:
	:		:
	%f0		floating-point value

7.2 BM3803 Instruction description

7.2.1 Load Integer Instructions

Opcode	Name
LDSB	Load Signed Byte
LDSH	Load Signed Halfword
LDUB	Load Unsigned Byte
LDUH	Load Unsigned Halfword
LD	Load Word
LDD	Load Doubleword
LDA†	Load Word (from Alternate space)

† privileged instruction

Suggested Assembly Language Syntax	
ldsb	[*address*], reg_{rd}
ldsh	[*address*], reg_{rd}
ldub	[*address*], reg_{rd}
lduh	[*address*], reg_{rd}
ld	[*address*], reg_{rd}
ldd	[*address*], reg_{rd}
lda	[*regaddr*] *asi* , *regrd*

eg:

set 0x80000000, %o0

ld [%o0], %o1

200:

ld [200], %o1

ASI:

lda [%o0]0xc, %o1

Description:

The load integer instructions copy a byte, a halfword, or a word from memory into r[rd]. A fetched byte or halfword is right-justified in destination register r[rd]; it is either sign-extended or zero-filled on the left, depending on whether or not the opcode specifies a signed or unsigned operation, respectively.

The load doubleword integer instructions (LDD, LDDA) move a double-word from memory into an r register pair. The more significant word at the effective memory address is moved into the even r register. The less significant word (at the effective memory address + 4) is moved into the following odd r register. (Note that a load doubleword with rd = 0 modifies only r[1].) The least significant bit of the rd field is unused and should be set to zero by software. An attempt to execute a load doubleword instruction that refers to a mis-aligned (odd) destination register number may cause an illegal_instruction trap.

The effective address for a load instruction is “r[rs1]+ r[rs2]” if the i field is zero, or “r[rs1] + sign_ext(simm13)” if the i field is one. Instructions that load from an alternate address space contain the address space identifier to be used for the load in the asi field, and must contain zero in the i field or an illegal_instruction trap will occur. Load instructions that do not load from an alternate address space access either

a user data space or system data space, according to the S bit of the PSR.

LD and LDA cause a mem_address_not_aligned trap if the effective address is not word-aligned; LDUH, LDSH, LDUHA, and LDSHA trap if the address is not halfword-aligned; and LDD and LDDA trap if the address is not doubleword-aligned.

Traps:

illegal_instruction // load alternate with i = 1; LDD, LDDA with odd rd

privileged_instruction // load alternate space only

mem_address_not_aligned //excluding LDSB, LDSBA, LDUB, LDUBA

data_access_exception //

data_access_error //

7.2.2 Load Floating-point Instructions

Opcode	Name
LDF	Load Floating-point
LDDF	Load Double Floating-point
LDFSR	Load Floating-point State Register

Suggested Assembly Language Syntax		
ld	[address],	$freg_{rd}$
ldd	[address],	$freg_{rd}$
ld	[address],	%fsr

Description:

The load single floating-point instruction (LDF) moves a word from memory into f[rd].

The load doubleword floating-point instruction (LDDF) moves a doubleword from memory into an f register pair. The most significant word at the effective memory address is moved into the even f register. The least significant word at the effective memory address + 4 is moved into the following odd f register. The least significant bit of the rd field is unused and should always be set to zero by software. If this bit is non-zero, it is recommended that LDDF cause an fp_exception trap with FSR.ftt = invalid_fp_register.

The load floating-point state register instruction (LDFSR) waits for all FPop

instructions that have not finished execution to complete, and then loads a word from memory into the FSR. If any of the three instructions that follow (in time) a LDFSR is an FBfcc, the value of the fcc field of the FSR which is seen by the FBfcc is undefined.

The effective address for the load instruction is “r[rs1]+ r[rs2]” if the i field is zero, or “r[rs1] + sign_ext(simm13)” if the i field is one.

LDF and LDFSR cause a mem_address_not_aligned trap if the effective address is not word-aligned; LDDF traps if the address is not doubleword-aligned. If the EF field of the PSR is 0, or if no FPU is present, a load floating-point instruction causes an fp_disabled trap.

Traps:

fp_disabled

fp_exception (sequence_error, invalid_fp_register(LDDF))

data_access_exception

data_access_error

mem_address_not_aligned

7.2.3 Store Integer Instructions

Opcode	Name
STB	Store Byte
STH	Store Halfword
ST	Store Word
STD	Store Doubleword
STA†	Store Word (into Alternate space)

† privileged instruction

Suggested Assembly Language Syntax			
stb	reg_{rd} ,	[address]	(: stub, stsb)
sth	reg_{rd} ,	[address]	(: stuh, stsh)
st	reg_{rd} ,	[address]	
std	reg_{rd} ,	[address]	
sta	reg_{rd} ,	[regaddr] asi	

eg:

set 0x80000000, %o0

st %o1, [%o0]

st %o1, [200]

ASI:

sta %o1, [%o0] 0xd

Description:

The store integer instructions copy the word, the less significant halfword, or the least significant byte from r[rd] into memory.

The store doubleword integer instructions (STD, STDA) copy a doubleword from an r register pair into memory. The more significant word (in the even-numbered r register) is written into memory at the effective address, and the less significant word (in the following odd-numbered r register) is written into memory at the "effective address + 4". The least significant bit of the rd field of a store doubleword instruction is unused and should always be set to zero by software. An attempt to execute a store doubleword instruction that refers to a mis-aligned (odd) rd may cause an illegal_instruction trap.

The effective address for a store instruction is "r[rs1]+ r[rs2]" if the i field is zero, or "r[rs1] + sign_ext(simm13)" if the i field is one. Instructions that store to an alternate address space contain the address space identifier to be used for the store in the asi field, and must contain zero in the i field or an illegal_instruction trap will occur. Store instructions that do not store to an alternate address space access either a user data space or system data space, according to the S bit of the PSR.

ST and STA cause a mem_address_not_aligned trap if the effective address is not word-aligned. STH and STHA trap if the effective address is not halfword-aligned. STD and STDA trap if the effective address is not doubleword-aligned.

Traps:

illegal_instruction // store alternate with i = 1; STD, STDA with odd rd

privileged_instruction // store alternate only

mem_address_not_aligned // excluding STB and STBA

data_access_exception

data_access_error

data_store_error

7.2.4 Store Floating-point Instructions

Opcode	Name
STF	Store Floating-point
STDF	Store Double Floating-point
STFSR	Store Floating-point State Register
STDFQ†	Store Double Floating-point deferred-trap Queue

† privileged instruction

Suggested Assembly Language Syntax	
st	$freg_{rd}$, [address]
std	$freg_{rd}$, [address]
st	%fsr, [address]
std	%fq, [address]

Description:

The store single floating-point instruction (STF) copies f[rd] into memory.

The store double floating-point instruction (STDF) copies a doubleword from an f register pair into memory. The more significant word (in the even-numbered f register) is written into memory at the effective address, and the less significant word (in the odd-numbered f register) is written into memory at “effective address + 4”. The least significant bit of the rd field is unused and should always be set to zero by software. If this bit is non-zero, it is recommended that STDF cause an fp_exception trap with FSR.ftt = invalid_fp_register.

The store floating-point state register instruction (STFSR) waits for any con-currently executing FPop instructions that have not completed to complete, and then writes the FSR into memory. STFSR may zero FSR.ftt after writing the FSR to memory.

The effective address for a store instruction is “r[rs1]+ r[rs2]” if the i field is zero, or “r[rs1] + sign_ext(simm13)” if the i field is one.

STF and STFSR cause a mem_address_not_aligned trap if the address is not word-aligned and STDF and STDFQ trap if the address is not doubleword-aligned. If the EF field of the PSR is 0, or if the FPU is not present, a store floating-point

instruction causes an fp_disabled trap.

Traps:

fp_exception (invalid_fp_register(STDF)) // sequence_error(STDFQ)

mem_address_not_aligned

data_access_exception

data_access_error

data_store_error

7.2.5 Atomic Load-Store Unsigned Byte Instructions

Opcode	Name
LDSTUB	Atomic Load-Store Unsigned Byte

Suggested Assembly Language Syntax	
ldstub	[address], reg_{rd}

eg:

```
set     0x40000010, %o0
ldstub  [%o0], %l0
```

Description:

The atomic load-store instructions copy a byte from memory into r[rd], then rewrite the addressed byte in memory to all ones. The operation is per-formed atomically, that is, without allowing intervening interrupts or deferred traps. In a multiprocessor system, two or more processors executing atomic load-store unsigned byte, SWAP, or SWAPA instructions addressing the same byte or word simultaneously are guaranteed to execute them in an undefined, but serial order.

The effective address of an atomic load-store is “r[rs1]+ r[rs2]” if the i field is zero, or “r[rs1] + sign_ext(simm13)” if the i field is one. LDSTUBA must contain zero in the i field, or an illegal_instruction trap will occur. The address space identifier used for the memory accesses is taken from the asi field. For LDSTUB, the address space is either a user or a system data space access, according to the S bit in the PSR.

Traps:

illegal_instruction // LDSTUBA with i = 1 only

privileged_instruction // LDSTUBA only

data_access_exception

data_access_error

data_store_error

7.2.6 SWAP Register with Memory Instruction

Opcode	Name
SWAP SWAPA†	Swap r Register with Memory Swap r Register with Memory (in Alternate space)

† privileged instruction

Suggested Assembly Language Syntax	
swap	[address], reg_{rd}
swapa	[regaddr]asi , reg_{rd}

eg:

```
set     0x40001000, %o0
swap    [%o0], %o1

set     0x40001000, %o0
swapa   [%o0] 0xd, %o1
```

Description:

The SWAP and SWAPA instructions exchange r[rd] with the contents of the word at the addressed memory location. The operation is performed atomically, that is, without allowing intervening interrupts or deferred traps. In a multiprocessor system, two or more processors executing SWAP, SWAPA, or atomic load-store unsigned byte instructions addressing the same word or byte simultaneously are guaranteed to execute them in an undefined, but serial order.

The effective address of a SWAP instruction is “r[rs1]+ r[rs2]” if the i field is zero, or “r[rs1] + sign_ext(simm13)” if the i field is one. SWAPA must contain zero in the i field, or an illegal_instruction trap will occur. The address space identifier used for the memory accesses is taken from the asi field. For SWAP, the address space is either a user or a system data space, according to the S bit in the PSR.

These instructions cause a mem_address_not_aligned trap if the effective address

is not word-aligned.

