Ver2.4.2

BM3803MG

32 位空间处理器

用户手册

北京微电子技术研究所 2010 年 8 月

目 录

一 概述和总体结构

1.1 概述

BM3803MG是基于SPARC V8体系结构的32位RISC嵌入式处理器，可用于板上嵌入式实时计算机系统，能够满足各种航天应用的功能以及性能指标要求，只要加上存储器和与应用相关的外围电路，就可以构成完整的单板计算机系统。

BM3803MG芯片内部包含整数处理单元，浮点处理单元，独立的指令和数据Cache，硬件乘法器和除法器，中断控制器，带有跟踪缓冲器的硬件调试单元，两个24位定时器，通用I/O接口，看门狗，能够支持PROM、SRAM、SDRAM和I/O映射空间访问的存储器控制器，具有软件可以控制的省电工作模式，具有可实现 PCI主机桥（Host bridge）和从属桥（Guest bridge）功能的PCI控制器。

1.2 主要性能

- 处理单元特性： BM3803MG处理器是基于SPARC V8的32位微处理器， 具有8 个寄存器堆窗口、5级流水线、16K字节大小的两路组相联数据Cache、32K字节大小的四路组相联指令Cache、具有支持单双精度浮点数据类型的浮点处理单元。
- 片上外设：具有支持PROM、SRAM、SDRAM和I/O映射空间访问的存储器控制器，具有两个24位的定时器，一个看门狗，三个串行通信接口，以及4个外部可编程输入端口的中断控制器，32个通用I/O接口，符合PCI2.3规范的33MHz PCI接口。
- 容错设计：完全的三模冗余设计、 EDAC和奇偶校验
- 调试测试：用于调试跟踪的硬件调试单元，具有4个硬件观测点。
- 工作范围： I/O电源电压： 3.3V +/- 0.30V；核心电源电压：1.8V +/- 0.15V；（两电源可同时上电）
- 工作温度：-55～+125℃。
- 工作频率：0MHz～100MHz。
- 功耗：1W/100MHz。
- 主要性能指标： 86MIPS@100MHz （Dhrystone 2.1），23MFLOPS@100MHz（Whetstone）。

- 抗辐性能： 总剂量抗辐能力100Krad（Si）， 优于1 E-5错误/器件/day的SEU事件，优于70 MeV cm^2/mg的抗闩锁能力。
- 开发工具： BM3803MG 评估板（Evaluation Board），SPE 软件集成开发环境。

总体结构框图如图 1-1 所示。

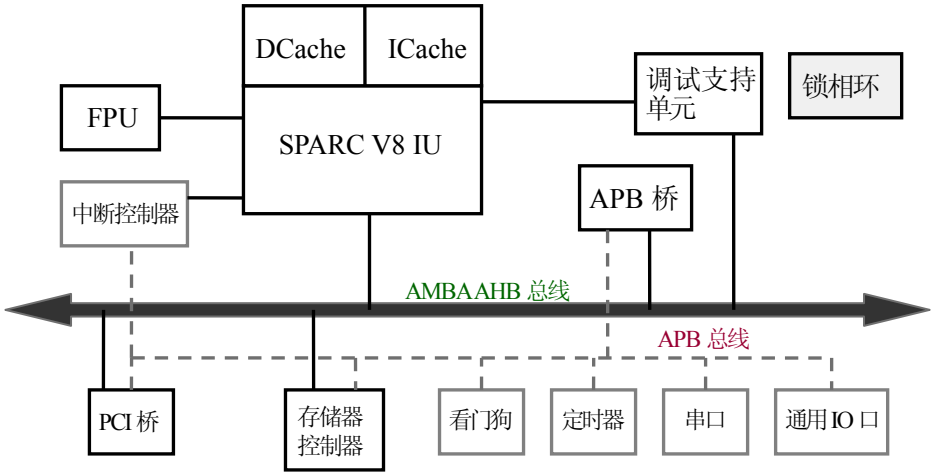


图 1-1 BM3803MG 总体结构框图

二 处理器核心 IU

2.1 概述

处理器核心 IU 实现了 SPARC V8 标准定义的整数指令集，特性如下：

- 5 级指令流水
- 8 个寄存器窗口
- 硬件乘/除法器
- 寄存器三模冗余结构，寄存器堆 EDAC 容错

2.2 指令集

BM3803MG 指令分为 6 种功能类别：加载/存储，算术/逻辑/移位，控制转移，读/写控制寄存器，浮点操作及其它指令。参见 SPARC V8 结构手册所列的所有实现的指令。

表 2- 1　BM3803MG 的指令执行周期

指令	周期
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

2.3 浮点处理单元（FPU）

浮点处理单元可以处理单精度数和双精度数，实现了所有的 SPARC V8 浮点指令。浮点指令的执行与 IU 串行，当 FPU 中指令没有执行完时，IU 处于等待状态。这意味着%fsr 寄存器中的 QNE 位一直为 0，而任何试图执行 STDFQ 指令的操作都会产生一个浮点执行异常。

2.4 IU 相关的内部寄存器

处理器状态寄存器 PSR（Processor State Register）

位域	31-28	27-24	23	22	21	20	19-14	13	12	11-8	7	6	5	4..0
名称	IMPL	VER	N	Z	V	C	保留	EC	EF	PIL	S	PS	ET	CWP
读写	R	R	R/W				R/w	R	R	R/W	R/W			R/W
初始值	B	3					0x20	0	1					

- IMPL（31:28）：处理器实现的等级，只读。
- VER（27:24）：版本号，与实现特性相关。（IMPL 和 VER 为 0xB3）
- ICC（23:20）：整数单元的条件码，被以字母 CC 结尾的指令操作。N，表示 ALU 的逻辑运算单元的 32 位结果是否为负数；Z，表示运算结果是否是零；V，表示运算结果是否超出 32 位数的范围，1 表示溢出；C，表示运算的结果是否引起进位。
- EC（13）：表示是否使能协处理器。
- EF （12）：表示是否使能浮点处理器。
- PIL（11:8）：指定处理器将要接受陷阱的级别。
- S（7）：制定处理器处于用户态还是管理态，S 表示处于管理员状态；
- PS（6）：保留最近一次中断发生时 S 位中的值；
- ET（5）：当 ET=0 时禁止陷阱，当 ET=1 时表示陷阱可用。
- CWP（4:0）：当前窗口指针，可以在 r 寄存器中识别当前寄存器窗口的计数器。

本处理器的初始值是：0xb302-1080；

窗口无效屏蔽寄存器（WIM Window Invalid Mask Register， WIM）

窗口无效屏蔽寄存器，IU 利用 WIM 来决定一条 SAVE、RESTORE 或者 RETT 指令是否会产生窗口下溢和溢出陷阱。

W31	W30	W29	……	W2	W1	W0

对于 IU 中的每一寄存器窗口，都对应一个有效状态位，即 WIM[n] 对应着 CWP=n 的寄存器组。可以通过指令 RDWIM 来读取 WIM 的值，也可以通过 WRWIM 指令对 WIM 进行写操作。没有实现的窗口位始终是‘0’，因此可以通过执行 WRWIM 指令将所有的 WIM 值置 1，然后执行 RDWIM 指令，得到位向量，从而判断实现窗口的数目。（由于对特殊

寄存器的流水处理中没有数据前推，因此对于特殊寄存器的访问，应该在读写之间插入 3 条 nop 指令。）

BM3803MG 具有 8 个寄存器窗口，因此 W31-W8 的值始终为零。W0-W7 初始值为全零，但会随着程序的运行而变化。

异常基址标志寄存器 TBR（Trap Base Register）

异常基址标志寄存器包含有 3 个字段。这三个字段构成了控制转移的目的地址。

TBA（31-12）	TT（11-4）	ZERO（3-0）

- TBA(31:12)：陷阱的基地址，这个基址由软件设置，包含高位的 20 位有效位和陷阱地址表，通过 WRTBR 指令实现对 TBA 字段的读写。
- TT(11:4)：陷阱类型字段，当陷阱发生时硬件对这 8 个字段进行写入，保持值不变，直到下一个陷阱发生。
- ZERO：全零，便于以后扩展使用。

初始值为全零。

乘法/除法寄存器（Multiply/Divide Register，Y）

32 位 Y 寄存器的内容为整数乘法双精度结果的最高有效字，作为执行整数乘法指令（SMUL，SMULcc，UMUL，UMULcc）的结果或用到整型乘法步指令(MULScc)的例行程序的结果。Y 寄存器还保存一个整型除法指令(SDIV, SDIVcc, UDIV, UDIVcc)的双精度被除数的最高字。

Y 寄存器由 RDY 和 WRY 指令读写。Y 寄存器初始值为 0。

辅助状态寄存器（Ancillary State Registers， ASR）

SPARC 提供了 31 个辅助状态寄存器（ASRs），编号从 1 到 31。

为了将来的使用，体系结构将编号 1 到 15 的 ASR 保留，软件无法访问这些寄存器。

编号 16 到 31 的 ASR 对实现相关的应用，比如计时器、计数器、诊断寄存器，自测寄存器和陷阱控制寄存器有效。对任意一个 ASR 存取的语义都是实现相关的。一个特殊的辅助状态寄存器是否为特权寄存器是实现相关的。

可以通过指令 RDASR 和 WRASR 对辅助状态寄存器进行读写操作。若被存取访问的寄存器为特权寄存器，则读/写辅助状态寄存器的指令是

特权指令。

ASR16 和 ASR17 用于 IU 的诊断测试。(详细介绍参见第二章第八节)

ASR24 到 ASR31 是用于调试控制的断点寄存器。(详细介绍参见第八章调试节口关于断点寄存器的介绍)

浮点状态寄存器 FSR （Floating Point State Register）

31-30	29-28	27-23	22	21-20	19-17	16-14	13-12	11-10	9-5	4-0
RD	保留	TEM	NS	保留	VER	FTT	保留	FCC	AEXEC	CEXEC

- RD(31:30)：舍入方向，根据 ANSI/IEEE 的 754-1985 标准选择 FPU 浮点算术操作时的舍入方向。

表 2-2 FSR 的舍入方式选择（RD）字段

RD	Round Toward:
0	Nearest (even, if tie)
1	0
2	+∞
3	−∞

- TEM(27:23)：异常使能标志，使能由 FPops 引起的异常。这些值同 cexc（当前异常域值）相与以决定是否对 IU 产生浮点异常。所有使能域的命名与 cexc 中一致，0＝不使能异常，1＝使能异常。
- NS(22)：使 FPU 产生详细定义的所要实现的结果，该结果可能不符合 IEEE 754-1985 标准。例如，为了获得更优的性能，当 NS 置位时，将欠规格化的浮点操作数或结果转换为 0。
- VER(19:17)：识别一个或多个特定的 FPU 结构实现版本，在 SPARC IU 实现中，可能有一个或多个 FPU，或者没有。该域值识别了特定的 FPU 实现版本。
- FTT(16:14)：浮点异常类型位，识别浮点异常类型，ftt 域值编码浮点异常直到指令 STFSR 或者另一个 FPop 指令执行。
- FCC(11:10)：包含 FPU 条件代码。这些位由浮点比较指令（FCMP，FCMPE）更新，由指令 STFSR，LDFSR 读写，FBfcc 以它的控制传输为基础。
- AEXEC(9:5)： fp_exception 异常不使能 TEM 时，积存 IEEE 浮点异常。在 FPop 执行完后， TEM 和 cexc 值被逻辑与，结果非 0，产生 fp_exception 异常，否则，新的 cexc 值与 aexc 相或并保存在 aexc 中。

因此，当屏蔽 trap 时，aexc 中的异常积存。

- CEXEC(4:0)：表明最近执行的 Fpop 指令引起的一个或多个 IEEE 浮点异常。异常不发生，对应位清零

2.5 处理器的启动方式

处理器初始化的时候，PC 指针的值设定为 0x0000-0000，因此程序将从这一地址空间开始执行程序。在整个 BM3803MG 的设计中，这一地址也是 PROM 的地址，因此程序的启动应该从 PROM 的 0x0000-0000 地址开始执行。如果从外部 PROM 启动，需要设置相应 PROM 的存储器数据宽度，通过设置 PIO[1:0]两位的输入来表明。“00”表示 8 位，“01”表示 16 位；“1X”表示 32 位。当使用 8 位 PROM 引导时，如果使用了 PROM 的 EDAC 功能，那么要在启动前使用外部管脚 ROMBSD[0..3]来确定 rom bank 的大小。

另一种启动方式，在 Reset 信号之后，如果 DSUBRE 和 DSUEN 信号有效，则 BM3803MG 处理器处于从 DSU 启动的调试方式，此时处理器处于调试模式，并且等待接受来自 DSU 的调试命令。

2.6 处理器的复位

BM3803MG 处理器的外部复位引脚保持一个时钟周期，就可以使得处理器进入复位。处理器进入复位之后，经过 7 个时钟周期后，处理器给出 PROM 的片选、地址和读使能信号。

2.7 异常

BM3803MG 处理器的陷阱模型以及优先级：

Trap	TT	优先级	说 明
reset	0x00	1	上电复位
write error	0x2B	2	写buffer错误
instruction_access_error	0x01	3	取指时出错
illegal_instruction	0x02	5	UNIMP 或其他未实现指令
privileged_instruction	0x03	4	用户模式下特殊指令的执行
fp_disabled	0x04	6	不使能FPU时使用浮点指令

cp_disabled	0x24	6	不使能协处理器时使用协处理器指令
watchpoint_detected	0x0B	7	硬件断点匹配
window_overflow	0x05	8	SAVE 进入无效窗口
window_underflow	0x06	8	RESTORE 进入无效窗口
register_hardware_error	0x20	9	寄存器堆EDAC错误
mem_address_not_aligned	0x07	10	访问存储器时地址未对齐
fp_exception	0x08	11	FPU 异常
cp_exception	0x28	11	协处理器异常
data_access_exception	0x09	13	load /store 指令访问错误
tag_overflow	0x0A	14	Tagged 运算溢出
divide_exception	0x2A	15	被0除
interrupt_level_1	0x11	31	异步中断 1
interrupt_level_2	0x12	30	异步中断 2
interrupt_level_3	0x13	29	异步中断 3
interrupt_level_4	0x14	28	异步中断 4
interrupt_level_5	0x15	27	异步中断 5
interrupt_level_6	0x16	26	异步中断 6
interrupt_level_7	0x17	25	异步中断 7
interrupt_level_8	0x18	24	异步中断 8
interrupt_level_9	0x19	23	异步中断 9
interrupt_level_10	0x1A	22	异步中断 10
interrupt_level_11	0x1B	21	异步中断 11
interrupt_level_12	0x1C	20	异步中断 12
interrupt_level_13	0x1D	19	异步中断 13
interrupt_level_14	0x1E	18	异步中断 14
interrupt_level_15	0x1F	17	异步中断 15
trap_instruction	0x80-0xFF	16	软件异常指令 (TA)

2.8 容错设计

IU 的容错设计包括两部分：所有寄存器采用三模冗余结构；存储器单元部分寄存器堆（Regfile）采用 EDAC。

BM3803MG 对寄存器堆中的数据采用 Hamming 码进行 EDAC 保护，可以检测多位错，纠正一位错。当监测到可以纠正的数据错误时，流水线断流，将纠正后的数据写按原读出的地址写回寄存器堆，之后，重启流水线，从断流位置重新运行程序；当监测到不可纠正的数据错误时，采用了陷阱方式处理，引起寄存器硬件错陷阱。

- ASR16 [31:28] EDAC 的控制位：
  “0000”，进行计数；
  “0100”，是错误清零；
  “1010”，打开造错功能，包括校验造错和数据位造错；
- ASR16 的[27:24]：保留位；
- ASR16 的[22:16]：为校验位造错的数据，可读写；
- ASR16 的[15:0]：单位错的计数值，写被忽略；
- ASR17 的[31:0]：数据位造错，可读写。

三 Cache 存储器

3.1 概述

BM3803MG 处理器采用哈佛结构，具有独立的指令总线和数据总线，分别与各自的 Cache 控制器相连。为了提高核心处理器的速度，对指令 Cache 和数据 Cache 都采用了多路组相联（multi-set-caches）技术。Cache 的替换标准基于 LRU（最近最少使用）算法。当有新数据要存储到 Cache 中时，替代 Cache 中的 LRU 部分。Cache 空间的大小是数据 Cache 16K 字节，指令 Cache 32K 字节。

3.2 Cache 映射

主存储器中的大部分区域能通过 Cache 空间存取。可缓存的地址空间包括 PROM 和 RAM 块，表 3-1 给出了处理器的缓存能力。

表 3- 1 Cache 的存储器的地址空间列表

地址范围	区域	能否 Cache
0x0000-0000-0x1FFF-FFFF	PROM 区域	可以
0x2000-0000-0x3FFF-FFFF	I/O 区域	不可以
0x4000-0000-0x7FFF-FFFF	RAM 区域	可以
0x8000-0000-0xFFFF-FFFF	内部	不可以

操作

在正常操作时，处理器访问指令和数据应用 SPARC 标准定义的 ASI 0x8-0xB。利用 LDA/STA 指令，可以访问不同的地址空间。在地址映射中，只有 ASI[3:0]有用，ASI[7:4]不起作用。

- 操作 ASI 0-3 强制 Cache 不命中，如果数据以前缓存过则更新 Cache，如果数据不在 Cache 中并且地址指向了可缓存区则分配新块。
- 操作 ASI 4 和 7 强制 Cache 不命中，如果数据以前缓存过则更新 Cache。

表 3-2 BM3803MG 的 ASI 实现

ASI	**Usage**
0x0, 0x1, 0x2, 0x3	强制cache不命中(若数据已缓存则进行替换)
0x4, 0x7	强制cache不命中(命中则更新)
0x5	刷新指令cache
0x6	刷新数据cache

0x8, 0x9, 0xA, 0xB	正常cache访问 (若数据已缓存则进行替换)
0xC	指令cache tags
0xD	指令cache tags
0xE	数据cache tags
0xF	数据cache data

3.3 Cache 的控制

指令 Cache 操作由 Cache 控制寄存器（CCR）控制。可以通过地址 0x8000-0014 读写该寄存器。CCR 各位的定义如下所示：

31:30	29:28	27:26	25:24	23	22	21	20:17	16	15	14	13:6	5	4	3:2	1:0
LRP	IRP	ISTS	DSTS	DS	FD	FI	保留	IB	IP	DP	保留	DF	IF	DC	ICS
r	r	r	r	rw	rw	rw		rw				rw	rw	rw	rw

- [31:30]：（只读），数据 Cache 的替换策略；
- [29:28]：（只读），指令 Cache 的替换策略；
- [27:26]：（只读）指令 Cache 的相联数；
- [25:24]：（只读）数据 Cache 的相联数；
- [23]：数据 Cache 的 Snoop 使能功能位；
- [22]：Flush 数据 Cache。若置位将 flush 数据 Cache；
- [21]：Flush 指令 Cache。若置位将 flush 指令 Cache；
- [20: 17]：保留；
- [16]：指令 burst 取，若置位将使能指令取指；
- [15]：指令 Cache flush 暂停，当这一位设置时指令 Cache 的 Flush 操作将暂停；
- [14]：数据 Cache flush 暂停；
- [13: 6]：保留；
- [5]：当中断来到时数据 Cache 冻结；
- [4]：当中断来到时指令 Cache 冻结；
- [3:2]： 数据 Cache 的状态，表明当前数据 Cache 的状态。
  “01”Cache 冻结、“11”Cache 使能，“X0”：禁止。
- [1: 0]： 指令 Cache 的状态，表明当前指令 Cache 的状态。
  “01”Cache 冻结、“11”Cache 使能，“X0”：禁止。

与 Cache 相关的控制寄存器还有 Cache 容错控制寄存器 CCR1

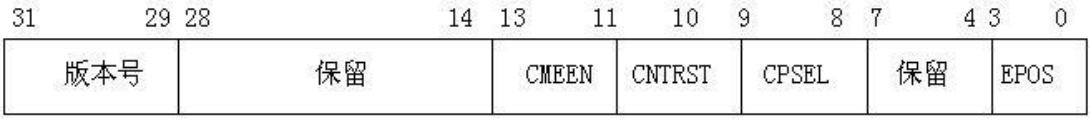
（0x8000-0110），Cache 造错寄存器 CCR2（0x8000-0114），Cache 错误计数寄存器 CCR3（0x8000-0118）三个寄存器。这三个控制寄存器主要和 Cache 的容错功能有关，寄存器中各位的详细说明参看 Cache 的容错功能模块。（本章第四节）

3.4 Cache 的容错功能控制

对于 Cache 部分，采用了奇偶校验的容错策略，当奇偶校验出错时，直接从外部存储器中读取数据。

Cache 容错功能的诊断主要通过设置扩展 Cache 控制寄存器实现。主要功能包括对不同 Cache 区域的计数选择，测试 Cache 区域容错的功能。

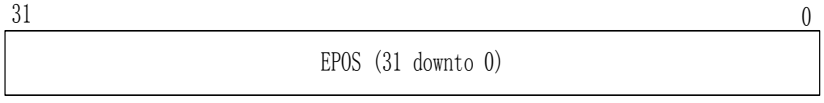
CCR1寄存器（地址为0x8000-0110）



CCR1

- [31:29]：Cache版本号寄存器，该三位只读不写，值是001。
- [28:14]：保留位。
- [13:11]：CMEEN(Cache Make Error Enable)，Cache造错使能标志位。"011"表示造错开启，其他值造错无效。
- [10]：CNTRST(CNT Reset)，Cache校验计数器清零标志。值为1时，清除所有错误计数器。
- [9:8]： CPSEL(Cache Parity Select)，Cache制造错误的目标区域选择。"00"表示指令Cache的data区；"01"表示指令Cache的tag区；"10"表示数据Cache的data区；"11"表示数据Cache的tag区；
- [7:4]： 保留位。
- [3:0]： EPOS(Error Position)，Cache造错位置高4位。当目标区域选择为data区，[3:0]对应数据的4个校验位；当目标区域选择为tag区，[3:0]：对应tag的4个校验位；

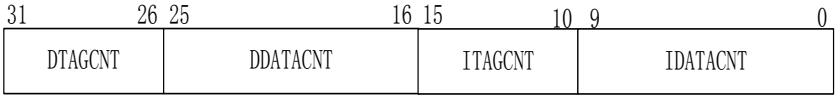
CCR2寄存器（地址为0x8000-0114）



CCR2

- [31:0]：造错位置标识低32位。如果Cache造错使能，且目标区域为data区，则[31:0]对应data[31:0]；如果目标区域为tag区，则按照下图确定：

CCR3寄存器（地址为0x8000-0118）



CCR3

CCR3是错误计数器，只读不可写。其中[31:26]位是 DCache的tag错误计数器；[25:16]是DCache的data错误计数器；[15:10]是ICache的tag错误计数器。[9: 0]是ICache的data错误计数器。

四 中断和陷阱

4.1 概述

BM3803MG 的异常处理机制支持中断和陷阱两种异常。陷阱是由与特定指令相关的硬件引起的异常，在引起异常产生的指令运行期间发生；中断是由处理器的外部事件产生的，该异常的产生不与任何特定的指令相关，并在指令执行期间发生。

异常的控制

异常被几个寄存器控制：陷阱和外部中断请求通过处理器状态寄存器（PSR）的使能区（ET），外部中断请求通过 PSR 的处理器中断等级区（PIL），浮点异常通过浮点状态寄存器（FSR）的陷阱使能区（TEM）分别控制。

异常要被正常处理，处理器状态寄存器（PSR）中的 ET 位必须为 1。ET=1 时，IU 在执行指令期间根据表 4-1 和表 4-2 区分未解决的异常和中断请求的优先次序来响应异常，在给定的时间内只有最高优先级的陷阱或中断请求被作为异常响应，当有多个未解决陷阱或外部中断请求时，较低特权的异常请求会保持，而如果产生异常的指令被重复执行的话，较低优先级的异常会重复发生。

对中断响应来说，IU 比较当前中断请求等级（bp_IRL）与 PSR 中的处理器中断等级（PIL）区域。如果 bp_IRL 大于 PIL（未屏蔽），此时处理器响应中断请求——假定没有更高的优先级未处理异常。

当 ET=0 时，中断异常不能发生，中断请求被忽略，即使 bp_IRL=15。

在存在挂起的陷阱异常的前提下，尝试执行会产生陷阱的指令，处理器将停止执行并进入错误模式状态。

如果 ET=1 时，从系统硬件上来讲，异常将会使以下事件自动发生：

- ✓ 再次异常被禁止：ET<—0。
- ✓ 目前的用户/管理模式被保留 PS<—S。
- ✓ 用户/管理模式被转换为管理模式 S <—1。
- ✓ 寄存器窗口前进到新的窗：CWP <—((CWP-1) modulo NWINDOWS)[注意：不检查寄存器窗口是否溢出]
- ✓ 被陷阱捕获的程序计数器存入新窗口中的局部寄存器%l1 和%l2：

r[17]<—PC, r[18]<—nPC。

- ✓ 除了 “复位异常”和“错误模式”，tt 区被写入一特定值以区别异常请求的类型。
- ✓ 如果异常是复位异常，则控制被转移到地址 0：PC<—0, nPC<—4；不是复位则 PC <—TBR, nPC<—TBR+4。因此异常处理就是根据特定的异常向量表来完成程序的跳转，异常向量表中存有异常处理的前四条指令，而异常处理表的基地址（TBR）是由软件建立的，在表中的跳转是由异常类型(TT)决定的。

4.2 陷阱

陷阱类型

BM3803MG 实现的陷阱类型符合 SPARC V8 规范，主要包括如下陷阱：

表 4-1 陷阱的类型

陷阱	TT（类型）	描述	优先级
reset	0x00	上电复位	1
write error	0x2b	写缓冲错误	2
Instruction_access_exception	0x01	取指时不可更正的EDAC错误	3
illegal_instruction	0x02	UNIMP或其他未实现的命令	5
privileged_instruction	0x03	用户模式下执行特权指令	4
fp_disabled	0x04	不使能FPU执行浮点指令	6
cp_disabled	0x24	不使能协处理器执行协处理器指令	6
Watchpoint_detected	0x0B	指令或数据观测点匹配	7
window_overflow	0x05	SAVE进入无效窗口	8
window_underflow	0x06	RESTORE进入无效窗口	8
register_hadrware_error	0x20	寄存器堆 EDAC错误	9
mem_address_not_aligned	0x07	访问存储器时地址未对齐	10
fp_exception	0x08	FPU异常	11
data_access_exception	0x09	加载/存储指令访问错误	13
tag overflow	0x0A	Tagged运算溢出	14
divide_exception	0x2A	被0除	15
trap_instruction	0x80 -0xFF	软件陷阱指令 (TA)	16