Traps:

illegal instruction // when i = 1, SWAPA only

privileged_instruction // SWAPA only

mem_address_not_aligned

data_access_exception

data_access_error

data_store_error

7.2.7 SETHI Instruction

Opcode	Name
SETHI	Set High 22 bits

Suggested Assembly Language Syntax	
sethi	const22 , reg_{rd}
sethi	%hi(value), reg_{rd}

eg:

sethi 0x1fffff, %o0

sethi %hi(0xffffffff), %o1

Description:

SETHI zeroes the least significant 10 bits of "r[rd]", and replaces its high-order 22 bits with the value from its imm22 field. SETHI does not affect the condition codes. A SETHI instruction with rd = 0 and imm22 = 0 is defined to be a NOP instruction.

Traps: (none)

7.2.8 NOP Instruction

Opcode	Name
NOP	NOP

Suggested Assembly Language Syntax
nop

eg:

```
1:  cmp   %o0, %o1
    be    1b
    nop
```

Description:

The NOP instruction changes no program-visible state (except the PC and nPC). Note that NOP is a special case of the SETHI instruction, with imm22 =0 and rd =0.

Traps: (none)

7.2.9 Logical Instructions

Opcode	Name
AND	Add
ANDcc	And (and modify icc)
ANDN	And Not
ANDNcc	And Not (and modify icc)
OR	Inclusive-Or
ORcc	Inclusive-Or (and modify icc)
ORN	Inclusive-Or Not
ORNcc	Inclusive-Or Not (and modify icc)
XOR	Exclusive-Or
XORcc	Exclusive-Or (and modify icc)
XNOR	Exclusive-Nor
XNORcc	Exclusive-Nor (and modify icc)

Suggested Assembly Language Syntax	
and	reg_{rs1} , reg_or_imm , reg_{rd}
andcc	reg_{rs1} , reg_or_imm , reg_{rd}
andn	reg_{rs1} , reg_or_imm , reg_{rd}
andncc	reg_{rs1} , reg_or_imm , reg_{rd}
or	reg_{rs1} , reg_or_imm , reg_{rd}
orcc	reg_{rs1} , reg_or_imm , reg_{rd}
orn	reg_{rs1} , reg_or_imm , reg_{rd}
orncc	reg_{rs1} , reg_or_imm , reg_{rd}
xor	reg_{rs1} , reg_or_imm , reg_{rd}
xorcc	reg_{rs1} , reg_or_imm , reg_{rd}
xnor	reg_{rs1} , reg_or_imm , reg_{rd}
xnorcc	reg_{rs1} , reg_or_imm , reg_{rd}

eg:

and %l0, %l1, %o0

addn %l0, 15, %o0

Description:

These instructions implement the bitwise logical operations. They compute “r[rs1] operation r[rs2]” if the i field is zero, or “r[rs1] operation sign_ext(simm13)” if the i field is one, and write the result into r[rd].

ANDcc, ANDNcc, ORcc, ORNcc, XORcc, and XNORcc modify the integer condition codes (icc).

Traps: (none)

7.2.10 Shift Instructions

Opcode	Name
SLL	Shift Left Logical
SRL	Shift Right Logical
SRA	Shift Right Arithmetic

Suggested Assembly Language Syntax	
sll	reg_{rs1} , reg_or_imm , reg_{rd}
srl	reg_{rs1} , reg_or_imm , reg_{rd}
sra	reg_{rs1} , reg_or_imm , reg_{rd}

eg:

sll %l0, 4, %o0

sll %l0, %l1, %o0

Description:

The shift count for these instructions is the least significant five bits of r[rs2] if the i field is zero, or the value in shcnt if the i field is one. When i is 0, the most significant 27 bits of the value in r[rs2] are ignored. When i is 1, bits 5 through 12 of the shift instruction are reserved and should be supplied as zero by software.

SLL shifts the value of r[rs1] left by the number of bits given by the shift count. SRL and SRA shift the value of r[rs1] right by the number of bits implied by the shift count. SLL and SRL replace vacated positions with zeroes, whereas SRA fills vacated positions with the most significant bit of r[rs1]. No shift occurs when the shift count is

zero. All of these instructions write the shifted result into r[rd]. These instructions do not modify the condition codes.

Traps: (none)

7.2.11 Add Instructions

Opcode	Name
ADD	Add
ADDcc	Add (and modify icc)
ADDX	Add with Carry
ADDXcc	Add with Carry (and modify icc)

Suggested Assembly Language Syntax	
add	reg_{rs1} , reg_or_imm , reg_{rd}
addcc	reg_{rs1} , reg_or_imm , reg_{rd}
addx	reg_{rs1} , reg_or_imm , reg_{rd}
addxcc	reg_{rs1} , reg_or_imm , reg_{rd}

eg:

%l0 + %l1 = %o0 %l0 + 15 = %o0

add %l0, %l1, %o0

add %l0, 15, %o0

Description:

ADD and ADDcc compute “r[rs1]+ r[rs2]” if the i field is zero, or “r[rs1]+ sign_ext(simm13)” if the i field is one, and write the sum into r[rd].

ADDX and ADDXcc (“ADD eXtended”) also add the PSR’s carry (c) bit; that is, they compute “r[rs1]+ r[rs2]+ c” or “r[rs1] + sign_ext(simm13)+ c” and write the sum into r[rd].

Traps: (none)

7.2.12 Tagged Add Instructions

Opcode	name
TADDcc	Tagged Add and modify icc
TADDccTV	Tagged Add and modify icc (and Trap on overflow)

Suggested Assembly Language Syntax	
taddcc	reg_{rs1} , reg_or_imm , reg_{rd}
taddcctv	reg_{rs1} , reg_or_imm , reg_{rd}

eg:

%l0 + %l1 = %o0

taddcc %l0, %l1, %o0

%l0 + 15 = %o0

taddcctv %l0, 15, %o0

Description:

These instructions compute a sum that is “r[rs1]+ r[rs2]” if the i field is zero, or “r[rs1] + sign_ext(simm13)” if the i field is one.

TADDcc modifies the integer condition codes (icc), and TADDccTV does so also if it does not trap.

A tag_overflow occurs if bit 1 or bit 0 of either operand is nonzero, or if the addition generates an arithmetic overflow (both operands have the same sign and the sign of the sum is different).

If a TADDccTV causes a tag_overflow, a tag_overflow trap is generated and r[rd] and the condition codes remain unchanged. If a TADDccTV does not cause a tag_overflow, the integer condition codes are updated (in particular, the overflow bit (v) is set to 0) and the sum is written into r[rd].

If a TADDcc causes a tag_overflow, the overflow bit (v) of the PSR is set; if it does not cause a tag_overflow, the overflow bit is cleared. In either case, the remaining integer condition codes are also updated and the sum is written into r[rd].

Traps:

tag_overflow (TADDccTV only) // TADDccTV only

7.2.13 Subtract Instructions

Opcode	Name
SUB	Subtract
SUBcc	Subtract (and modify icc)
SUBX	Subtract with Carry
SUBXcc	Subtract with Carry (and modify icc)

Suggested Assembly Language Syntax	
sub	reg_{rs1} , reg_or_imm , reg_{rd}

subcc	reg_{rs1} , reg_or_imm , reg_{rd}
subx	reg_{rs1} , reg_or_imm , reg_{rd}
subxcc	reg_{rs1} , reg_or_imm , reg_{rd}

eg:

%l0 - %l1 = %o0 %l0 - 15 = %o0

sub %l0, %l1, %o0 sub %l0, 15, %o0

Description:

These instructions compute “r[rs1] . r[rs2]” if the i field is zero, or “r[rs1] . sign_ext(simm13)” if the i field is one, and write the difference into r[rd].

SUBX and SUBXcc (“SUBtract eXtended”) also subtract the PSR’s carry (c) bit; that is, they compute “r[rs1] . r[rs2] . c” or “r[rs1] . sign_ext(simm13) . c”, and write the difference into r[rd]. SUBcc and SUBXcc modify the integer condition codes (icc). Overflow occurs on subtraction if the operands have different signs and the sign of the difference differs from the sign of r[rs1].

Traps: (none)

7.2.14 Tagged Subtract Instructions

Opcode	Name
TSUBcc	Tagged Subtract and modify icc
TSUBccTV	Tagged Subtract and modify icc (and Trap on overflow)

Suggested Assembly Language Syntax
tsubcc reg_{rs1} , reg_or_imm , reg_{rd}
tsubcctv reg_{rs1} , reg_or_imm , reg_{rd}

eg:

%l0 - %l1 = %o0 %l0 - 15 = %o0

tsubcc %l0, %l1, %o0 tsubcc %l0, 15, %o0

Description:

These instructions compute “r[rs1] . r[rs2]” if the i field is zero, or “r[rs1] . sign_ext(simm13)” if the i field is one. TSUBcc modifies the integer condition codes (icc) and TSUBccTV does so also if it does not trap.

A tag_overflow occurs if bit 1 or bit 0 of either operand is nonzero, or if the subtraction generates an arithmetic overflow (the operands have different signs and

the sign of the difference differs from the sign of r[rs1]).

If a TSUBccTV causes a tag_overflow, a tag_overflow trap is generated and the destination register and condition codes remain unchanged. If a TSUBccTV does not cause a tag_overflow condition, the integer condition codes are updated (in particular, the overflow bit (v) is set to 0) and the difference is written into r[rd].

If a TSUBcc causes a tag_overflow, the overflow bit (v) of the PSR is set; if it does not cause a tag_overflow, the overflow bit is cleared. In either case, the remaining integer condition codes are also updated and the difference is written into r[rd].

Traps:

tag_overflow (TSUBccTV only) // TSUBccTV only

7.2.15 Multiply Step Instruction

Opcode	Name
MULScc	Multiply Step (and modify icc)

Suggested Assembly Language Syntax
mulscc regrs1 , reg_or_imm , regrd

eg:

```
set     0x5, %o0
mov     0x5, %y
set     0x5, %o1
mulscc  %o0, %o1, %o0
```

Description:

MULScc treats r[rs1] and the Y register as a single 64-bit, right-shiftable doubleword register. The least significant bit of r[rs1] is treated as if it were adjacent to the most significant bit of the Y register. The MULScc instruction conditionally adds, based on the least significant bit of Y.

Multiplication assumes that the Y register initially contains the multiplier, r[rs1] contains the most significant bits of the product, and r[rs2] contains the multiplicand. Upon completion of the multiplication, the Y register contains the least significant bits of the product.

MULScc operates as follows:

(1) The multiplier is established as r[rs2] if the i field is zero, or sign_ext(simm13) if the i field is one.

(2) A 32-bit value is computed by shifting r[rs1] right by one bit with "N xor V" from the PSR replacing the high-order bit. (This is the proper sign for the previous partial product.)