陷阱描述

- ✓ 复位：复位陷阱由外部复位请求引起。它使得处理器在实际地址 0x0 开始执行。复位陷阱产生后，将使得 PSR 中的‘et’和‘s’

位分别被初始化为‘0’和‘1’。

- ✓ 写错误：数据存储到存储器时发生的异常。
- ✓ 指令访问错误 ：当访问一条指令时发生的错误。
- ✓ 非法指令 ：执行带有未实现操作数的指令，或者 UNIMP 指令以及会引起非法处理器状态的指令时发生的异常。
- ✓ 特权指令 ： 当 PSR 中的管理位为 0（即处于用户模式）而执行特权指令时产生的异常。
- ✓ fp-disabled ：当 FPU 无效或者不存在时执行 FPU 指令产生的异常。
- ✓ cp-disabled ： 当协处理器无效或者不存在时执行协处理器指令产生的异常。
- ✓ 探测监测点（watchpoint－detected）：指令返回存储器地址或加载/存储数据存储器地址，这些地址与以前加载的实现相关的“watchpoint”寄存器的内容相匹配。
- ✓ 窗口上溢： SAVE 指令使得当前窗口指针（CWP）指向无效窗口时引起的异常。
- ✓ 窗口下溢： RESTORE 或 RETT 指令使得 CWP 指向无效窗口引起的异常。
- ✓ 寄存器硬件错误（在只读寄存器访问时发生）： 检测到寄存器堆不可更正的错误。
- ✓ 存储器地址未对齐 ：加载/存储指令产生与指令地址未正确对齐的存储器地址或 JMPL，RETT 指令产生字未对齐地址时产生的异常。
- ✓ fp 异常： 浮点异常类型编码在 FSR 中的 ftt 位。

当 FPU 指令产生 IEEE_754 异常并且相应的异常使能(TEM)位为 1；或 FPU 指令未实现；或 FPU 指令不完整；或在 FPU 中存在序列或者硬件错误，这些情况都会产生 fp 异常。

浮点 IEEE_754 异常

在浮点状态寄存器（FSR）中，有 IEEE_754 浮点数异常相关字段（Floating-Point Exception Fields），实现如下：

NVM	OFM	UFM	DZM	NXM
27	26	25	24	23

图 4-1 FSR 中的陷阱允许屏蔽字段

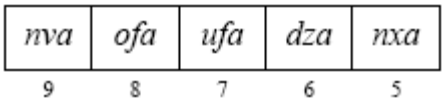


图 4-2 FSR 已发生的异常位字段

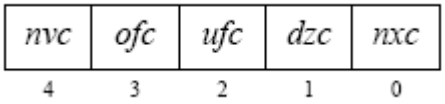


图 4-3 FSR 当前异常位字段

FSR_ invalid(nvc,nva)：对于将要执行的操作，操作数为非法操作数。比如，$0\div0$ 和 $\infty-\infty$ 运算都是无效运算。该位为 1 表示无效操作数，为 0 表示合法操作数。

FSR_overflow(ofc,ofa)：在某种数据格式中，舍入后的结果可能会在数量级上大于正常数值的上限。1 表示上溢，0 表示未上溢。

FSR_underflow(ufc,ufa)：在指定的数据格式中，舍入后的结果可能会不精确而且会在数量级上小于正常数值得下限。1 表示下溢出，0 表示未下溢出。

FSR_ division-by-zero(dzc,dza)：$X\div0$，X 为非正常操作数或者正常操作数时，对该位置位。注意 $0\div0$ 运算不会将 dzc 置位。置 1 时除数为 0，置 0 表示除数不为 0。

FSR_ inexact(nxc,nxa)：舍入后的结果与正确的无限精确结果不同时进行置位。置 1 表示不精确结果，置 0 表示精确结果。

浮点 IEEE_754_exceptions 可以通过用户能够访问的 FSR 的陷阱使能屏蔽（TEM）域控制。如果 TEM 的特定位为 1，与之相关联的 IEEE_754_exceptions 异常会产生一次 fp 陷阱。

如果 TEM 的特定位为 0，与之相关联的 IEEE_754_exceptions 不会产生一次 fp 陷阱，其产生的 IEEE_754_exceptions 异常会被记录在 FSR 的产生异常域（aexc）。

如果浮点 IEEE_754_exceptions 异常导致 fp 陷阱产生，那么目的 f 寄存器 fcc 和 aexc 区保持不变。相反，如果浮点 IEEE_754_exceptions 异常未导致 fp 陷阱，那么目的寄存器 fcc 和 aexc 区被更新。

4.3 中断

中断控制

BM3803MG 可以支持 11 种外部中断，并分为两个优先层次。这些中断都是由外部请求所引起，而与指令的操作无关。

下图给出了中断控制器的结构框图：

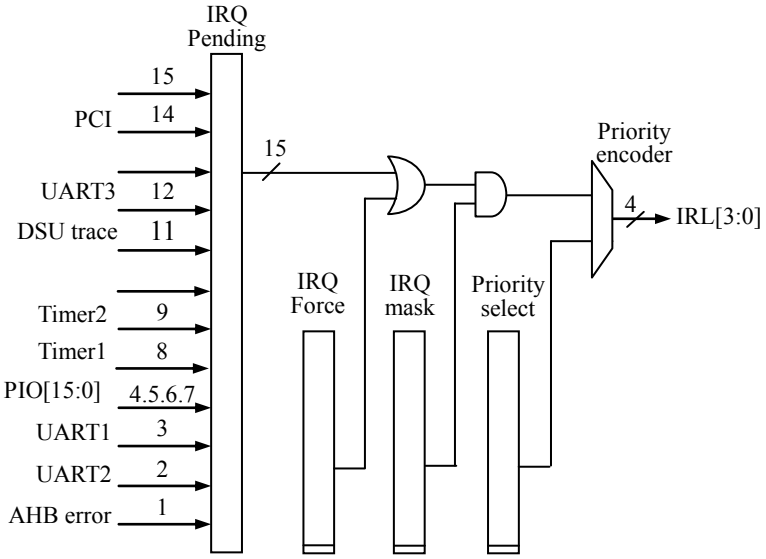


图 4-4 中断控制器示意框图

中断产生时，中断请求寄存器中的相应位被置位。中断控制器将当前优先级最高的未屏蔽的中断送给 IU。

当 IU 确认中断，则中断请求寄存器的相应位自动清零。在中断强制寄存器中可以通过置位强制中断。这时 IU 确认中断后，中断强制寄存器相应位清零，而不是中断请求寄存器相应位清零。复位以后，中断屏蔽寄存器全部清零，其他控制寄存器为不定态。

中断列表

表 4-2 中断概述

中断号	TT类型	中断源	优先级
15	0x1F	unused	17
14	0x1E	PCI 中断	18
13	0x1D	unused	19
12	0x1C	UART3	20
11	0x1B	DSU 跟踪缓冲器	21

10	0x1A	Unused	22
9	0x19	定时器 2	23
8	0x18	定时器 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 总线错误	31

外部中断信号

BM3803MG 考虑了来自芯片外部中断输入。在同一时间，可以有四个外部中断输入，四个外部中断的中断号是 4、5、6、7。在 I/O 中断寄存器（IOIT）中，每个 I/O 中断包含四个域：ENx，LEx，PLx，ISELx。

置 ENx 位为 1，中断有效，为 0 则中断无效。ISELx 位定义了在 32 位的通用接口的哪一个端口应该产生中断 x。这 32 位的端口包括 PIO[15:0] 和 D[15:0]。

同时每个 I/O 中断都有它的触发模式，同时各自配置触发极性。当 LEx 置 1，相应的 I/O 中断边沿触发，同时当 PLx 置 1，引脚信号处于上升沿时中断触发，若置 0，则在下降沿时触发中断；当 LEx 置 0，相应的 I/O 异常电平触发，同时当 PLx 置 1，引脚信号处于高电平时中断触发，若置 0，则在低电平时触发中断。下表进一步进行了说明：

表 4-3 I/O 中断配置

LEx	PLx	Trigger
0	0	低电平
0	1	高电平
1	0	下降沿
1	1	上升沿

中断优先级

BM3803MG 处理的 11 个中断，每个中断都有两个中断级别 0 和 1，对应于中断级别/屏蔽控制寄存器的中断级别字段，级别 1 的优先级高于级别 0。在同一级别中，中断 15（TT=0x1F）具有最高优先级，中断 1

（TT=0x11）具有最低优先级。

中断控制寄存器

中断控制主要包括四个中断相关的寄存器，中断级别和优先控制寄存器、中断请求寄存器、强制中断寄存器、中断清除寄存器。

中断级别/屏蔽控制寄存器（0x8000-0090）：

31-17	16	15-1	0
中断级别位	保留	中断屏蔽位	保留

中断优先级别共有 15 位，每一位的值是 1 或 0，表明中断属于哪个优先级。15-1 则表明该中断是否使能，1 表示中断使能，0 表示屏蔽。

中断请求寄存器（0x8000-0094）

31-16	15-1	0
保留	中断请求位	保留

中断请求位是 1，表明存在一个相应的中断请求。

强制中断寄存器（0x8000-0098）

31-16	15-1	0
保留	强制中断位	保留

强制中断位的某一位是 1，表明该位进入中断

中断清除寄存器（0x8000-009C）

31-16	15-1	0
保留	清除中断位	保留

当向相应位写入 1 时，清除某一中断位。

具体的寄存器描述请看附录。

五 存储器接口

5.1 概述

BM3803MG 通过外部存储器控制器可以访问 PROM、存储器映射的 I/O 设备、SRAM 和 SDRAM。系统可配两个 PROM、一个 I/O 设备、5 个 SRAM 以及两个 SDRAM。存储器的地址空间为 2G 字节，它的地址映射如下表所示：

表 5-1 存储器地址空间分配

地址范围	大小	映射
0x0000-0000 - 0x1FFF-FFFF	512M	PROM
0x2000-0000 - 0x3FFF-FFFF	512M	I/O
0x4000-0000 - 0x7FFF-FFFF	1G	SRAM/SDRAM

存储器接口配置通过三个存储器控制寄存器进行：MCFG1、MCFG2、MCFG3。其中 MCFG1 主要用于配置 PROM 和 I/O 设备，MCFG2 和 MCFG3 用于配置 SRAM 和 SDRAM。

表 5-2 MCFG1 各位的配置

比特位	名称	描述
31:29		保留
28:27	IOwdh	I/O 设备总线宽度（”00”=8，“01”=16，“1x”=32）
26	Brdy	总线准备好(BRDYN) 使能，用于 I/O 设备。高电平有效。
25	Bexc	总线错误(BEXCN)信号，用于全部外部存储器空间（PROM、I/O 和 RAM），高电平有效。
24		保留
23:20	IOWS	I/O 设备访问等待周期数。初始值为 15。(“0000”=0,“0001”=1, “0010”=2,…, “1111”=15)
19	IOen	I/O 设备访问使能，高电平有效。
16:12	PRWWS	PROM 写等待周期数。初始值为 31。(“00000”=0,“00001”=1,“00010”=2,…,“11111”=31)。
11	PRWen	PROM 写使能，高电平有效。
10		保留
9:8	PRwdh	PROM 数据宽度(“00”=8,“01”=16, “1x”=32)。此字段的初值由 GPIO 的第 1、0 两位配置。
7:5		保留

4:0	PRRWS	PROM 读等待周期数。初始值为 31。(“00000”=0,“00001”=1,“00010”=2,…,“11111”=31)。

表 5-3 MCFG2 各位的配置

比特位	名称	描述
31	Sdref	SDRAM 刷新，如果置位，SDRAM 刷新将被使能；
30	Trp	SDRAM T_{RP} 时间。如果清零，则 T_{RP} 为 2 个时钟周期，如果置位，则 T_{RP} 为 3 个时钟周期。
29:27	Trfc	SDRAM T_{RFC} 时间。T_{RFC} 为（设置数值＋3）个系统时钟周期。
26	SdrCAS	SDRAM CAS 延时。如果清零，则 CAS 延时为 2 个时钟周期，如果置位，则 CAS 延时为 3 个时钟周期。
25:23	Sdrbs	SDRAM BANK 的大小（“000”=4 Mbyte,“001”=8 Mbyte, “010”=16 Mbyte .... “111”=512 Mbyte）。
22:21	Sdrcls	SDRAM Column 的大小。 （“00”=256, “01”=512, “10”=1024, 当[22:21]=“11”并且[25:23]=“111”时，为 4096， 当[22:21]=“11”并且[25:23]/=“111”时，为 2048。）
20:19	Sdrcmd	SDRAM 命令。当写入非 0 数值时，将产生 SDRAM 命令。 “01”=PRECHARGE, “10”=AUTO-REFRESH, “11”=LOAD-COMMAND-REGISTER. 命令执行后，此数值清零。
18:15	SrWWS	SRAM 写等待周期数，初始值为 15。(“0000”=0,“0001”=1,“0010”=2,…, “1111”=15)。
14	Sdren	SDRAM 使能。如果置位，SDRAM 使能。
13	Srdis	SRAM 不使能。如果置位，SRAM 不使能。
12:9	SrBS	SRAM bank 大小。定义每个 SRAM 大小。(“0000”=8 Kbyte, “0001”=16 Kbyte , … ,“1111”=256 Mbyte)。
8		保留。
7	Brdy	总线准备好(BRDYN) 使能。如果置位，则使能 BRDYN 用于第 5 bank 的 SRAM。
6	RMW	“读改写”方式使能，高电平有效。

5:4	Srwdh	SRAM 数据总线宽度(“00”=8,“01”=16, “1x”=32)。
3:0	SrRWS	SRAM 读等待周期数，初始值为 15。(“0000”=0,“0001”=1,“0010”=2,…,“1111”=15)。

表 5-4 MCFG3 寄存器列表

比特位	名称	描述
31:27		保留
26:12	SdrRV	SDRAM 刷新计数器重载值。计算 AUTO-REFRESH 命令间隔时间公式为： $t_{REFRESH}$ = ((reload value) + 1) / SYSCLK
11:0		保留

存储器控制器和外部各种存储器的连接如图 5-1 所示。

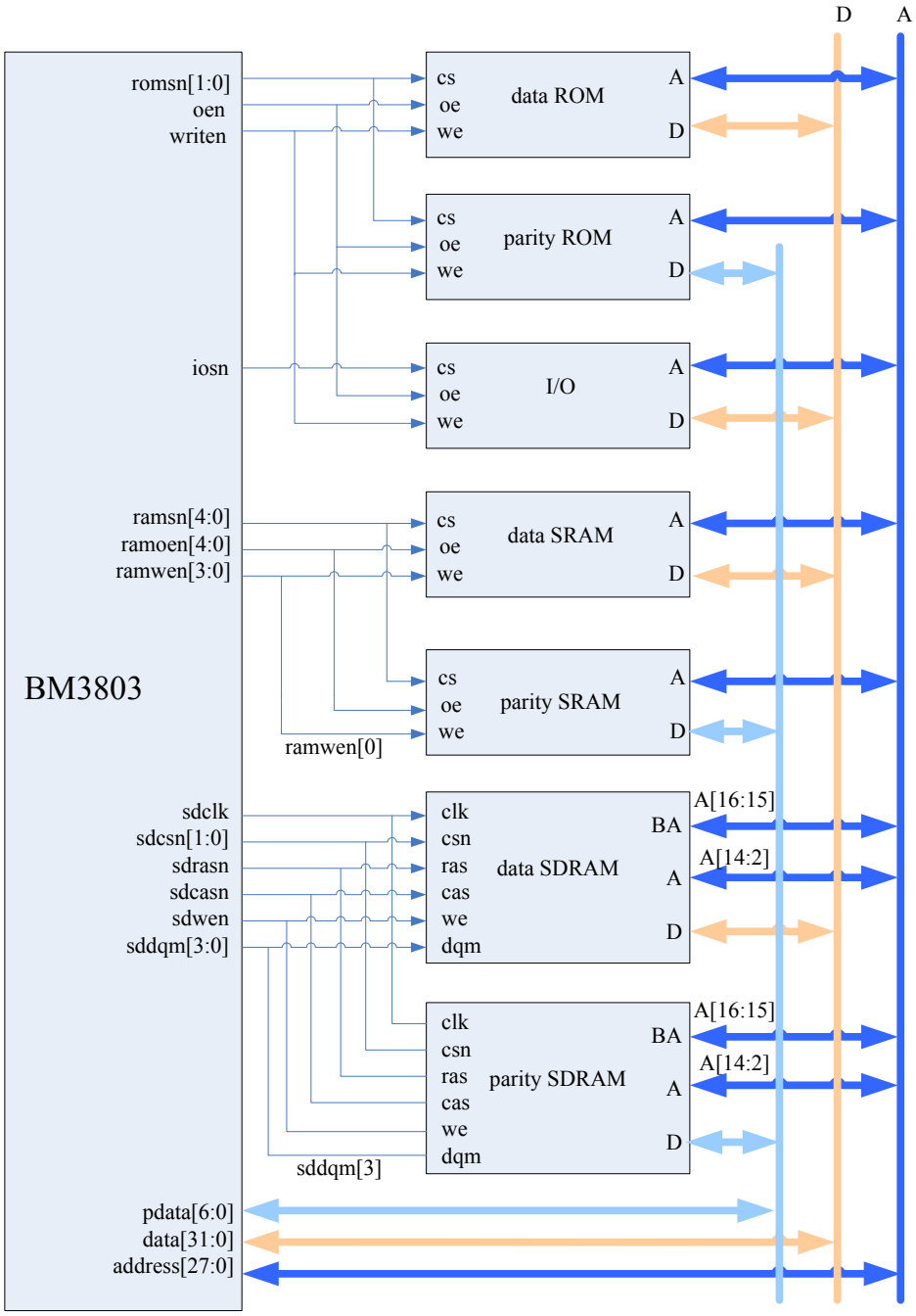


图 5-1 存储器控制器连接示意图

5.2 RAM 接口

RAM地址范围可达1G字节，支持两种RAM类型：SRAM和SDRAM。地址映射的关系如下：

表 5-5 RAM 地址空间分配

空间	地址
RAM[4]	0x6000-0000 开始
RAM[3]	0x4000-0000 开始，前四块之间平均分配。最大可用空间是128M字节。当分配大小是256M字节时，低端只有两块。
RAM[2]	
RAM[1]	
RAM[0]	

RAM[0]到 RAM[3]的起始地址从 0x4000-0000 开始，依据定义的大小顺序分配。RAM[4]大小固定有 256M 字节空间。

SRAM 和 SDRAM 的地址分配： SDRAM 块的起始地址取决于所用到的 SRAM。如果存储器配置寄存器（MCFG2)中的 SRAM 不使能位（SRDIS）和 SDRAM 使能位（SDREN）置为 1，SDRAM 的起始地址就为 0x4000-0000。如果 SRAM 不使能位（SRDIS）置 0，SDRAM 使能位（SDREN）置 1，SDRAM 的起始地址就为 0x6000-0000。如果 SDREN 置 0，则不使用 SDRAM。

关于地址分配：

- 当 SRAM 块是 256M 时，第五块 SRAM 仍然存在，而低端只有 1、2 两块 RAM。
- 当 SDRAM 有效时，第五块 SRAM 无效，高端地址被 SDRAM 使用。

SRAM 接口

SRAM 接口可以控制 5 个 SRAM 模块，存储器控制访问采用标准的管脚配置，主要信号包括：片选（RAMSN）、输出使能（RAMOEN）和写使能（RWEN）。其中前四个 RAM 块的大小能够通过设置 MCFG2 中［12:9］的相应位进行配置，按照 2 倍的关系在 8K 字节和 256M 字节范围内选择，这些块的选择由管脚的片选 RAMSN[4:0]操作，第 5 块的使用由RAMSN[4]控制，具有固定的大小，位于上面的地址空间0x60000-0000，大小为 256M 字节。

SRAM 数据总线的位宽，可以配置成 8 位，16 位和 32 位。可以通过配置 MCFG2 的[5:4]两位确定位宽大小，其值是“00”表示 8 位总线，“01”表示 16 位总线，“1x”表示 32 位。此外 MCFG2 [6]位可以控制 SRAM 是否工作在“读改写”模式。

对于第五块 SRAM，还可以通过外部信号 BRDYN 的低电平，来延迟适配外部读写的时间。这种方式实际上可以将第五块 SRAM 空间作为 I/O 空间使用。这个 BRDYN 的使用需要通过配置寄存器 MCFG2[7]位，这样只有当外部 BRDYN 信号为低电平时，才表明完成读写周期。

SRAM 的读访问时序如下图所示：

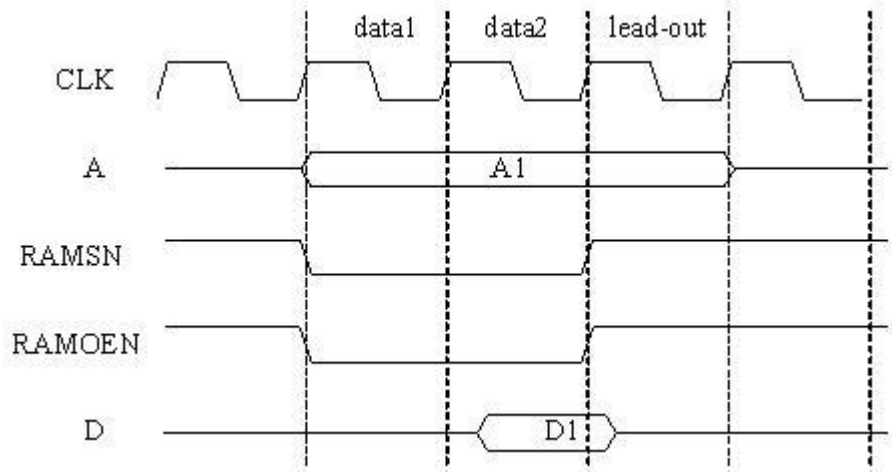


图 5-2 SRAM 读访问时序图

SRAM 的写访问时序如下图所示：

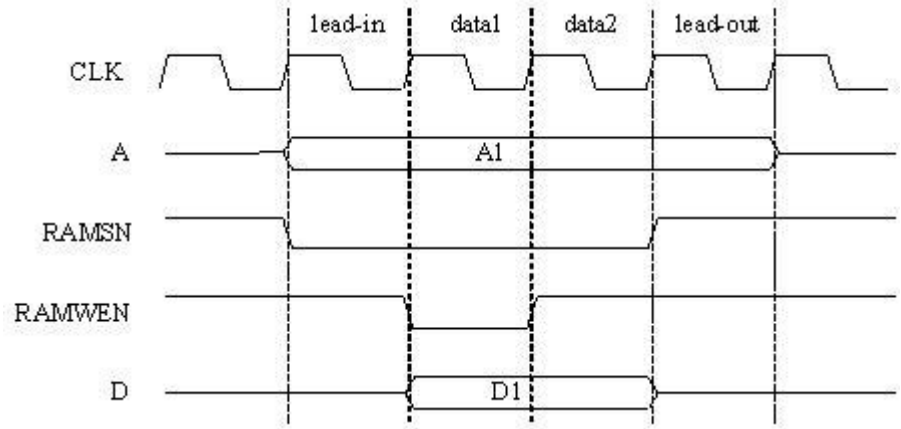


图 5-3 SRAM 写访问时序图

SDRAM 接口

同步动态随机存储器接口分为两个 SDRAM 块，SDRAM 存储器的访问控制应用了标准的管脚配置，包括：片选（SDCSN）、写使能（SDWEN）、数据屏蔽（SDDQM）和时钟引脚（SDCLK）。两块 SDRAM 的块大小通过设置 MCFG2 中的 SDRAM 块大小域的值设定，块的大小按照两倍的关系能够在 4M 到 512M 字节范围内编程。控制器支持 64M.、256M 和 512M 的器件，这些器件带有 8 到 12 列地址位，13 行地址位和

4 个块，SDRAM 只支持 32 位数据总线。

SDRAM 的地址总线应连接到 A[14:2]，块地址应连接到 A[16:15]。少于 13 个地址引脚的器件只能用 A[14:2]的最低位。

- SDRAM时序参数的配置

为了对于不同的 SDRAM 器件能够提供合适的访问周期，某些 SDRAM 参数能够通过 MCFG2 编程。

可编程参数如下：

表 5-6 SDRAM 可编程时序参数表

功能	参数	范围	单元
CAS延迟	T_{CAS}	2 - 3	clocks
预充	T_{RP}	2 - 3	clocks
自动更新命令周期	T_{RFC}	3 - 11	clocks
自动更新间隔	$T_{REFRESH}$	10 - 32768	clocks

- SDRAM命令

SDRAM 控制器能够处理三种 SDRAM 命令。被执行的命令通过 MCFG2 中的 SDRAM 命令域编程。当域值写为非零时，处理 SDRAM 命令：

“01” ：发送预充电（Precharge）命令；

“10” ：发送自动更新命令；

“11” ：发送加载模式寄存器（LMR）命令。

当处理 LMR 命令时，应用 MCFG2 中的 CAS 延时编程。命令执行完后，命令域清零。当改变 CAS 延时的值时，同时产生 LMR 命令。

控制器还提供更新命令。通过设置 MCFG2 中的更新使能位（SDREF）为 1 激活命令。对于所有的 SDRAM 块，应用自动更新命令可以使能周期更新。两个自动更新命令之间的周期在 MCFG3 的更新计数器重载区编程。

根据 SDRAM 类型，所需周期为 7.8us 或 15.6us，即对应频率 100MHZ 下的 780 或 1560 个时钟周期。

更新周期计算公式如下：

刷新周期=（重载值+1）/ 时钟频率

- SDRAM初始化

复位后，SDRAM 自动执行 SDRAM 初始化序列。同时对于两个块，它包含 PRE-CHARGE 命令，两个 AUTO-REFRESH 命令和 LOAD-MODE-REG 命令。对于读操作，控制器用页突发模式编程，对于