(3) If the least significant bit of the Y register = 1, the shifted value from step (2) is added to the multiplier.

If the LSB of the Y register = 0, then 0 is added to the shifted value from step (2).

(4) The sum from step (3) is written into r[rd].

(5) The integer condition codes, icc, are updated according to the addition performed in step (3).

(6) The Y register is shifted right by one bit, with the LSB of the unshifted r[rs1] replacing the MSB of Y.

Traps: (none)

7.2.16 Multiply Instructions

Opcode	Name
UMUL	Unsigned Integer Multiply
SMUL	Signed Integer Multiply
UMULcc	Unsigned Integer Multiply (and modify icc)
SMULcc	Signed Integer Multiply (and modify icc)

Suggested Assembly Language Syntax	
umul	reg_{rs1} , reg_or_imm , reg_{rd}
smul	reg_{rs1} , reg_or_imm , reg_{rd}
umulcc	reg_{rs1} , reg_or_imm , reg_{rd}
smulcc	reg_{rs1} , reg_or_imm , reg_{rd}

eg:

%l0 ×%l1 = %o0 %l0 ×15 = %o0

smul %l0, %l1, %o0

umul %l0, 15, %o0

Description:

The multiply instructions perform 32-bit by 32-bit multiplications, producing 64-bit results. They compute "r[rs1] × r[rs2]" if the i field is zero, or "r[rs1] × sign_ext(simm13)" if the i field is one. They write the 32 most significant bits of the product into the Y register and the 32 least significant bits into r[rd].

An unsigned multiply (UMUL, UMULcc) assumes unsigned integer word operands and computes an unsigned integer doubleword product. A signed multiply (SMUL, SMULcc) assumes signed integer word operands and computes a signed integer doubleword product.

UMUL and SMUL do not affect the condition code bits.

UMULcc and SMULcc write the integer condition code bits, icc, as follows. Note that negative (N) and zero (Z) are set according to the less significant word of the product.

icc	UMULcc	SMULcc
N	Set if product[31] = 1	Set if product[31] = 1
Z	Set if product[31:0] = 0	Set if product[31:0] = 0
V	0	0
C	0	0

Programming Note 32-bit overflow after UMUL/UMULcc is indicated by 32-bit overflow after SMUL/SMULcc is indicated by Y != 0. Y != (r[rd]>>31).

Traps: (none)

7.2.17 Divide Instructions

Opcode	Name
UDIV	Unsigned Integer Divide
SDIV	Signed Integer Divide
UDIVcc	Unsigned Integer Divide (and modify icc)
SDIVcc	Signed Integer Divide (and modify icc)

Suggested Assembly Language Syntax	
udiv	reg_{rs1} , reg_or_imm , reg_{rd}
sdiv	reg_{rs1} , reg_or_imm , reg_{rd}
udivcc	reg_{rs1} , reg_or_imm , reg_{rd}
sdivcc	reg_{rs1} , reg_or_imm , reg_{rd}

eg:

%l0 ÷%l1 = %o0 %l0 ÷15 = %o0

sdiv %l0, %l1, %o0

udiv %l0, 15, %o0

Description:

The divide instructions perform 64-bit by 32-bit division, producing a 32-bit result. If the i field is zero, they compute "(r[rs1]) ÷ r[rs2]". Otherwise (the i field is one), the divide instructions compute "(r[rs1]) ÷ sign_ext(simm13)". In either case, the 32 bits of the integer quotient are written into r[rd]. The remainder (if generated) is discarded.

An unsigned divide (UDIV, UDIVcc) assumes an unsigned integer double-word dividend (r[rs1]) and an unsigned integer word divisor (r[rs2]) and computes an unsigned integer word quotient (r[rd]). A signed divide (SDIV, SDIVcc) assumes a signed integer doubleword dividend (r[rs1])and a signed integer word divisor (r[rs2] or sign_ext(simm13)) and computes a signed integer word quotient (r[rd]).

Signed division rounds an inexact quotient toward zero if there is a nonzero remainder; for example, -3 ÷ 2 equals -1 with a remainder of -1 (not -2 with a remainder of 1).

The result of a divide instruction can overflow the 32-bit destination register r[rd] under certain conditions. When overflow occurs (whether or not the instruction sets the condition codes in icc), the largest appropriate integer is returned as the quotient in r[rd]. The conditions under which overflow occurs and the value returned in r[rd] under those conditions are specified in the following table:

Divide Overflow Detection and Value Returned		
Instruction	Condition under which overflow occurs	Value returned in r[rd]
UDIV, UDIVcc	result > 232 -1	232 -1 (0xffffffff)
SDIV, SDIVcc (positive result)	result > 231 -1	231 -1 (0x7fffffff)
SDIV, SDIVcc (negative result)	result < (− 231)	− 231 (0x80000000)

UDIV and SDIV do not affect condition code bits. UDIVcc and SDIVcc write

the integer condition code bits as follows.

icc	UDIVcc	SDIVcc
N	Set if quotient[31] = 1	Set if quotient[31] = 1
Z	Set if quotient[31:0] = 0	Set if quotient[31:0] = 0
V	Set if overflow (per above table)	Set if overflow (per above table)
C	0	0

Traps:

division_by_zero

7.2.18 SAVE and RESTORE Instructions

Opcode	op3	Name
SAVE	111100	Save caller's window
RESTORE	111101	Restore caller's window

Suggested Assembly Language Syntax
save reg_{rs1} , reg_or_imm , reg_{rd}
restore reg_{rs1} , reg_or_imm , reg_{rd}

eg:

save %sp, -112, %sp

restore

Description:

The SAVE instruction subtracts one from the CWP (modulo NWINDOWS) and compares this value (new_CWP) against the Window Invalid Mask (WIM) register. If the WIM bit corresponding to the new_CWP is 1, that is, (WIM and 2new_CWP) = 1, then a window_overflow trap is generated. If the WIM bit corresponding to the new_CWP is 0, then no window_overflow trap is generated and new_CWP is written into CWP. This causes the current window to become the CWP.1 window, thereby saving the caller's window.

The RESTORE instruction adds one to the CWP (modulo NWINDOWS) and compares this value (new_CWP) against the Window Invalid Mask (WIM) register. If the WIM bit corresponding to the new_CWP is 1, that is, (WIM and 2new_CWP) = 1, then a window_underflow trap is generated. If the WIM bit corresponding to the new_CWP = 0, then no window_underflow trap is generated and new_CWP is written

into CWP. This causes the CWP+1 window to become the current window, thereby restoring the caller's window.

Furthermore, if and only if an overflow or underflow trap is not generated, SAVE and RESTORE behave like normal ADD instructions, except that the source operands r[rs1] and/or r[rs2] are read from the old window (that is, the window addressed by the original CWP) and the sum is written into r[rd] of the new window (that is, the window addressed by new_CWP).

Traps:

window_overflow // SAVE only

window_underflow // RESTORE only

7.2.19 Branch on Integer Condition Codes Instructions

opcode	cond	operation	test icc
BA	1000	Branch Always	1
BN	0000	Branch Never	0
BNE	1001	Branch on Not Equal	not Z
BE	0001	Branch on Equal	Z
BG	1010	Branch on Greater	not (Z or (N xor V))
BLE	0010	Branch on Less or Equal	Z or (N xor V)
BGE	1011	Branch on Greater or Equal	not (N xor V)
BL	0011	Branch on Less	N xor V
BGU	1100	Branch on Greater Unsigned	not (C or Z)
BLEU	0100	Branch on Less or Equal Unsigned	(C or Z)
BCC	1101	Branch on Carry Clear (Greater than or Equal, Unsigned)	not C
BCS	0101	Branch on Carry Set (Less than, Unsigned)	C
BPOS	1110	Branch on Positive	not N
BNEG	0110	Branch on Negative	N
BVC	1111	Branch on Overflow Clear	not V
BVS	0111	Branch on Overflow Set	V

Suggested Assembly Language Syntax		
ba{, a}	label	
bn{, a}	label	
bne{, a}	label	(synonym: bnz)
be{, a}	label	(synonym: bz)
bg{, a}	label	

ble{, a}	label	
bge{, a}	label	
bl{, a}	label	
bgu{, a}	label	
bleu{, a}	label	
bcc{, a}	label	(synonym: bgeu)
bcs{, a}	label	(synonym: blu)
bpos{, a}	label	
bneg{, a}	label	
bvc{, a}	label	
bvs{, a}	label	

Note To set the “annul” bit for Bicc instructions, append “, a” to the opcode mnemonic. For example, use “bgu, a label”. The preceding table indicates that the “, a” is optional by enclosing it in braces ({}).

Eg:

```
1:  subcc   %o0, %o1, %o2
    bpos        1b
    nop

1:  subcc   %o0, %o1, %o2
    bpos, a 1b
    set     111, %o2
```

Description:

Unconditional Branches (BA, BN)

If its annul field is 0, a BN (Branch Never) instruction acts like a “NOP”. If its annul field is 1, the following (delay) instruction is annulled (not executed). In neither case does a transfer of control take place.

BA (Branch Always) causes a PC-relative, delayed control transfer to the address “PC + (4 × sign_ext(disp22))”, regardless of the values of the integer condition code bits. If the annul field of the branch instruction is 1, the delay instruction is annulled (not executed). If the annul field is 0, the delay instruction is executed.

Icc-Conditional Branches

Conditional Bicc instructions (all except BA and BN) evaluate the integer condition codes (icc), according to the cond field of the instruction. Such evaluation produces either a “true” or “false” result. If “true”, the branch is taken, that is, the

instruction causes a PC-relative, delayed control transfer to the address "PC + (4 × sign_ext(disp22))". If "false", the branch is not taken.

If a conditional branch is taken, the delay instruction is always executed regardless of the value of the annul field. If a conditional branch is not taken and the a (annul) field is 1, the delay instruction is annulled (not executed). (Note that the annul bit has a different effect on conditional branches than it does on unconditional branches.)

Traps: (none)

7.2.20 Branch on Floating-point Condition Codes Instructions

opcode	cond	operation	test fcc
FBA	1000	Branch Always	1
FBN	0000	Branch Never	0
FBU	0111	Branch on Unordered	U
FBG	0110	Branch on Greater	G
FBUG	0101	Branch on Unordered or Greater	G or U
FBL	0100	Branch on Less	L
FBUL	0011	Branch on Unordered or Less	L or U
FBLG	0010	Branch on Less or Greater	L or G
FBNE	0001	Branch on Not Equal	L or G or U
FBE	1001	Branch on Equal	E
FBUE	1010	Branch on Unordered or Equal	E or U
FBGE	1011	Branch on Greater or Equal	E or G
FBUGE	1100	Branch on Unordered or Greater or Equal	E or G or U
FBLE	1101	Branch on Less or Equal	E or L
FBULE	1110	Branch on Unordered or Less or Equal	E or L or U
FBO	1111	Branch on Ordered	E or L or G

Suggested Assembly Language Syntax	
fba{, a}	label
fbn{, a}	label
fbu{, a}	label
fbg{, a}	label
fbug{, a}	label
fbl{, a}	label
fbul{, a}	label
fblg{, a}	label

fbne{, a}	label	(synonym: fbnz)
fbe{, a}	label	(synonym: fbz)
fbue{, a}	label	
fbge{, a}	label	
fbuge{, a}	label	
fble{, a}	label	
fbule{, a}	label	
fbo{, a}	label	

Note To set the “annul” bit for FBfcc instructions, append “, a” to the opcode mnemonic. For example, use “fbl, a label”. The preceding table indicates that the “, a” is optional by enclosing it in braces ({}).

Description:

Unconditional Branches (FBA, FBN)

If its annul field is 0, a FBN (Branch Never) instruction acts like a “NOP”. If its annul field is 1, the following (delay) instruction is annulled (not executed). In neither case does a transfer of control take place.

FBA (Branch Always) causes a PC-relative, delayed control transfer to the address “PC + (4 × sign_ext(disp22))”, regardless of the value of the floating-point condition code bits. If the annul field of the branch instruction is 1, the delay instruction is annulled (not executed). If the annul field is 0, the delay instruction is executed.

Fcc-Conditional Branches

Conditional FBfcc instructions (all except FBA and FBN) evaluate the floating-point condition codes (fcc), according to the cond field of the instruction. Such evaluation produces either a “true” or “false” result. If “true”, the branch is taken, that is, the instruction causes a PC-relative, delayed control transfer to the address “PC + (4 × sign_ext(disp22))”. If “false”, the branch is not taken.

If a conditional branch is taken, the delay instruction is always executed regardless of the value of the annul field. If a conditional branch is not taken and the a (annul) field is 1, the delay instruction is annulled (not executed). (Note that the annul bit has a different effect on conditional branches than it does on unconditional branches.)

If the PSR’s EF bit is 0, or if an FPU is not present, an FBfcc instruction does not branch, does not annul the following instruction, and generates an fp_disabled trap.

If the instruction executed immediately before an FBfcc is an FPop2 instruction, the result of the FBfcc is undefined. Therefore, at least one non-FPop2 instruction should be executed between an FPop2 and a subsequent FBfcc. If any of the three instructions that follow (in time) an LDFSR is an FBfcc, the value of the fcc field of the FSR that is seen by the FBfcc is undefined.

Traps:

fp_disabled

fp_exception

7.2.21 Call and Link Instruction

opcode	operation
CALL	Call and Link

Suggested Assembly Language Syntax
call label

eg:

call main

nop

Description:

The CALL instruction causes an unconditional, delayed, PC-relative control transfer to address “PC + (4 × disp30)”. Since the word displacement (disp30) field is 30 bits wide, the target address can be arbitrarily distant. The PC-relative displacement is formed by appending two low-order zeros to the instruction’s 30-bit word displacement field. The CALL instruction also writes the value of PC, which contains the address of the CALL, into r[15] (out register 7).

Traps: (none)

7.2.22 Jump and Link Instruction

opcode	operation
JMPL	Jump and Link

Suggested Assembly Language Syntax
jmpl address , reg_{rd}

eg:

```
set     0x40000000, %o0
jmpl    %o0, %o1
```

Description:

The JMPL instruction causes a register-indirect delayed control transfer to the address given by “r[rs1]+ r[rs2]” if the i field is zero, or “r[rs1]+ sign_ext(simm13)” if the i field is one.

The JMPL instruction copies the PC, which contains the address of the JMPL instruction, into register r[rd].

If either of the low-order two bits of the jump address is nonzero, a mem_address_not_aligned trap occurs.

A JMPL instruction with rd = 15 functions as a register-indirect call using the standard link register. JMPL with rd = 0 can be used to return from a subroutine. The typical return address is “r[31]+8”, if a non-leaf (uses SAVE instruction) subroutine is entered by a CALL instruction, or “r[15]+8” if a leaf (doesn’t use SAVE instruction) subroutine is entered by a CALL instruction.

Traps:

mem_address_not_aligned

7.2.23 Return from Trap Instruction

opcode	***operation***
RETT†	Return from Trap

† privileged instruction

Suggested Assembly Language Syntax
rett *address*

Eg:

```
jmpl    %l1, %g0
rett    %l2
```

Description:

RETT is used to return from a trap handler. Under some circumstances, RETT may itself cause a trap. If a RETT instruction does not cause a trap, it (1) adds 1 to the CWP (modulo NWINDOWS), (2) causes a delayed control transfer to the target address, (3) restores the S field of the PSR from the PS field, and (4) sets the ET field of the PSR to 1. The target address is "r[rs1] +r[rs2]" if the i field is zero, or "r[rs1] + sign_ext(simm13)" if the i field is one.

One of several traps may occur when an RETT is executed. These are described in priority order (highest priority first):

If traps are enabled (ET=1) and the processor is in user mode (S=0), a privileged_instruction trap occurs.

If traps are enabled (ET=1) and the processor is in supervisor mode (S=1), an illegal_instruction trap occurs.

If traps are disabled (ET=0), and (a) the processor is in user mode (S=0), or (b) a window_underflow condition is detected (WIM and 2^{new_CWP}) = 1, or (c) either of the low-order two bits of the target address is nonzero, then the processor indicates a trap condition of (a) privileged_instruction, (b)window_underflow, or (c) mem_address_not_aligned (respectively) in the tt field of the TBR register, and enters the error_mode state.

The instruction executed immediately before an RETT must be a JMPL instruction. (If not, one or more instruction accesses following the RETT may be to an incorrect address space.)

Programming Note: To reexecute the trapped instruction when returning from a trap handler use the sequence:

```
jmpl    %l1, %g0        ! old PC
rett    %12             ! old nPC
```

To return to the instruction after the trapped instruction (for example, after emulating an instruction) use the sequence:

```
jmpl    %l2, %g0        ! old nPC
rett    %l2+4           ! old nPC + 4
```

Traps:

illegal_instruction

privileged_instruction

privileged_instruction // may cause processor to enter error_mode

mem_address_not_aligned // may cause processor to enter error_mode

window_underflow // may cause processor to enter error_mode

7.2.24 Trap on Integer Condition Codes Instruction

opcode	*operation*	*icc test*
TA	Trap Always	1
TN	Trap Never	0
TNE	Trap on Not Equal	**not** Z
TE	Trap on Equal	Z
TG	Trap on Greater	**not** (Z **or** (N **xor** V))
TLE	Trap on Less or Equal	Z **or** (N **xor** V)
TGE	Trap on Greater or Equal	**not** (N **xor** V)
TL	Trap on Less	N **xor** V
TGU	Trap on Greater Unsigned	**not** (C **or** Z)
TLEU	Trap on Less or Equal Unsigned	(C **or** Z)
TCC	Trap on Carry Clear (Greater than or Equal, Unsigned)	**not** C
TCS	Trap on Carry Set (Less Than, Unsigned)	C
TPOS	Trap on Positive	**not** N
TNEG	Trap on Negative	N
TVC	Trap on Overfw Clear	**not** V
TVS	Trap on Overflow Set	V

Suggested Assembly Language Syntax		
ta	*software_trap#*	
tn	*software_trap#*	
tne	*software_trap#*	*(synonym:* tnz*)*
te	*software_trap#*	*(synonym:* tz*)*
tg	*software_trap#*	
tle	*software_trap#*	
tge	*software_trap#*	
tl	*software_trap#*	

tgu	*software_trap#*	
tleu	*software_trap#*	
tcc	*software_trap#*	*(synonym:* tgeu*)*
tcs	*software_trap#*	*(synonym:* tlu*)*
tpos	*software_trap#*	
tneg	*software_trap#*	
tvc	*software_trap#*	
tvs	*software_trap#*	

Eg:

ta 4

ta %l0

Description:

A Ticc instruction evaluates the integer condition codes (icc) according to the cond field of the instruction, producing either a “true” or “false” result. If “true” and no higher priority exceptions or interrupt requests are pending, then a trap_instruction trap is generated. If “false”, a trap_instruction trap does not occur and the instruction behaves like a NOP.

If a trap_instruction trap is generated, the tt field of the Trap Base Register (TBR) is written with 128 plus the least signifcant seven bits of “r[rs1]+ r[rs2]” if the i field is zero, or 128 plus the least signifcant seven bits of “r[rs1] + sign_ext(software_trap#)” if the i field is one.

After a taken Ticc, the processor enters supervisor mode, disables traps, decrements the CWP (modulo NWINDOWS), and saves PC and nPC into r[17] and r[18] (local registers 1 and 2) of the new window.

Programming Note: Ticc can be used to implement breakpointing, tracing, and calls to supervisor software. It can also be used for run-time checks, such as out-of-range array indexes, integer overflow, etc.

Traps:

trap_instruction

7.2.25 Read State Register Instructions

opcode	***operation***

RDY	Read Y Register
RDASR‡	Read Ancillary State Register
RDPSR†	Read Processor State Register
RDWIM†	Read Window Invalid Mask Register
RDTBR†	Read Trap Base Register

† privileged instruction

‡ privileged instruction if source register is privileged

Suggested Assembly Language Syntax	
rd	%y, reg_{rd}
rd	asr_reg_{rs1} , reg_{rd}
rd	%psr, reg_{rd}
rd	%wim, reg_{rd}
rd	%tbr, reg_{rd}

Eg:

rd %psr, %o0

Description:

These instructions read the specified IU state register into r[rd].

In BM3803, there are two Ancillary State Registers, %asr16 and %asr17.

Traps:

privileged_instruction // except RDY

illegal_instruction // RDASR only; implementation-dependent

7.2.26 Write State Register Instructions

opcode	*operation*
WRY	Write Y Register
WRASR‡	Write Ancillary State Register
WRPSR†	Write Processor State Register
WRWIM†	Write Window Invalid Mask Register
WRTBR†	Write Trap Base Register

† privileged instruction

‡ privileged instruction if destination register is privileged

Suggested Assembly Language Syntax
wr reg_{rs1} , *reg_or_imm* , %y
wr reg_{rs1} , *reg_or_imm* , asr_reg_{rd}
wr reg_{rs1} , *reg_or_imm* , %psr
wr reg_{rs1} , *reg_or_imm* , %wim
wr reg_{rs1} , *reg_or_imm* , %tbr

Eg:

```
set  0xffffffff, %o0
wr   %o0, %g0, %y
```

Description:

WRY, WRPSR, WRWIM, and WRTBR write “r[rs1] xor r[rs2]” if the i field is zero, or “r[rs1] xor sign_ext(simm13)” if the i field is one, to the writable fields of the specified IU state register. (Note the exclusive-or operation.)