写操作，利用 single location 访问编程。CAS 等待时间默认为 3，并能通过软件更新。

- SDRAM读访问

在读周期包括三个主要的操作：首先，执行对所需的块和行的 ACTIVE 命令；然后，在编程的 CAS 延时后，传送 READ 命令；最后，以 PRE-CHARGE 命令结束读周期。在两次访问之间，没有块被 left open。如果内部总线发出突发模式访问请求则执行突发模式读操作。

- SDRAM写访问

在写周期包括三个主要的操作：首先，执行对所需的块和行的 ACTIVE 命令；然后，在编程 CAS 延时后，传送 WRITE 命令；最后，以 PRE-CHARGE 命令结束写周期。

内部总线的突发写操作产生没有空闲周期（in－between）的写命令的突发模式。

- 访问错误

处理器通过 BEXCN 信号指示访问错误。如果通过设置 MCFG1 中的 bexcnen 位为 1 使能访问错误，接受数据的同时接受 BEXCN 信号。如果访问外部设备时 BEXCN 信号为低，内部总线会收到错误响应。

若处于取指操作，则进入 trap 0x01；

若处于读数据操作，则进入 trap 0x09；

若处于写数据操作，则进入 trap 0x2B。

5.3 PROM 访问接口

PROM 的接口分为 2 个 PROM 块，每块的大小是 256M 字节。PROM 存储器访问控制采用了标准的管脚配置，包括：片选、输出使能、读、写块。PROM 块的块大小是不可编程的，PROM 的其中一块由 ROMSN[0]选择信号控制，用于选择地址空间较低的部分，高半部分地址 0x1000-0000 到 0x1FFF-FFFF 则由 ROMSN[1]选择信号控制。

PROM 的访问主要通过配置 MCFG1 寄存器选择各种状态，通过对[16:12]和[4:0]的设置可以分别设置 PROM 写读访问的等待周期，等待周期从 0 到 31；［9:8］是选择 PROM 的数据宽度；[11]位是确定 PROM 的写使能。PROM 的读访问接口时序图如下图所示：

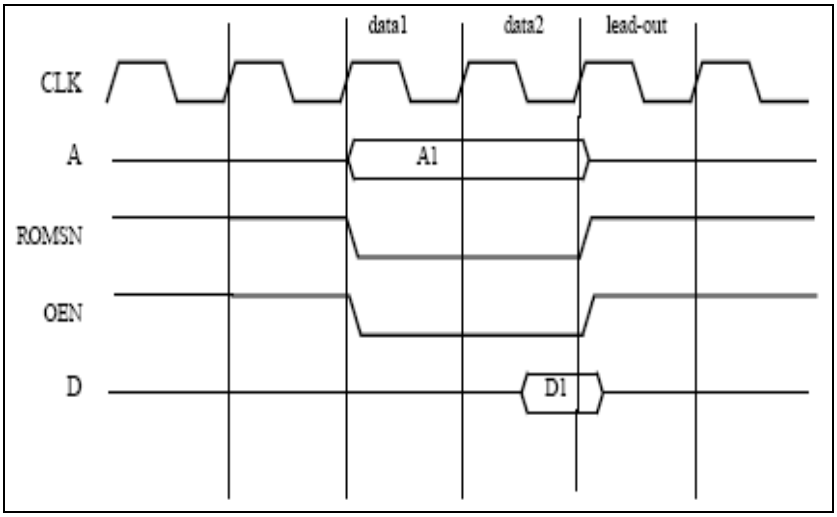


图 5-4 PROM 读访问接口时序图

5.4 I/O 空间访问接口

I/O 空间具有 512M 字节，是单独的一块，I/O 访问控制应用了标准的管脚配置，包括片选、输出使能、读、写块。I/O 块大小是不可编程的，全部 I/O 区是由 IOSN 选择信号控制。

I/O 访问和存储器的访问相似，不同的是对于 I/O，IOSN 选择信号相对于地址信号延迟一个时钟周期，同时 I/O 还可以通过外部的 BRDYN 信号来控制延迟的访问。对于 I/O 访问的控制主要通过配置寄存器 MCFG1，其中 MCFG1 的[28:27]位定义数据总线的宽度、[26]位设定 BRDYN 使能，[25]设定总线错误使能，[23:20]定义等待周期，[19]则是使能 I/O 空间。I/O 空间访问时序如下图所示：

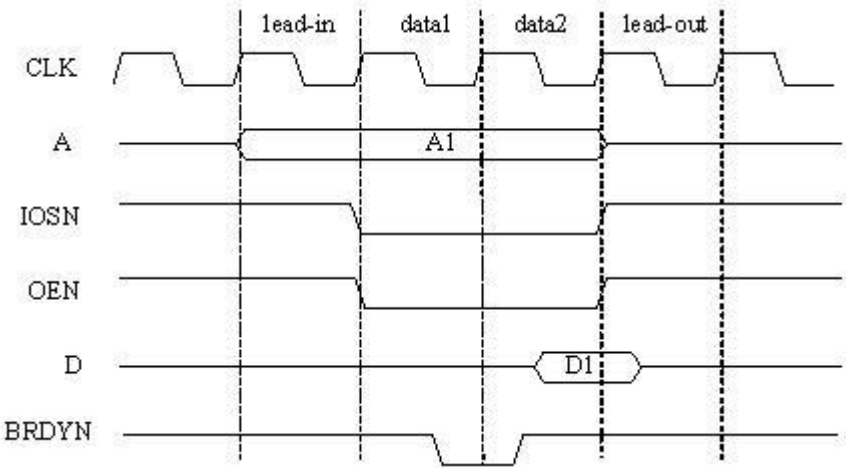


图 5-5 I/O 读时序图

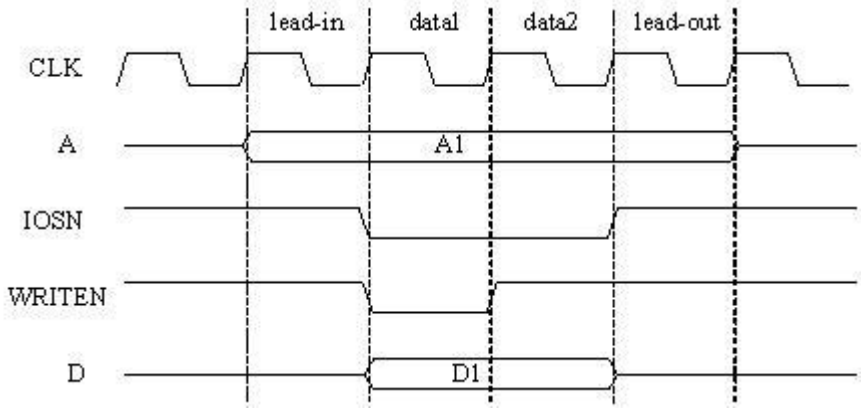


图 5-6 I/O 写时序图

5.5 错误管理

BM3803MG 处理器支持片上纠错检错功能，能够纠正 32 位字中的一个错误，能够检测 32 位中的两个错误。处理器对 8 位的 PROM、32 位的 RPOM、8 位的 SRAM、32 位的 SRAM 和 SDRAM 存储器访问具有 EDAC 功能。

通过如下的存储器容错配置寄存器实现对存储器的纠检错功能的控制。

存储器容错配置寄存器 MECFG1（0x8000-0100）

实现对 SRAM 和 SDRAM 不同 BANK 的 EDAC 使能位的控制，同时在验证 EDAC 使能模式下记录数据的校验位。

比特位	可读可写	名称	描述
31	r/w	Sr1EEn	SRAM bank1 EDAC 使能，高有效。
30	r/w	Sr2EEn	SRAM bank2 EDAC 使能，高有效。
29	r/w	Sr3EEn	SRAM bank3 EDAC 使能，高有效。
28	r/w	Sr4EEn	SRAM bank4 EDAC 使能，高有效。
27	r/w	Sr5EEn	SRAM bank5 EDAC 使能，高有效。
26	r/w	romEEn	Prom EDAC 使能，高有效。此字段的初值在上电复位时由 GPIO 的第 2 位配置。
25:23			保留。
22	r/w	EWB2	EDAC 写旁路（EWB2），EWB2、1、0=“011”时有效。
21	r/w	ERB	EDAC 读旁路(ERB)，高有效。
20	r/w	EWB1	EDAC 写旁路（EWB1），EWB2、1、0=“011”时有效。
19	r/w	SdEEn	SDRAM EDAC 使能，高有效。
18	r/w	EWB0	EDAC 写旁路(EWB0)，EWB2、1、0=“011”

			时有效。
17:14	r/w	PRBS	PROM bank size。定义每个 PROM 大小。(“0000”=8 Kbyte, “0001”=16 Kbyte , … ,“1111”=256 Mbyte)。用于 8 位 prom 的 EDAC。此字段的初值在上电复位时由外部管脚 ROMBSD [3:0]配置。
13:10			保留。
9:8	r/w	TCBarea	TCB area。将每个 bank 的 SRAM 和每个 bank 的 SDRAM 平均分成 4 份区域。在通过设置此字段而指向的区域中，被访问的数据的校验位才可以存入 TCB 字段。第 5 个 bank 的 SRAM 除外，其不受 TCBarea 字段控制。
7			保留。
6:0	r/w	TCB	Test check bits(TCB)。用于测试校验，如果 MECFG1 中 ERB 被使能，且对于 32 位数据宽度 SRAM 和 SDRAM，那么在读数据时，根据 TCBarea 字段，读得的正常的校验位存入此字段。

MECFG1[26] romEEn 此配置位，可以通过写 MECFG1 来配置，同时还可以在上电复位时通过 GPIO[2]来配置。如果 GPIO[2]为‘1’，则 romEEn 配置位在上电复位时为‘1’。

MECFG1[17:14] PRBS 此配置位，可以通过写 MECFG1 来配置，同时还可以在上电复位时通过外部管脚 ROMBSD[3:0]来配置。

EWB 写旁路说明（对于 SRAM 和 SDRAM）：

- 不论 EDAC 是否使能，只需 EWB 有效，且 32 位数据宽度 SRAM 和 SDRAM，且整个字的写操作，则写数据位和写校验位制造错误，即数据位按 MECFG2 翻转来写和校验位按 MECFG1 的 PR 字段翻转来写。
- 不论 EDAC 是否使能，只需 EWB 有效，且 8 位数据宽度 SRAM，且整个字的写操作，则写校验位制造错误，即校验位按 MECFG1 的 PR 字段翻转来写。
- 对于 8 位数据宽度 SRAM，只需 EWB 有效，且 stb 或 sth 指令，则用于直接进行数据位造错，即，造错数据就是所写数据。
- 只要 EWB 有效，则纠错不使能。
- 只要 EWB 无效，对于 32 位 SRAM 和 SDRAM，则写正常的数据位和写正常的校验位，即不制造错误。
- 当 EDAC 使能时，如果 EWB 无效，对于 8 位 SRAM，则写正常的数

据位和写正常的校验位，即不制造错误。

- 当 EDAC 使能时，如果 EWB 无效，则纠错使能，又叫 EDAC 写使能。

ERB 读旁路说明（对于 SRAM 和 SDRAM）：

- 不论 EDAC 是否使能，只需 8 位、32 位数据宽度 SRAM 和 SDRAM 读访问时，ERB 有效，则校验位存入 TCB 字段。否则，TCB 字段保持。
- 只要 ERB 有效，则检错不使能。
- 只要 ERB 无效，则 TCB 字段保持。
- 当 EDAC 使能时，如果 ERB 无效，则检错使能，又叫 EDAC 读使能。

MECFG1 配置中 EDAC、EWB、ERB 详细说明（对于 SRAM 和 SDRAM）：

- 组合 1：当设置 EDAC 使能，且 EWB 无效，且 ERB 清零时，属于正常的 EDAC 检错和纠错设置。可以写入正确的数据位和正确的校验位，不能读出实际的校验位。
- 组合 2：当设置 EDAC 使能，且 EWB 无效，且 ERB 置位时，可以写入正确的数据位和正确的校验位，以及读出实际的数据位和实际的校验位。对于 32 位 SRAM 和 SDRAM 此组合设置同第 6 组合设置相同。此组合设置在正常工作中和测试过程中，用于写入正确的数据位和正确的校验位，以及读出实际的数据位和实际的校验位。此组合设置不检错，也不纠错。
- 组合 3：当设置 EDAC 使能，且 EWB 有效，且 ERB 清零时，配合设置全 0 的 MECFG2 和 MECFG1 的 PR 字段，属于正常的 EDAC 检错和不纠错设置。
- 组合 4：当设置 EDAC 使能，且 EWB 有效，且 ERB 置位时，对于 32 位 SRAM 和 SDRAM 可以写入错误的数据位和错误的校验位，以及读出实际的数据位和实际的校验位，校验位将存入 TCB 字段。对于 8 位 SRAM 不能写入错误的数据位，可以写入错误的校验位，可以读出实际的数据位和实际的校验位，校验位将存入 TCB 字段。此组合设置在测试过程中，对于 32 位 SRAM 和 SDRAM 用于写入错误的数据位和错误的校验位，以及读出实际的数据位和实际的校验位。对于 8 位 SRAM 用于写入错误的校验位，以及读出实际的数据位和实际的校验位。此组合设置不检错，也不纠错。此组合设置同第 8 组合设置相同。
- 组合 5：当设置 EDAC 不使能，且 EWB 无效，且 ERB 清零时，对

于 32 位 SRAM 和 SDRAM 可以写入正确的数据位和正确的校验位，不能读出实际的校验位。对于 8 位 SRAM 可以写入正确的数据位，不能写校验位，不能读出实际的校验位。此组合设置可以在测试过程中，对于 32 位 SRAM 和 SDRAM 用于写入正确的数据位和正确的校验位，不能读出实际的校验位。此组合设置不检错，也不纠错。此组合设置通常不用。

- 组合 6：当设置 EDAC 不使能，且 EWB 无效，且 ERB 置位时，对于 32 位 SRAM 和 SDRAM 可以写入正确的数据位和正确的校验位，以及读出实际的数据位和实际的校验位。对于 8 位 SRAM 可以写入正确的数据位，不能写校验位，可以读出实际的数据位和实际的校验位。对于 32 位 SRAM 和 SDRAM 此组合设置同第 2 组合设置相同。此组合设置在测试过程中，对于 32 位 SRAM 和 SDRAM 用于写入正确的数据位和正确的校验位，以及读出实际的数据位和实际的校验位。此组合设置不检错，也不纠错。
- 组合 7：当设置 EDAC 不使能，且 EWB 有效，且 ERB 清零时，对于 32 位 SRAM 和 SDRAM 可以写入错误的数据位和错误的校验位，不能读出实际的校验位。对于 8 位 SRAM 不能写入错误的数据位，可以写入错误的校验位，不能读出实际的校验位。此组合设置在测试过程中，对于 32 位 SRAM 和 SDRAM 用于写入错误的数据位和错误的校验位。对于 8 位 SRAM 用于写入错误的校验位，以及可以通过 stb 和 sth 指令来直接进行数据位造错。此组合设置不检错，也不纠错。此组合设置通常不用。
- 组合 8：当设置 EDAC 不使能，且 EWB 有效，且 ERB 置位时，对于 32 位 SRAM 和 SDRAM 可以写入错误的数据位和错误的校验位，以及读出实际的数据位和实际的校验位，校验位将存入 TCB 字段。对于 8 位 SRAM 不能写入错误的数据位，可以写入错误的校验位，可以读出实际的数据位和实际的校验位，校验位将存入 TCB 字段。此组合设置在测试过程中，对于 32 位 SRAM 和 SDRAM 用于写入错误的数据位和错误的校验位，以及读出实际的数据位和实际的校验位。对于 8 位 SRAM 用于写入错误的校验位，以及读出实际的数据位和实际的校验位。此组合设置不检错，也不纠错。此组合设置同第 4 组合设置相同。

PROM 的 EDAC 功能配置说明：

- 当访问 8、32 位 PROM，读操作时，当 ERB 读旁路有效时，读得的正常的校验位将存入 MECFG1 中的 TCB 字段。
- PROM 的 EDAC 功能，与 EWB 写旁路和 ERB 读旁路无关。
- PROM 没有造错功能。

存储器容错配置寄存器 MECFG2（0x8000-0104）

主要用于控制造错模式下，数据比特位中某位的翻转。

比特位	可读可写	名称	描述
31：0	r/w	DR	Data reversal 数据位翻转，高有效。

说明：当 MECFG1 中 EWB 设置有效时，对于 8 位、32 位数据宽度 SRAM 和 SDRAM 整个字的写操作，则写入数据位按 MECFG2 来翻转。当 MECFG1 中 EWB 设置有效时，写入数据位按 MECFG2 来翻转。

存储器容错配置寄存器 MECFG3（0x8000-0108）

比特位	可读可写	名称	描述
31：7			保留。
6：0	r/w	PR	Pardata Reversal 校验位翻转，高有效。

说明：当 MECFG1 中 EWB 设置有效时，对于 8、32 位数据宽度 SRAM 和 SDRAM 整个字的写操作，则写入校验位按 MECFG3 来翻转。 当 MECFG1 中 EWB 设置有效时，写入校验位按 MECFG3 来翻转。

存储器容错配置寄存器 MECFG4（0x8000-010C）

MCTRL 版本号是 0x0011；

SRAM 和 SDRAM 出错处理

1. 如果MECFG1配置成mecfg1_1（EDAC配置成使能状态，即，MECFG1中的 Sr1EEn、Sr2EEn、Sr3EEn、Sr4EEn、Sr5EEn 和 SdEEn 置位，并且 EWB 无效和 ERB 清零），那么

   a）如果读存储器时出现单位错误（不论是数据位或校验位），那么存储器控制器将此数据进行纠正并回写存储器。这种情况不产生任何陷阱和中断。

   b）时出现非单位错误，那么，

如果此时处于读数据，那么将导致进入数据访问异常 trap0x09；
如果此时处于取指，那么将导致进入指令访问异常 trap0x01。

c）存储器时写字节或写半字时，要写入的字节或半字所在的字出现单位错误（不论是数据位或校验位），那么存储器控制器将要写入的字节或半字所在的字进行纠正，再改写这个字要写入的字节或半字，再将整个字写入存储器。这种情况不产生任何陷阱和中断。

d）存储器时写字节或写半字时，要写入的字节或半字所在的字出现非单位错误，那么将导致进入写访问异常 trap0x2b。

2. 如果 MECFG1 配置成 mecfg1_3（MECFG1 中的 Sr1EEn、Sr2EEn、Sr3EEn、Sr4EEn、Sr5EEn 和 SdEEn 置位，并且 EWB 有效和 ERB 清零，并且 MECFG1 中的 PR 字段和 MECFG2 配置为全 0），这种情况，写操作是写入正确的数据位和正确的校验位。读操作的数据校验位不存入 MECFG1 中的 TCB 字段，那么

a）如果读存储器时出现单位错误（不论是数据位或校验位），那么
如果此时处于读数据，那么将导致进入数据访问异常 trap0x09；
如果此时处于取指，那么将导致进入指令访问异常 trap0x01。

b）如果读存储器时出现非单位错误，那么
如果此时处于读数据，那么将导致进入数据访问异常 trap0x09；
如果此时处于取指，那么将导致进入指令访问异常 trap0x01。

c）对于 32 位的 sram，如果写存储器时写字节或写半字时，要写入的字节或半字所在的字出现单位错误（不论是数据位或校验位），那么

如果设置 rmw 使能，将导致进入写访问异常 trap0x2b。
如果设置 rmw 不使能，将不作任何处理。
对于 8 位的 sram 和 32 位 sdram，如果写存储器时写字节或写半字时，要写入的字节或半字所在的字出现单位错误（不论是数据位或校验位），那么

将不作任何处理，同设置 rmw 使能与否无关。

d）对于 32 位的 sram，如果写存储器时写字节或写半字时，要写入的字节或半字所在的字出现非单位错误，那么

如果设置 rmw 使能，将导致进入写访问异常 trap0x2b。
如果设置 rmw 不使能，将不作任何处理。
对于 8 位的 sram 和 32 位 sdram，如果写存储器时写字节或写半字时，要写入的字节或半字所在的字出现非单位错误，那么

将不作任何处理，同设置 rmw 使能与否无关。

3. 如果 EDAC 配置成不使能状态，即，MECFG1 中的 Sr1EEn、Sr2EEn、Sr3EEn、Sr4EEn、Sr5EEn 和 SdEEn 清零，这种情况不检错和不纠错，不产生任何陷阱和中断。
4. 如果 ERB 置位，而不论 EWB 如何配置，也不论 EDAC 使能如何配置，这种情况不检错和不纠错，不产生任何陷阱和中断。当读存储器时，读得的正常的校验位存入 MECFG1 中的 TCB 字段。

8 位、32 位 PROM 出错处理：

1. 如果 prom 的 EDAC 配置成使能状态，即，MECFG1 中的 romEEn 置位，那么当对 8 位、32 位 prom 进行访问时，
   a）如果读操作出现单位错误（不论是数据位或校验位），那么存储器控制器将此数据进行纠正并向下传递，但不回写存储器。这种情况不产生任何陷阱和中断。
   b）如果读操作时出现非单位错误，那么，
   如果此时处于读数据，那么将导致进入数据访问异常 trap0x09；
   如果此时处于取指，那么将导致进入指令访问异常 trap0x01。
2. 其它配置情况不检错和不纠错，
   a）如果读 prom 数据，不产生任何陷阱和中断。
   b）如果读 prom 指令，出现单位错误或非单位错误，那么将导致进入指令访问异常 trap0x01。(校验位不理会)

其他

对于 32 位 SRAM 和 SDRAM，字节写或半字写不能通过 MECFG2 和 MECFG3 对数据位和校验位造错。

对于 8 位 SRAM，字节写或半字写不能通过 MECFG3 对校验位造错。

作纠错处理的前提是纠错使能并检出单位错，同时纠错使能。如果作纠错处理，则不报错。

对于 8 位、32 位 SRAM，在 EDAC 写使能情况下，如果进行字节或半字写操作，则自动进入“读改写”操作而不论 rmw 配置位如何配置。

32 位数据宽度数据位 PROM 存储器同它的 8 位数据宽度校验位 PROM 存储器，除了数据信号以外，其它信号共用。

32 位数据宽度数据位 SRAM 存储器同它的 8 位数据宽度校验位 SRAM 存储器，共用相同的片选、地址、读使能信号。校验位 SRAM 存

储器的写使能信号用 BM3803 的 wrn[0]。

32 位数据宽度数据位 SDRAM 存储器同它的 8 位数据宽度校验位 SDRAM 存储器，除了数据信号和 DQM[3:0]以外，其它信号共用。校验位 SDRAM 存储器的 DQM[0]接 BM3803 输出信号 SDDQM[3]信号，校验位 SDRAM 存储器的 DQM[3:1]接高。

六 PCI 接口和仲裁器

6.1 概述

BM3803MG PCI 接口为 32 位 AHB to PCI 总线桥，实现了 AHB2.0 总线和 PCI2.3 33MHz PCI 总线之间的协议转换。BM3803MG PCI 接口可配置成主机桥（Host Bridge）形式或从属桥（Guest Bridge）形式，对应 PCI 总线上的 Host 设备和 Guest 设备。此模块实现了标准 32 位 PCI 接口的所有必选信号和可选信号中全部的 PCI 中断信号。

PCI 总线上的 BM3803MG Host 设备和 BM3803MG Guest 设备如图 6-1。图 6-1 左图，BM3803MG 处于 Host 模式，配置总线上其他 PCI 设备，并为其他设备提供总线仲裁。图 6-1 右图，BM3803MG 处于 Guset 模式，被 PCI Host 设备配置，在总线 Host 的管理下进行总线通信。

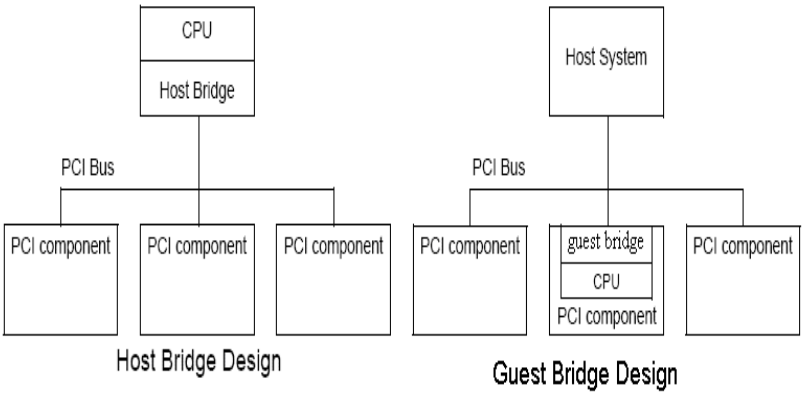


图 6-1 PCI 总线设备

6.2 AHB-PCI 模块结构图

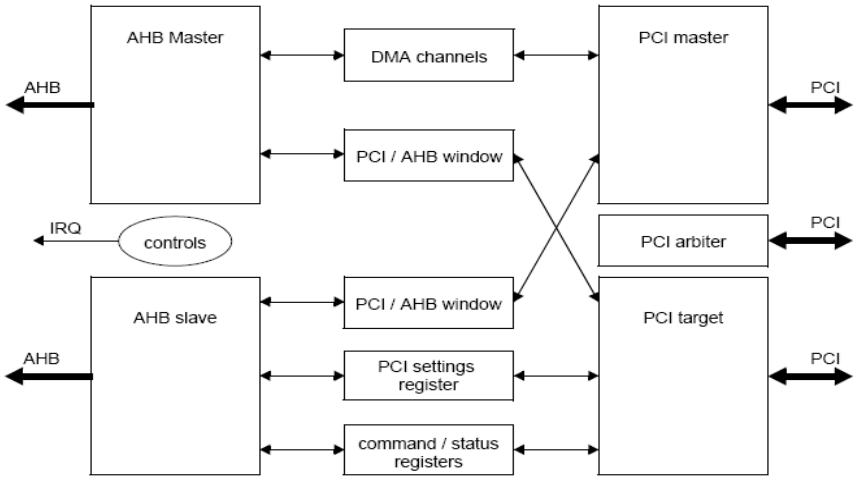


图 6-2 AHB-PCI 桥模块结构框图

PCI 接口结构如图 6-2，PCI master/target 接口将 AHB 系统连接到 PCI 总线上；BM3803MG 处理器通过 AHB slave 模块控制和配置 AHB-PCI 桥模块；BM3803MG 处理器可通过 DMA 通道配置和访问 PCI 总线设备。Host 模式下的 AHB-PCI 桥包含一个内嵌的 PCI 仲裁模块，此模块可管理 7 个外部 PCI 设备。PCI-AHB 地址空间映射（PCI-AHB window） 使 PCI 总线 Master 设备可以直接访问 CPU 存储器和 AHB 设备。