Note that WRY is distinguished from WRASR only by the rd field. The rd field must be zero and op3 = 0x30 to write the Y register.

If the result of a WRPSR instruction would cause the CWP field of the PSR to point to an unimplemented window, it causes an illegal_instruction trap and does not write the PSR.

Traps:

```
privileged_instruction     // except WRY
illegal_instruction        // WRPSR, if CWP ≥ NWINDOWS
illegal_instruction        // WRASR; implementation-dependent
```

7.2.27 Unimplemented Instruction

opcode	*operation*
UNIMP	Unimplemented

Suggested Assembly Language Syntax
unimp *const22*

eg:

unimp

Description:

The UNIMP instruction causes an illegal_instruction trap. The const22 value is ignored by the hardware; specifically, its values are not reserved by the architecture for any future use.

Traps:

illegal_instruction

7.2.28 Flush Instruction Memory

opcode	***operation***
FLUSH	Flush Instruction Memory

Suggested Assembly Language Syntax
flush *address*

eg:

flush

Description:

The FLUSH instruction ensures that subsequent instruction fetches to the target of the FLUSH by the processor executing the FLUSH appear to execute after any loads, stores, and atomic load-stores issued by that processor prior to the FLUSH.

The effective virtual address operand for the FLUSH instruction is "r[rs1] +r[rs2]" if the i field is zero, or "r[rs1] + sign_ext(simm13)" if the i field is one.

Attention: In BM3803, flush instruction is different from the flush function in criterion of SPARC V8.

Traps:

unimplemented_FLUSH

illegal_instruction

7.2.29 Convert Integer to Floating point Instructions

opcode	***operation***

FiTOs	Convert Integer to Single
FiTOd	Convert Integer to Double

Suggested Assembly Language Syntax		
fitos	$freg_{rs2}$,	$freg_{rd}$
fitod	$freg_{rs2}$,	$freg_{rd}$

Description:

These instructions convert the 32-bit integer word operand in f[rs2] into a floating-point number in the destination format. They write the result into the f register(s) specified by rd.

FiTOs rounds according to the RD field of the FSR.

Traps:

fp_disabled

fp_exception (NX (FiTOs only), invalid_fp_register (FiTOd)

7.2.30 Convert Floating point to Integer Instructions

opcode	*operation*
FsTOi	Convert Single to Integer
FdTOi	Convert Double to Integer

Suggested Assembly Language Syntax		
fstoi	$freg_{rs2}$,	$freg_{rd}$
fdtoi	$freg_{rs2}$,	$freg_{rd}$

Description:

These instructions convert the floating-point operand in the f register(s) specified by rs2 into a 32-bit integer word in f[rd].

The result is always rounded toward zero (the RD field of the FSR register is ignored).

Traps:

fp_disabled

fp_exception (NV, NX, invalid_fp_register (FdTOi, FqTOi))

7.2.31 Convert Between Floating-point Formats Instructions

opcode	*operation*
FsTOd	Convert Single to Double
FdTOs	Convert Double to Single

Suggested Assembly Language Syntax		
fstod	*freg*$_{rs2}$,	*freg*$_{rd}$
fdtos	*freg*$_{rs2}$,	*freg*$_{rd}$

Description:

These instructions convert the floating-point operand in the f register(s) specified by rs2 to a floating-point number in the destination format. They write the result into the f register(s) specified by rd.

Rounding is performed according to the RD field of the FSR.

FdTOs can raise OF, UF and NX exceptions, FsTOd cannot.

Either of these two instructions can trigger an NV exception if the source operand is a signaling NaN.

Traps:

fp_disabled

fp_exception (OF, UF, NV, NX, invalid_fp_register)

7.2.32 Floating-point Move Instructions

opcode	operation
FMOVs	Move
FNEGs	Negate
FABSs	Absolute Value

Suggested Assembly Language Syntax		
fmovs	*freg*$_{rs2}$,	*freg*$_{rd}$
fnegs	*freg*$_{rs2}$,	*freg*$_{rd}$
fabss	*freg*$_{rs2}$,	*freg*$_{rd}$

Description:

FMOVs copies the contents of f[rs2]to f[rd].

FNEGs copies the contents of f[rs2]to f[rd] with the sign bit complemented.

FABSs copies the contents of f[rs2]to f[rd] with the sign bit cleared.

These instructions do not round.

To transfer a multiple-precision value between f registers, one FMOVs instruction is required per word to be transferred.

If the source and destination registers ($freg_{rs2}$ and $freg_{rd}$) are the same, a single FNEGs (FABSs) instruction performs negation (absolute-value) for any operand precision.

If the source and destination registers are different, a double-precision negation (absolute value) is performed by a FNEGs (FABSs) and a FMOVs instruction. Similarly, a quad-precision negation (absolute value) requires a FNEGs (FABSs) and three FMOVs instructions.

Traps:

fp_disabled

7.2.33 Floating-point Square Root Instructions

opcode	***operation***
FSQRTs	Square Root Single
FSQRTd	Square Root Double

Suggested Assembly Language Syntax		
fsqrts	*$freg_{rs2}$*,	*$freg_{rd}$*
fsqrtd	*$freg_{rs2}$*,	*$freg_{rd}$*

Description:

These instructions generate the square root of the floating-point operand in the f register(s) specified by the rs2 field. They place the result in the destination f register(s) specified by the rd field.

Rounding is performed according to the rd field of the FSR.

Traps:

fp_disabled

fp_exception (NV, NX, invalid_fp_register (FSQRTd, FSQRTq))

7.2.34 Floating-point Add and Subtract Instructions

opcode	*operation*
FADDs	Add Single
FADDd	Add Double
FSUBs	Subtract Single
FSUBd	Subtract Double

Suggested Assembly Language Syntax			
fadds	$freg_{rs1}$,	$freg_{rs2}$,	$freg_{rd}$
faddd	$freg_{rs1}$,	$freg_{rs2}$,	$freg_{rd}$
fsubs	$freg_{rs1}$,	$freg_{rs2}$,	$freg_{rd}$
fsubd	$freg_{rs1}$,	$freg_{rs2}$,	$freg_{rd}$

Description:

The floating-point add instructions add the f register(s) specified by the rs1 field and the f register(s) specified by the rs2 field, and write the sum into the f register(s) specified by the rd field.

The floating-point subtract instructions subtract the f register(s) specified by the rs2 field from the f register(s) specified by the rs1 field, and write the difference into the f register(s) specified by the rd field.

Traps:

fp_disabled

fp_exception (OF, UF, NX, NV,

invalid_fp_register (all except FADDs and FSUBs))

7.2.35 Floating-point Multiply and Divide Instructions

opcode	*operation*
FMULs	Multiply Single
FMULd	Multiply Double
FsMULd	Multiply Single to Double

FDIVs	Divide Single
FDIVd	Divide Double

Suggested Assembly Language Syntax			
fmuls	*freg*$_{rs1}$,	*freg*$_{rs2}$,	*freg*$_{rd}$
fmuld	*freg*$_{rs1}$,	*freg*$_{rs2}$,	*freg*$_{rd}$
fsmuld	*freg*$_{rs1}$,	*freg*$_{rs2}$,	*freg*$_{rd}$
fdivs	*freg*$_{rs1}$,	*freg*$_{rs2}$,	*freg*$_{rd}$
fdivd	*freg*$_{rs1}$,	*freg*$_{rs2}$,	*freg*$_{rd}$

Description:

The floating-point multiply instructions multiply the f register(s) specified by the rs1 field by the f register(s) specified by the rs2 field, and write the product into the f register(s) specified by the rd field.

The FsMULd instruction provides the exact double-precision product of two single-precision operands, without underflow, overflow, or rounding error.

The floating-point divide instructions divide the f register(s) specified by the rs1 field by the f register(s) specified by the rs2 field, and write the quotient into the f register(s) specified by the rd field.

Traps:

fp_disabled

fp_exception (OF, UF, DZ (FDIV only), NV, NX,

invalid_fp_register (all except FMULs and FDIVs))

7.2.36 Floating-point Compare Instructions

opcode	**operation**
FCMPs	Compare Single
FCMPd	Compare Double
FCMPEs	Compare Single and Exception if Unordered
FCMPEd	Compare Double and Exception if Unordered

Suggested Assembly Language Syntax

fcmps	*fregrs1* ,	*fregrs2*
fcmpd	*fregrs1* ,	*fregrs2*
fcmpes	*fregrs1* ,	*fregrs2*
fcmped	*fregrs1* ,	*fregrs2*

Description:

These instructions compare the f register(s) specified by the rs1 field with the f register(s) specified by the rs2 field, and set the floating-point condition codes according to the following table:

fcc	*Relation*
0	*fregrs1* = *fregrs2*
1	*fregrs1* < *fregrs2*
2	*fregrs1* > *fregrs2*
3	*fregrs1* ? *fregrs2* (unordered)

The “compare and cause exception if unordered” (FCMPEs and FCMPEd) instructions cause an invalid (NV) exception if either operand is a signaling NaN or a quiet NaN. FCMP causes an invalid (NV) exception if either operand is a signaling NaN.

A non-FPop2 (non-floating-point-operate2) instruction must be executed between an FPop2 (FCMP or FCMPE) instruction and a subsequent FBfcc instruction. Otherwise, the result of the FBfcc is unpredictable.

Traps:

fp_disabled

fp_exception (NV, invalid_fp_register (all except FCMPs and FCMPEs))

7.3 BM3803 Software Considerrations

7.3.1 Register Window and procedure-call

Procedure-call

The general principles of procedure-call:

1. A CALL instruction can obtain the pointer which branch to PC entry address of the callee;

2. A callee uses the SAVE instruction to obtain a new register window and stack

space;

3. A callee uses the "RET; RESTORE" assembly instructions to return the caller.

Example 1: the Caller is main(), and the Callee is printf()

```
main:
        call printf             ! Call printf()
        nop                     ! Delay instruction
        nop                     ! Delay instruction
            ...
printf:
        save %sp, -1024, %sp        ! Obtain a new register window
                                    ! and the new stack space
            ...
        ret
        restore                     ! The callee returns to the main() position
```

Parameters passing

The general rules of parameters passing:

1. Up to six parameters may be passed by placing them in out registers %o0…%o5; additional parameters are passed in the memory stack.

2. The caller uses a CALL instruction to call the callee;

3. The callee uses a SAVE instruction to obtain a new register window and stack space. The callee finds its first six parameters in %i0 ... %i5, and the remainder (if any) on the stack.

4. The callee returns integer values by writing them into its ins (which are the caller's outs), starting with %i0.

Example 2:

```
int reg_window(int a1, int a2, int a3, int a4, int a5, int a6, int a7, int a8)
{
    int b;
    b=a1+a2+a3+a4+a5+a6+a7+a8;
    return b;
}
```

```
main()
{
    int c1=1, c2=2, c3=3, c4=4, c5=5, c6=6, c7=7, c8=8;
    int sum = 0;

    sum = reg_window(c1, c2, c3, c4, c5, c6, c7, c8);
    sum = sum + 1;
}
```

After compiling the above two functions:

```
<main>:
 save   %sp, -152, %sp
 mov    1, %o0
 st     %o0, [ %fp + -12 ]
 mov    2, %o0
 st     %o0, [ %fp + -16 ]
 mov    3, %o0
 st     %o0, [ %fp + -20 ]
 mov    4, %o0
 st     %o0, [ %fp + -24 ]
 mov    5, %o0
 st     %o0, [ %fp + -28 ]
 mov    6, %o0
 st     %o0, [ %fp + -32 ]
 mov    7, %o0
 st     %o0, [ %fp + -36 ]
 mov    8, %o0
 st     %o0, [ %fp + -40 ]
 clr    [ %fp + -44 ]
 ld     [ %fp + -36 ], %o0
 st     %o0, [ %sp + 0x5c ]      ！ Parameter c7, passing by the stack
 ld     [ %fp + -40 ], %o0
 st     %o0, [ %sp + 0x60 ]      ！ Parameter c8, passing by the stack
```

```
ld    [ %fp + -12 ], %o0        ! Parameter c1
ld    [ %fp + -16 ], %o1        ! Parameter c2
ld    [ %fp + -20 ], %o2        ! Parameter c3
ld    [ %fp + -24 ], %o3        ! Parameter c4
ld    [ %fp + -28 ], %o4        ! Parameter c5
ld    [ %fp + -32 ], %o5        ! Parameter c6
call    40001d30 <reg_window>
nop
st    %o0, [ %fp + -44 ]        ! From the output register

! obtain the return value b→sum
ld    [ %fp + -44 ], %o0
add    %o0, 1, %o1
st    %o1, [ %fp + -44 ]
ret
restore

<reg_window>:
save    %sp, -112, %sp
st    %i0, [ %fp + 0x44 ]      ! Input parameter, c1→a1
st    %i1, [ %fp + 0x48 ]      ! Input parameter, c2→a2
st    %i2, [ %fp + 0x4c ]      ! Input parameter, c3→a3
st    %i3, [ %fp + 0x50 ]      ! Input parameter, c4→a4
st    %i4, [ %fp + 0x54 ]      ! Input parameter, c5→a5
st    %i5, [ %fp + 0x58 ]      ! Input parameter, c6→a6
ld    [ %fp + 0x44 ], %o0
ld    [ %fp + 0x48 ], %o1
add    %o0, %o1, %o0
ld    [ %fp + 0x4c ], %o1
add    %o0, %o1, %o0
ld    [ %fp + 0x50 ], %o1
add    %o0, %o1, %o0
ld    [ %fp + 0x54 ], %o1
```

```
add    %o0, %o1, %o0
ld    [ %fp + 0x58 ], %o1
add    %o0, %o1, %o0
ld    [ %fp + 0x5c ], %o1      ！ Input parameter, c7→a7
add    %o0, %o1, %o0
ld    [ %fp + 0x60 ], %o1      ！ Input parameter, c8→a8
add    %o0, %o1, %o0
st    %o0, [ %fp + -12 ]
ld    [ %fp + -12 ], %o0
mov    %o0, %i0                  ！ return the value b
b    40001d9c <reg_window+0x6c>
nop
ret
restore
```

C and assembly language functions' mixed calling

The above principles must be followed about parameters-passing when C language and assembly language functions call each other. The differences are that we need determine the parameters' address and order manually.

Example 3, C program calls the assembly language function.

An assembly language function: asm_shift_function (p1, p2), there p1 is the data to be shifted, and p2 is the offset.

Here we use asm_shift_function () to move 1 left by 2 bits.

```
asm("
asm_shift_function:
  save %sp, -112, %sp
  sll %i0, %i1, %i0            ！ Obtain the input parameters from input registers
  ret
  restore
  ");

main()
{
```

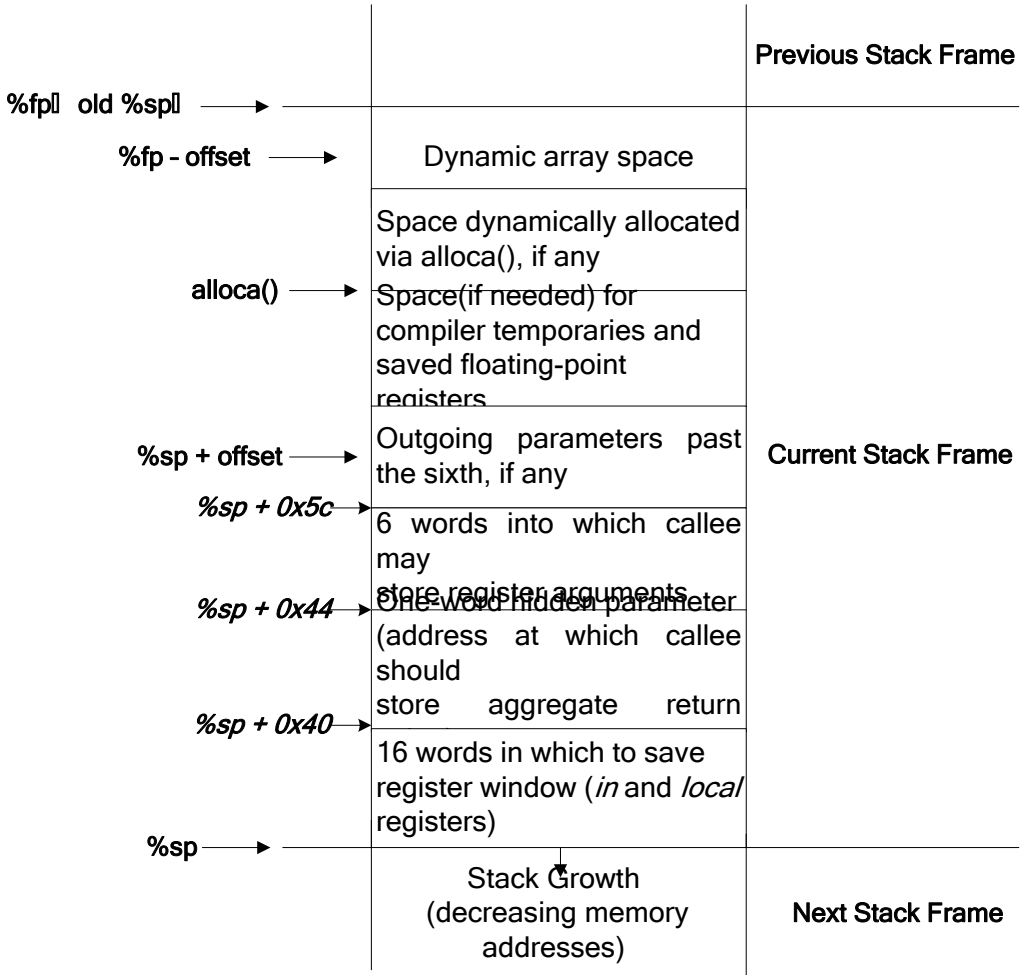
```
        int result = 0;
        result = asm_shift_function(1,2);              // input parameters
    }
    after compiling:
    <asm_shift_function>:              !  The compiler does not modify the assembly
                                       ! language programs.
     save    %sp, -112, %sp
     sll    %i0, %i1, %i0
     ret
     restore

    <main>:
     save    %sp, -152, %sp
     clr    [ %fp + -48 ]
     mov    1, %o0                     !  put the parameter into output register
                                       ! in accordance with the C code
     mov    2, %o1                     !  put the parameter into output register
                                       ! in accordance with the C code
     call    40001da4 <asm_shift_function>
     nop
     st    %o0, [ %fp + -48 ]          !  Obtain the function's return value
                                       ! from the output register
     ret
     restore
```

7.3.2 Stack

Stack description

According the SPARC V8 criterion, the stack is composed of a frame pointer (%fp) and a stack pointer (%sp), and it grows from the high address to the low address.

The stack is used as shown below:



Every time you allocate a register window, and also allocate a space as the stack. As the stack growing upwards, a program's first address is at the low-end of the ROM or RAM, and the stack's first address is at the high-end of the RAM in order to the highest utilization.

7.3.3 Traps and interrupts

Overview

BM3803 supports two types of traps:

- A precise trap (commonly known as a trap)
- An interrupting trap (commonly known as an interrupt)

Configurations about interrupts and traps are as follows:

1. Processor State Register (%PSR)

ET: the Trap Enable field

PIL: the Processor Interrupt Level field

2. Floating-point State Register (%FSR)

TEM: Floating-point trap enable mask field

FTT: Floating-point trap type field

AEXC: a floating-point exception has been generated

CEXC: the current floating-point exception

3. Trap Base Register (%TBR)

4. Interrupt control registers

Interrupt mask and priority register (0x8000 0090)

Interrupt pending register (0x8000 0094)

Interrupt force register (0x8000 0098)

Interrupt clear register (0x8000 009C)

5. a table of trap/interrupt allocation and priority

trap	Priority	trap type (tt)	Description
reset	1	0x00	Power-on reset
write_error	2	0x2B	write buffer error
instruction_access_error	3	0x01	Error during instruction fetch
privileged_instruction	4	0x03	execution of privileged instruction in user mode
illegal_instruction	5	0x02	UNIMP or other un-implemented instruction
fp_disabled	6	0x04	FP instruction while FPU disabled
cp_disabled	6	0x24	CP instruction while Co-processor disabled
watchpoint_detected	7	0x0B	hardware breakpoint match
window_overflow	8	0x05	SAVE into invalid window
window_underflow	8	0x06	RESTORE into invalid window
register_hardware_error	9	0x20	register file EDAC error
mem_address_not_aligned	10	0x07	memory access to un-aligned address
fp_exception	11	0x08	FPU exception
data_access_exception	13	0x09	access error during load or store

			instruction
tag_overflow	14	0x0A	Tagged arithmetic overflow
divide_exception	15	0x2A	Divide by zero
trap_instruction	16	0x80 − 0xFF	software trap instruction(TA)
interrupt_level_15	17	0x1F	reserved
interrupt_level_14	18	0x1E	PCI interrupt
interrupt_level_13	19	0x1D	reserved
interrupt_level_12	20	0x1C	UART3
interrupt_level_11	21	0x1B	DU trace buffer
interrupt_level_10	22	0x1A	reserved
interrupt_level_9	23	0x19	Timer2
interrupt_level_8	24	0x18	Timer1
interrupt_level_7	25	0x17	GPIO interrupt3
interrupt_level_6	26	0x16	GPIO interrupt2
interrupt_level_5	27	0x15	GPIO interrupt1
interrupt_level_4	28	0x14	GPIO interrupt0
interrupt_level_3	29	0x13	UART1
interrupt_level_2	30	0x12	UART2
interrupt_level_1	31	0x11	AHB bus error

Trap control

1. ET and PIL control

The ET bit in the PSR must be 1 for traps to occur normally. While ET = 1, the IU— between the execution of instructions — prioritizes the outstanding exceptions and interrupt requests according to the table of trap/interrupt allocation and priority. At a given time, only the highest priority exception or interrupt request is taken as a trap. Lower priority exceptions will be ignored. While ET = 0, the processor halts execution and enters the error_mode state if a trap occur.