AHB 总线 Master 设备可以通过 AHB-PCI 地址空间映射（AHB-PCI window）读写存储器和 I/O 空间来访问 PCI 总线设备。

6.3 PCI 接口复位

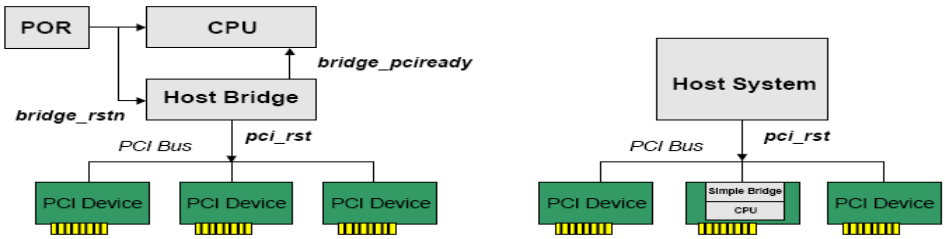


图 6-3 PCI 总线复位

如图 6-3 所示。Host 模式下的 BM3803MG PCI 接口，在 BM3803MG 芯片复位时同时复位，发出 PCI 总线复位信号，并报告 PCI 总线复位是否完成。BM3803MG 向 AHB 地址 0x8000-00D0 寄存器写入 0x05 可以使 PCI 总线复位。

Guest 模式下的 BM3803MG PCI 接口，在接收到 PCI 总线上的 RST#

信号时 PCI 侧逻辑复位，接收到 BM3803MG 复位信号时 AHB 侧逻辑复位。

6.4 PCI 接口 AHB 侧地址空间

BM3803MG PCI 接口，映射到四块独立的 AHB 总线空间，如图 6-4 所示。

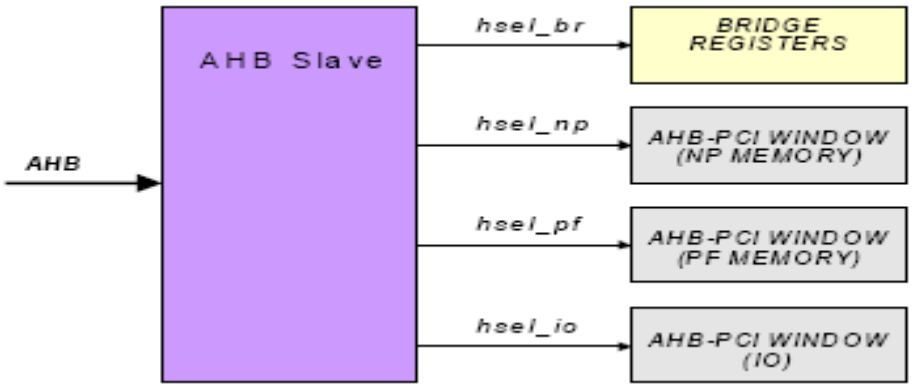


图 6-4 AHB-PCI 地址映射

空间分配如下：

寄存器空间（bridge registers）： 0xC000-0000 - 0xC7FF-FFFF

AHB-PCI 不可预取地址空间映射（NP）：0xD000-0000 - 0xDFFF-FFFF

AHB-PCI 可预取地址空间映射（PF）： 0xE000-0000 - 0xFFFF-FFFF

AHB-PCI I/O 地址空间映射（I/O）： 0xC800-0000 - 0xCFFF-FFFF

其中 AHB 侧桥寄存器空间，基地址为 0xC000-0000，偏移地址如表 6-1 所示。

表 6-1 AHB 侧 PCI 相关映射寄存器

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 PCI 仲裁器

PCI 仲裁模块在 BM3803MG 处于 PCI Host 模式时有效，此模块提供 BM3803MG PCI 接口自身的仲裁信号和最多七个其他 PCI 设备的仲裁信号。此模块的功能是：

- 使能 PCI 总线 Master 设备以开始 DMA 传输；
- 避免 PCI 总线冲突；
- 保证所有的 PCI 总线 Master 设备以同等的机会有规则地传送数据。

仲裁器能发现屏蔽并报告出错的 PCI 总线设备，以使其不影响 PCI 系统的性能。如果一个 PCI Master 设备请求 PCI 总线 16 个时钟周期以上而不开始一次总线操作，仲裁器就认为此 PCI 总线设备功能不正常。

PCI_Broken 寄存器的低 8 位反映 PCI 总线 Master 设备的状态，其中任 1 位为 1 则说明对应的 PCI Master 设备功能不正常而且被禁止访问 PCI 总线。向此位写 1，则重新使能相应的 PCI Master 设备。

6.6 PCI 侧寄存器空间

Guest 模式下，BM3803MG PCI 接口实现了 PCI 标准配置空间，除此之外，PCI 侧寄存器还包括标准 PCI 中断的桥寄存器和 PCI 版本寄存器（这些寄存器地址在 0x80-0x8c 之间）。PCI 侧寄存器的地址空间如表 6-2 所示。

表 6-2 PCI 配置寄存器地址空间

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 接口处于 Guest 模式时，实现了 PCI2.3 协议规定的标准 PCI 配置寄存器中，六个 PCI 地址空间定义寄存器 bar0---bar5 中的 bar3-bar5，如表 6-3 所示。

表 6-3 PCI 目标接口地址空间

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)

当 BM3803MG PCI 接口处于 Host 模式时，并不存在 PCI 侧寄存器，但可以通过配置读写对其他 PCI 总线设备进行配置（6.8）。

PCI Host 设备为 PCI 总线上的 BM3803MG PCI Guest 设备资源（包括 Memory 和 I/O）分配地址空间，需设置 PCI 配置空间的 BAR3、BAR4、BAR5 三个基地址寄存器[1]；其中 BAR5 为配置空间 PCI 总线基地址，这样可以让 BM3803MG PCI Guest 通过 Memory 读写取代通过配置读写来访问其他设备配置空间中的各种寄存器的方式，如读写中断寄存器（参见中断相关章节），或者重新配置配置空间中的 BAR3 或/和 BAR4 寄存器以实现地址空间重新分配。

中断相关寄存器使用见 6.9。

6.7 地址空间映射

AHB-PCI 地址空间映射

AHB-PCI 地址空间映射将 AHB 总线读写操作转化为 PCI 总线读写操作。

AHB 总线 Master 设备使用 AHB-PCI 地址空间映射直接访问 PCI 设

[1] BM3803MG PCI 设备配置空间中，PCI 头标区偏移 0x1c、0x20、0x24 的三个寄存器，即 BAR0，BAR1，BAR2 三个基地址寄存器未实现。

备。AHB-PCI 地址空间映射捕获指向 PCI 接口 AHB 空间的 AHB 操作，并将这些操作传输到 PCI 总线上。这些操作并不需要 3803 的 IU 等其他模块介入。

AHB-PCI 地址空间映射包括不可预取存储器映射空间、可预取存储器映射空间，可预取 I/O 映射空间三部分，AHB 侧地址范围在 PCI 接口 AHB 侧地址范围之内。

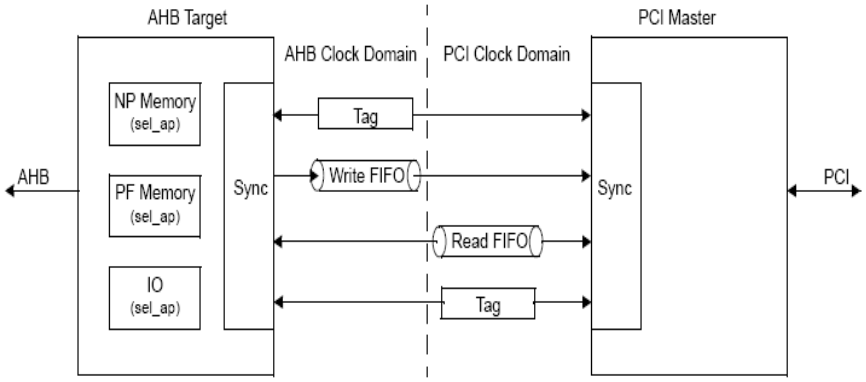


图 6-5 AHB-PCI 地址空间映射

AHB-PCI 地址空间映射使用扁平寻址方式，AHB 总线操作的地址与其转化成的 PCI 总线操作的地址相同。

具体说，BM3803MG 访问 AHB 地址空间段 0xC800-0000 - 0xFFFF-FFFF 时，可以自动产生相应的 PCI 总线操作，即 PCI 接口访问 PCI 地址空间 0xC800-0000 - 0xFFFF-FFFF 地址时：

- 0xC800-0000 - 0xCFFF-FFFF 落在 AHB-PCI I/O 地址空间映射；将产生 PCI I/O 读写操作。
- 0xE000-0000 - 0xFFFF-FFFF 落在 AHB-PCI 可预取地址空间映射；将产生 PCI 可预取存储器读写操作。
- 0xD000-0000 - 0xDFFF-FFFF 落在 AHB-PCI 不可预取地址空间映射。将产生 PCI 不可预取存储器读写操作。

如上所述，BM3803MG 可以利用地址空间映射机制，通过 AHB 总线读写操作直接访问 0xC800-0000 - 0xFFFF-FFFF 范围的 PCI 总线地址[2]（包括可预取地址空间映射和不可预取地址空间映射以及 I/O 地址空间映射），其余的 PCI 地址空间则只能通过 PCI 接口的 DMA 机制访问。

PCI-AHB 地址空间映射

PCI-AHB 地址空间映射将 PCI 总线读写操作转化为 AHB 总线读写

[2] 当然这个范围内的资源也可以通过 DMA 访问。

操作。

PCI 总线的其他 Master 设备通过 PCI -AHB 地址空间映射直接访问 BM3803MG 本地存储器和片内 AHB 总线设备。PCI-AHB 地址空间映射捕获指向自己 PCI 空间的 PCI 操作，并将这些操作传输到 AHB 总线上。同 AHB-PCI 地址空间映射类似，这些操作并不将 IU 等其他模块牵连其中。

PCI-AHB 地址空间映射包括 Bar3、Bar4、Bar5 三部分，分别为不可预取 PCI 存储器映射空间、可预取 PCI 存储器映射空间，和 PCI 接口 PCI 侧寄存器空间。地址转换的机制随 PCI 接口模式不同而不同。

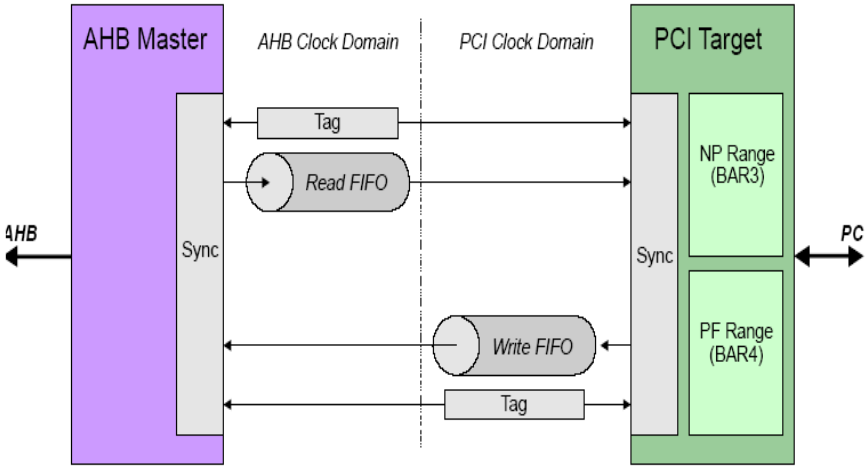


图 6-6 PCI-AHB 地址映射

PCI-AHB 地址空间映射包括 Bar3、Bar4、Bar5 三部分：

- 不可预取存储器映射空间被映射到 PCI 设备的 BAR3，处理指向此空间范围内的 PCI 操作不会进行数据预取。
- 可预取存储器映射空间被映射到 PCI 设备的 BAR4，处理指向此空间范围内的 PCI 操作会自动进行数据预取。
- PCI 接口 PCI 侧寄存器空间在 PCI 总线上被映射成 BAR5。这样使用存储器读写也可以修改 PCI 接口配置寄存器。

Bar3和Bar4的大小在256字节和1G字节之间。其对应的AHB侧地址使用AHB侧寄存器定义。分别由PCIAHB_ADDR_NP和PCIAHB_ADDR_PF定义其AHB基地址，PCIAHB_SIZ_NP和PCIAHB_SIZ_PF两个寄存器定义其大小。

在Host模式下，PCI-AHB 地址空间映射不进行地址转换，即PCI总线读写操作和其转换成的AHB总线读写操作目标地址一致。反过来说Host模式下PCI接口的PCI-AHB 地址空间映射的PCI侧地址范围由AHB侧寄

存器定义。

在 Guest 模式下，PCI 接口的 PCI-AHB 地址空间映射的 PCI 侧地址范围由 BAR3 base address、BAR4 base address、BAR5 base address 决定（PCI spec2.3）。PCI 空间和 AHB 空间之间存在如下转换：