While ET = 1, for interrupt requests, the IU compares the ILEVEL and IMASK fields of the interrupt mask and priority register against the processor interrupt level (PIL) field of the PSR. If ILEVEL and the interrupt number are the greatest, and the interrupt number is greater than PIL, then the processor takes the interrupt request trap when the IMASK field of the interrupt mask and priority register is 1. While ET = 0, all interrupts are ignored.

2. TEM control

While ET = 1, if the TEM field of the FSR is 1, the associated_exception can cause an fp_exception trap. Then the destination f register, FCC and AEXC fields remain unchanged. If the TEM field is 0, an fp_exception trap is masked. Instead，the occurrence of the exception is recorded in the FSR's AEXC field, and the destination f register and FCC field are updated to their new values.

Trap Identification

The trap base address (TBA) field of the trap base register (TBR) saves the upper 20 bits of the trap table address.

When a trap occurs (except for an external reset request), a value that uniquely identifies the trap is written into the 8-bit TT field of the TBR by the hardware. Control is then transferred into the supervisor trap table to the address contained in the 32-bit TBR. Since the low 4 bits of the TBR are zero, each entry in the trap table contains the first 16 bytes (4 words) of the corresponding trap handler.

Trap processing

A trap causes the following to occur, if ET = 1:

- Traps are disabled: ET←0.
- The existing user/supervisor mode is preserved: PS←S.
- The user/supervisor mode is changed to supervisor: S←1.
- The register window is advanced to a new window: CWP←((CWP-1) modulo 8) [note: without test for window overflow].
- The current PC and nPC are saved in local registers of the new window: %l1 ← PC, %l2 ← nPC.
- The tt field is written to the particular value that identifies the exception or interrupt request, except as defined for "Reset Trap" and "Error Mode" above.
- If the trap is a reset trap, control is transferred to address 0: PC←0, nPC←4. If the trap is not a reset trap, control is transferred into the trap table: PC ←%TBR, nPC←%TBR+4.

If ET = 0:

- A trap occurs, and then the processor enters the error_mode state and halts execution.
- An interrupt occurs, and then it is ignored.

Differences between the trap and the interrupt

- Causing position

  A trap occurs during the running of an instruction which causes the trap.

  An interrupt occurs after an instruction has completed.

- Control

  A trap is only controlled automatically by ET field or ET + TEM fields

  Besides ET field, an interrupt can be controlled by PIL field and four interrupt control registers.

7.3.4 Cache flushing

Overview

BM3803 can flush its cache, and tag bits of the cache are all cleared after flushed.

Flush methods

- Flush of the instruction cache (either will be able)

1. Execute the FLUSH instruction.
2. Set the 21st bit in the cache control register to 1.
3. Write to any location with ASI = 0x5.

- Flush of the data cache (either will be able)

1. Execute the FLUSH instruction.
2. Set the 22nd bit in the cache control register to 1.
3. Write to any location with ASI = 0x6.

- Flushing State

The data cache is flushing when the 14th bit in the cache control register is 1. And the instruction cache is flushing when the 15th bit in the cache control register is 1. Also 0 indicates flush is over or there is no flush.

Flush Steps

1. Disable the cache.
2. Flush the cache.
3. Judge the flushing state and wait for the end.
4. Enable the cache.

7.4 BM3803 software fault-tolerance

7.4.1 Overview

The regfile，cache and memory controller of BM3803 have fault-tolerant system. This chapter describes the use of these three modules' fault-tolerance.

7.4.2 Regfile EDAC

Working process

The BM3803 protects the regfile with EDAC that can correct one error and detect two or more errors.

The regfile is composed of a 32-bit data and a 7-bit EDAC checksum. When writing the regfile, first calculate the EDAC checksum of the data to write, and then write the regfile with data and its EDAC checksum. When reading the regfile, any single-bit error is corrected and written back to the regfile before the instruction is executed. If an un-correctable error is detected, a register hardware error trap (trap 0x20) is generated.

Registers Description

There are two related registers of the Regfile EDAC：%ASR16 and %ASR17. Details are as shown in the "Appendix4 Ancillary State Registers" section.

Basic Operations

- Correct and detect an error

1．%asr16 and %asr17 are configured in the initialization, clearing the "TCB" and "DCB" fields.

2．Enable or disable the "CB" field on demand. If you want to record the number of 1-bit errors, first clear the "CNT" field, and then enable the "CB" field.

- Error making

1． %asr16 and %asr17 are configured in the initialization, clearing the "TCB" and "DCB" fields.

2．Configure the "TCB" and "DCB" fields, with setting the error making bit to 1.

3．Enable the "CB" field of %asr16.

4．Write the error making register with correct data. Then the data in this register has made errors.

5．Disable the error making.

Example 1:

The data of %l1 makes an error. The correct data is 0x5555 5555. We make an error at bit 0, and then the error data is 0x5555 5554.

```
mov %g0, %asr16                 ! clear %asr16
mov %g0, %asr17                 ! clear %asr17
set 0x55555555, %o0
set 0xa0000000, %o1
set 0x1, %o2
mov %o2, %asr17                  ! write %asr17 with 0x1
! make an error at the bit 0
mov %o1, %asr16                  ! write %asr16 with 0xa000 0000
! That can make error
mov %o0, %l1                     ! data is written to% l1, make error
mov %g0, %asr16                  ! disable error making
mov %g0, %asr17                  ! clear
```

Example 2:

The data of %l1 makes two parity errors. The correct data is 0xaaaa aaaa, and its checkbit is 0x0c. We make error at 4th and 5th parity bits. Then the data is 0xaaaaaaaa, and the checkbits is 0x3c.

```
mov %g0, %asr16                  ! clear %asr16
mov %g0, %asr17                  ! clear %asr17
set 0xaaaaaaaa, %o0
set 0xa0300000, %o1
mov %o1, %asr16                  ! write %asr16 with 0xa030 0000
! make two parity errors
mov %o0, %l1                     ! data is written to% l1, made errors
mov %g0, %asr16                  ! disable error making
```

Notes

1. At the beginning, %asr16 is indefinite value after power-on. Before reading/writing regfile, error making must be disabled in %asr16.

2. Regfile EDAC cann`t be disabled.

3. Disable the error making of regfile at the beginning of the boot loader.

4. Before running, cann`t read the program which has made an error. And the error making program cannot be debugged.

5. The data-bit and parity-bit can make up to 2 errors. If more than 2, it may cause program errors.

7.4.3 Cache Parity-check

Working process

The BM3803 protects data and tag of the cache with parity-check, and can detect odd number of errors.

Data and tag of the instruction/data cache are both composed of data-bit and parity-bit. When writing the cache, first calculate the parity checksum of the data to write, and then write the cache with data and its parity checksum. When reading the cache with any error detected, data from external memory is read directly for the IU. Also the the correct data is written to the cache.

Registers Description

There are three related registers of the cache parity-check. Details are as shown in the “Appendix4 Cache Registers” section.

Basic Operations

- Error detection

1. Clear the CCR2 register and CCR3 register.
2. Write the CCR1 register with "0x0" to disable the error making.

- Error making

1. Disable the cache.
2. Configure the CCR1:
   (1) Write the CMEEN (Cache Make Error Enable) field with “011” to enable the error making.
   (2) Configure the CPSEL (Cache Parity Select) field to select the error making target: “00” implicates the error making target is the data of instruction cache, “01” implicates the error making target is the tag of instruction cache, “10” implicates the error making target is the data of

data cache, and “11” implicates the error making target is the tag of data cache.

3. Configure the EPOS (Error Position) field of CCR1 register and CCR2 register.

4. Enable cache.

5. Read data from external memory, then the data with error making will fill the cache automatically. So the error making is finished.

6. Disable the cache error making.

Example: make error of the data cache’s data area

Write data 0x1234 5678 to address 0x4001 0000. After making error at the 31st bit, there is 0x9234 5678.

```
set 0x12345678, %l0
set 0x40010000, %l1
st    %l0, [%l1]                ! write data 0x12345678 to address 0x40010000
set 0x40010000, %l0
set 0x80000000, %l1
set 0x1c00, %l2
set 0xc, %l3

st    %g0, [%l1 + 0x14]                    ! clear CCR, disable the cache
st    %g0, [%l1 + 0x114]                    ! clear CCR2
st    %g0, [%l1 + 0x118]                    ! clear CCR3

st    %l1, [%l1 + 0x114]        !write CCR2 to specify the error making bit is 31st
st    %l2, [%l1 + 0x110]        ! write CCR1 to implicate the error making target
                                ! is data area of data cache
st    %l3, [%l1 + 0x14]         ! write CCR to enable the data cache

ld    [%l0], %o0     ! read data from 0x4001000, and make error of the data cache

st    %g0, [%l1 + 0x14]                     ! disable cache
st    %g0, [%l1 + 0x110]                    ! disable the error making
st    %g0, [%l1 + 0x114]
```

```
st    %g0, [%l1 + 0x118]

lda [%l0] 0xf, %o1                    ! read data from the data area of data cache
```

Notes:

1. Cache parity-check is always enabled.

2. Use LOAD/STORE related instructions to make error, and not use the ASI related instructions.

3. When error making, please note how many instructions will be cached into the cache, especially the instruction cache. Since the continuity of reading instructions, it is difficult to make error of a single instruction.

7.4.4 EDAC of External Memory Controller

Overview

BM3803 implements an error detector and corrector (EDAC) of external memory that can correct one error and detect two or more errors. EDAC protection capabilities are as shown below:

Address Range	Area		EDAC Protected
0x00000000 - 0x1FFFFFFF	PROM	8 bits	yes
		16 bits	no
		32 bits	yes
0x20000000 - 0x3FFFFFFF	I/O	All	no
0x40000000 - 0x7FFFFFFF	RAM	8 bits	yes
		16 bits	no
		32 bits	yes

Registers Description

See Appendix4 Memory Interface Registers.

Basic Operation

- MECFG1 configuration

When enable EDAC, error detecting and correcting configurations are shown below:

EWB ERB	0	1

0	mecfg 1_1		mecfg 1_3	
	detector corrector	W: data-bit, check-bit R: data-bit, check-bit	detector no corrector	W:data-bit, check-bit R:data-bit, check-bit
1	mecfg 1_2			
	no detector no corrector	W: data-bit, check-bit R: data-bit, check-bit	no detector no corrector	W:data-bit, check-bit R:data-bit, check-bit

When disable EDAC, error detecting and correcting configurations are shown below:

EWB ERB	0		1	
0			mecfg 1_7	
	no detector no corrector	W: data-bit, check-bit R: data-bit, check-bit	no detector no corrector	W: data-bit, check-bit R: data-bit, check-bit
1	mecfg 1_6			
	no detector no corrector	W: data-bit, check-bit R: data-bit, check-bit	no detector no corrector	W: data-bit, check-bit R: data-bit, check-bit

Explain:

1. If set “0”, disables the EWB/ERB. And if set “1”, enables the EWB/ERB. (EWB enabled = 011).

2. mecfg1_x implicates a combination of "EDAC, EWB, ERB". Such as: mecfg1_1 implicates "EDAC enables, EWB disables, ERB disables."

3. "W" means write memory. "R" means read memory.

4. The word in a box implicates "error-making of the write data” when write, or "cannot obtain the data" when read. For example:

“W: data-bit” means making error of the data-bit when writing.

“R: check-bit” means that check-bit cannot be obtained when reading.

5. Example:

mecfg1_7 configuration:

EDAC = 0, EWB = 001, ERB = 0

It means no detector and no corrector.

Make errors of data-bit and check-bit when writing.

Only data-bit can be obtained when reading and check-bit cannot.

- Error detecting and correcting

1. Clear the MECFG2 and MECFG3 to disable the error-making.

2. Configure the MECFG1 as mecfg1_1.

- Error-making

1. Configure the MECFG2 and MECFG3 to make an error at one bit of data-bit or check-bit.

2. Configure the MECFG1 as mecfg1_7.

3. When writing memory, the specified bit will upset according to the configurations of the MECFG2 and MECFG3.

Example:

Write data 0x8765_4321 to the SDRAM address 0x6000_0000, and make an error at bit 0 of the data. Then there is 0x8765_4320.

```
set 0x60000000, %l0
set 0x87654321, %l1
set 0x01, %l2
set 0x80000100, %l3
set 0x001c0000, %l4              ! mecfg 1_7
st   %g0, [%l3 + 0x08]           ! Clear MECFG3
st   %l2, [%l3 + 0x04]           ! Configure MECFG2 with making error at bit 0
st   %l4, [%l3]                  ! Configure MECFG1 to make an error
st   %l1, [%l0]                   ! make error
st   %g0, [%l3]                   ! Clear MECFG1
st   %g0, [%l3 + 0x04]            ! Clear MECFG2
st   %g0, [%l3 + 0x08]            ! Clear MECFG3
```

Notes

1. There are only 7 pins of chesk-bit in the BM3803 when using the 32-bit PROM. Cannot write check-bit when using 8-bit PROM.

2. Only a quarter of a bank's check-bit can be read. Especially note: the address of the data whose check-bit will be read and the current PC (Program Counter) should be in the different quarters of a bank. Otherwise, the check-bit will be covered with the sequence of instructions' check-bit.

Appendix8 Avoid common problems by software

8.1 How to handle the IEEE-754_exception of floating-point

Problem

There are two ways to handle the IEEE_754_excption of the BM3803FMGRH. One way is masking the five IEEE_754_excptions in FSR. When an exception occurs, there is no handler. But the CEXC field of the FSR is set, and the instruction continues being executed. The other way is enabling the five IEEE_754_excptions in the FSR. When an exception occurs, there is a handler. The instruction is interrupted to have no result. And the CEXC field of the FSR won't be changed.

Solution

To ensure the reliability of the system, there are two solutions of the BM3803FMGRH.

1）After masking the 5 IEEE_754_excptions in the FSR, we prejudge the floating-point data by software to ensure no IEEE_754_exception happens during floating-point operations. Take divided_by_zero as an example: firstly we prejudge the divisor, if it is 0, the division won't be executed.

2）When enableing the 5 IEEE_754_excptions in the FSR, if the CEXC field of the FSR need to be set, the software can do it in the trap handler.

8.2 How to handle the double precision floating-point operation problem of specific instructions sequence

Problem

The following instructions sequence is in application:

```
ld [%l1],%f2
```

```
nop                                    （interrupt service routine）
ld [%l2],%f3
faddd %f4,%f2,%f4
```

When running in BM3803FMGRH, the result of double precision floating-point operation faddd is wrong.

On analysis, there are 6 double precision floating-point operation problems of specific instructions sequence:

case1:

```
LD [%rn], %fx+1
FPOPd(1) %fx, %fy, %fx
```

case 2:

```
LD [%rn], %fx+1
FPOPd(1) %fy, %fx, %fx
```

case 3:

```
LD [%rn], %fx+1
FPOPd(1) %fx, %fy, %fy
```

case 4:

```
LD [%rn], %fx+1
FPOPd(1) %fy, %fx, %fy
```

case 5:

```
LD [%rn], %fx+1
FPOPd(2) %fx, %fx, %fx
```

case 6:

```
LD [%rn], %fx+1
FSQRTd %fx, %fx
```

Note: 1. FPOPd(1) is one of FADDd、FSUBd、FMULd and FDIVd;

2. FPOPd(2) is one of FADDd and FMULd.

Solution

Avoid affecting the application of the system, the problems can be solved by eliminating the specific instructions sequence in the software. We can change the order of the instructions to ensure there is no the specific instructions sequence.

According to different programming modes, problems can be solved as follows:

When programming in advanced language, we check the compiled and linked file by binary inspection tool to find the instruction-sequences that may cause the problem. Then we exchange the two ld instructions before the double precision floating-point instruction. If there is only one ld instruction (or there is a static code between the two ld instructions) before the double precision floating-point instruction in the former 6 cases, choose a suitable instruction before the double precision floating-point instruction to exchange with this ld instruction.

1）original error instructions sequence:

```
ld [%l1],%f2
nop                          （interrupt service routine）
ld [%l2],%f3
faddd %f4,%f2,%f4
```

change the instructions sequence to:

```
ld [%l2],%f3
nop                          （interrupt service routine）
ld [%l1],%f2
faddd %f4,%f2,%f4
```

2）when programming in assembly language, please add a nop instruction between two ld instructions and the double precision floating-point instruction.

original error instructions sequence:

```
ld [%l1],%f2
nop                          （interrupt service routine）
ld [%l2],%f3
faddd %f4,%f2,%f4
```

change the instructions sequence to:

```
ld [%l1],%f2
nop                          （interrupt service routine）
ld [%l2],%f3
nop
faddd %f4,%f2,%f4
```

3）original error instructions sequence:

```
sethi    %hi(0x40001800), %g1
```

```
st    %o1, [ %fp + -16 ]
or    %g1, 0x230, %g1
sll    %o2, 3, %o2
ldd    [ %g1 + %o2 ], %f10
ld    [ %fp + -16 ], %f9
faddd    %f10, %f8, %f8
```

change the instructions sequence to:

```
sethi    %hi(0x40001800), %g1
st    %o1, [ %fp + -16 ]
or    %g1, 0x230, %g1
sll    %o2, 3, %o2
ld    [ %fp + -16 ], %f9
ldd    [ %g1 + %o2 ], %f10
faddd    %f10, %f8, %f8
```

Appendix9 Cautions

1. I/O pin voltage 3.3V.
2. Main frequency: 1MHz-100MHz, input PLL frequency range: 2MHz-30MHz。
3. Reset order: firstly reset the peripheral circuit, then this circuit。
4. Keep the reset signal 10 clock cycles when reset by the external reset pin.
5. Strictly follow the power supply sequence in this manual, power I/O prior to the processor core.
6. The input pins can't left hanging. If the input pins haven't been pulled up or down (see Appendix1) inside the circuit, and they are unused, please pull them up or down according to their function, and notice the software engineer and system design engineer of that.
7. If the output pins are unused, never connect them to other pins, including VDD and ground.
8. When the processor is working in operation mode, DUBRE and DUEN signal must be set 0, DURX must be set 1.
9. In SRAM, PROM, I/O read timing, the address value of "lead_out" cycle is not retain the address at last cycle but be the address of the processor internal bus at last cycle.(see Appendix2)
10. When both SDRAM and SRAM are enabled, and SDRAM bank size is 512MB, SDCSN[1] should be connect to chip selects of SDRAM.
11. When enable EDAC, SDRAM don't support access in burst mode of internal AHB bus. So do the following to avoid burst transfer mode when access SDRAM with EDAC enabled:
    (1).Turn off burst function in cache control register（0x8000-0014）.
    (2)Avoid PCI operation in burst transfer mode。
12. Ancillary State Register ASR16 is default when power on, so it must be set to disable EDAC before read. Never read/write Regfile when ASR16 enable EDAC.
13. Never visit no response address, ie: 0xa0000000~0xbfffffff
14. If write protection need to use segment mode, that is to say, set BP to 0, only one write protection register can be used at a time, the other one can not use segment

mode at the same time.

15. Instruction LDDx launches burst transfer on AHB bus.

16. PCI can't operate by byte or half-word, the address must be word-aligned.

17. To ensure no external interrupt happen while writing power-down register, all external interrupt must be cleared before start up power-down mode.

18. The circuit is product by CMOS technics, attention should be paid to electrostatic protection when using.

Service and Support：

Address: No.2 Siyingmen N. Road. Donggaodi. Fengtai District. Beijing, China.
Department: Department of international cooperation
Telephone: +86(0)10-67968115-6751
Email: gjhz@mxtronics.com
Fax: +86(0)10-68757343
Zip code: 100076