不可预取空间：

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

可预取空间：

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

6.8 DMA

PCI 接口的很大一部分功能是通过 DMA 机制进行的，DMA 通道如图 6-6 所示，在 DMA 机制中 DMA 控制器在控制 PCI 总线的同时也控制 AHB 总线，实现独立于 CPU 的数据传输：

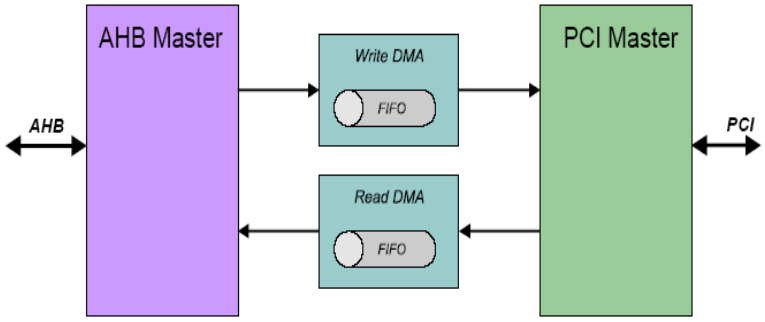


图 6-7 DMA 通道

DMA 机制可以在 AHB 总线和 PCI 总线之间实现大量的 Memory 或 I/O 数据传输而仅需要少量的 CPU 干预，也可以执行配置周期和 I/O 操作。

CPU 分配和初始化一段存储器缓冲区，然后初始化设置 DMA 寄存器，并指定 PCI 交易的类型和目标地址。交易被初始化为 AHB Master 后，CPU 再无任何动作，PCI Master 会自动执行所需要的 PCI 总线数据交易（除非用户终止 DMA 传输）。DMA 传输状态将报告给 CPU_ISTATUS 寄存器，可以通过此寄存器引起 BM3803MG 处理器中断。

在 DMA 操作中，DMA 读是指从 PCI 地址空间中向 BM3803MG 的 AHB 地址空间中读取数据；DMA 写是指把 BM3803MGAHB 地址空间中的数据写入 PCI 地址空间。

发起 DMA 操作需要三个寄存器：PCI 地址寄存器、AHB 地址寄存器

和 DMA 控制寄存器，而 DMA 读寄存器和 DMA 写寄存器具有不同地址。DMA 写控制寄存器（地址为 0xC000-0008）、DMA 读控制寄存器（地址为 0xC000-0028），即 DMA 传输的 AHB 地址（WDMA_AHB_ADDR、RDMA_AHB_ADDR）会随着 DMA 的进行自动增长，直至 DMA 结束。

DMA 控制寄存器 DMA_CONTROL（包括读和写）定义了 PCI 总线上数据传输的 PCI 命令和控制操作，结构见下图：

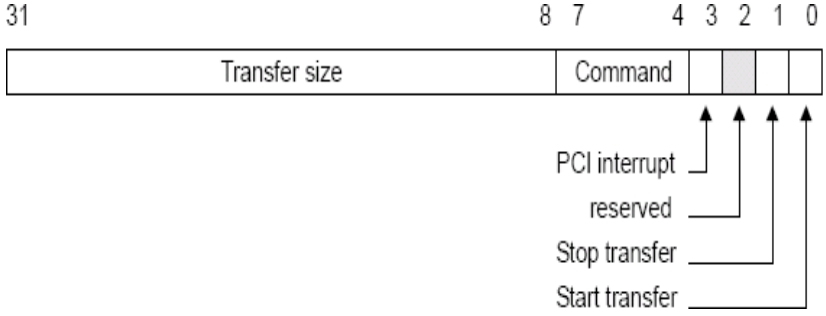


图 6-8 DMA 控制寄存器

- Bit[0] 置 1 可以启动 DMA，完成时自动清零[3]，DMA 完成/退出时会在中断状态寄存器（CPU_ISTATUS）相应的位置 1（见中断章节相关描述）；
- Bit[1] 置 1 可以终止正在进行的 DMA；
- Bit[3] 置 1 可以使 DMA 完成时在 PCI 总线上产生中断信号。

Transfer size 为 DMA 传输字节长度，最大为 16MB，取值为 0x1-0xFFFFFC；（16MB 应该是 24 位）

Command 部分表示发起的 DMA 操作类型，见下表：

表 6-4 DMA 操作类型

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

[3] 在 DMA 开始之后，等待并判断 DMA 完成时可以查询此位

例如，代码

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

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

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

发起把 AHB 地址 0x4000-0000 处 0x100 字节写往 PCI 地址 0xC000-0000 处的 DMA Memory Write 操作。

代码

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

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

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

则读出了槽位地址在 0x4000-0000 处的 PCI 设备配置空间中 0x100 字节的内容。

BM3803MG PCI Host 设备通过配置 DMA 进行 PCI 配置时，应尽量使用单字 DMA 读写。配置 DMA 读写时 PCI 地址的确定由 PCI 总线底板的槽位及选择关系决定。

通过DMA操作产生PCI总线类型0和类型1配置周期时的PCI总线地址格式如下图所示：

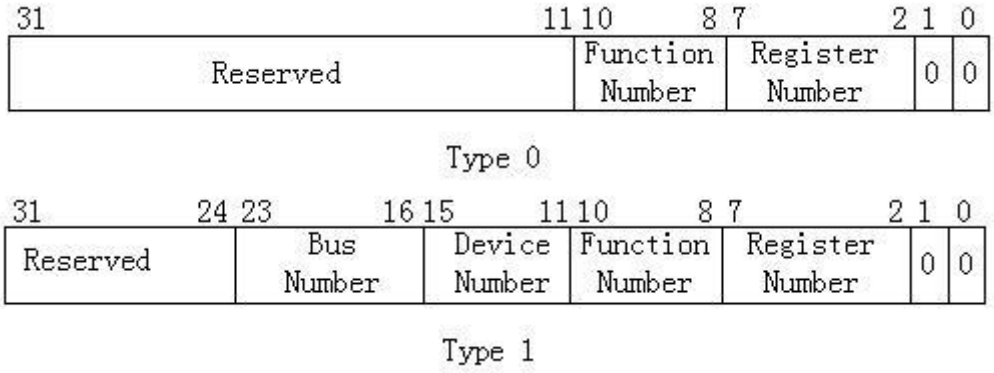


图6-9　Type 0和Type 1地址空间配置

BM3803MG AHB 总线为大端总线，而 PCI 总线为小端总线，所以他们之间的 DMA 传输需按照一定的要求来保证数据顺序一致。

当 BM3803MG DMA 目标或源地址为 32 位的 RAM 空间且进行 DMA 传输的 PCI 设备为两个 BM3803MG PCI 设备时，需遵循以下原则以避免字节序错位：

- PCI 地址和 AHB 地址后两位相等时，进行任意类型字节长度的 DMA；
- PCI 地址和 AHB 地址 2 字节（半字）对齐时，DMA 长度为 2 或 2

的倍数；

- PCI 地址和 AHB 地址 4 字节（字）对齐时，DMA 长度为 4 或 4 的倍数。

当 3803 DMA 目标或源地址为 8 位 RAM 空间时，则需要 PCI 地址和 AHB 地址 4 字节对齐，DMA 长度为 4 字节或 4 的倍数。

推荐使用 PCI 地址和 AHB 地址 4 字节对齐，DMA 长度为 4 字节或 4 的倍数的 DMA 方式，此时无论何种情况都不会被大小端问题影响。

6.9 PCI 模块中断

PCI 模块中断既可以通过 AHB 总线向 BM3803MG 的 IU 模块发出中断请求，也可以通过 PCI 总线发出中断（Guest 模式）或响应来自 PCI 总线的中断请求（Host 模式）。前者称为处理器中断，后者称为 PCI 中断。

处理器中断

PCI 接口的处理器中断为 BM3803MG 处理器的二级中断，PCI 部分配置为 Host 或 Guest 时，其实现稍有不同。AHB 侧有三个中断相关寄存器：

- 0xC000-0040 ： CPU_IMASK　　CPU 中断掩码寄存器
- 0xC000-0044 ： CPU_ISTATUS　　CPU 中断状态寄存器
- 0xC000-0048 ： CPU_ICMD　　CPU 中断命令寄存器

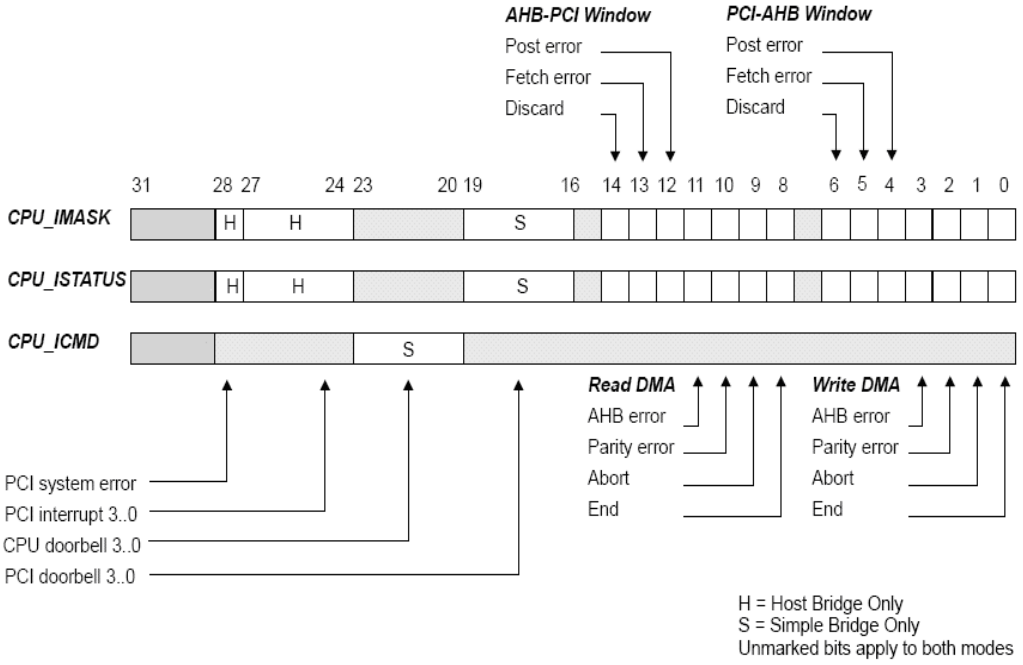


图 6-10 CPU 中断寄存器

- CPU_ISTATUS：CPU中断状态寄存器，此寄存器可读/可写/可清零。相应的中断源被激活，状态寄存器相应位自动置位为1，各个中断源是相互独立的且多个中断源可以同时触发。CPU可以监控并清除状态位，写1清除相应状态位，写0无效。
- CPU_IMASK：CPU中断掩码寄存器，此寄存器可读/可写。寄存器置位使能相应的中断源，清零屏蔽相应的中断源。上述操作不影响CPU_ISTATUS寄存器各位的值：也就是说被屏蔽的中断仍会报告给相应的状态位。如果一个或多个中断源被触发且未被屏蔽，则PCI桥就会向CPU申请中断。
- CPU_ICMD：CPU 中断命令寄存器，只写寄存器，用于人为使能一些中断源。读操作返回 0。写 1 置位，写 0 无效。

PCI 中断

PCI 侧配置空间中也有三个中断寄存器，PCI_IMASK、PCI_ISTATUST、PCI_ICMD，分别为 PCI 中断掩码寄存器，PCI 中断状态寄存器，PCI 中断命令寄存器。这三个寄存器的 CPU Doorbell 位和 PCI Doorbell 位和 CPU 中断寄存器的对应位配合使用产生中断请求（只在 Guest 模式下有效）。这三个寄存器只能通过 PCI 总线读写，如图所示。

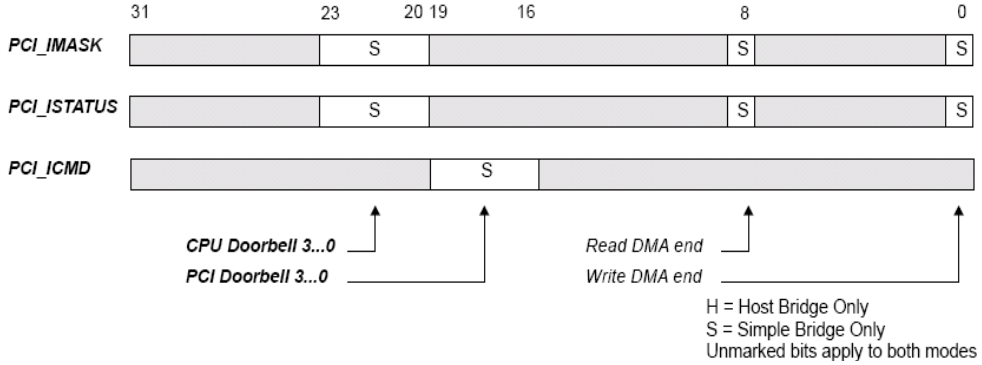


图 6-11 PCI 中断寄存器

- PCI Doorbell：PCI设备可以通过访问此位向3803中断控制器发出中断请求。若PCI Doorbell中断在CPU_IMASK寄存器中使能，此时置位PCI Doorbell位，则BM3803MG的CPU中断中的PCI Doorbell中断有效。该位只写，写0无效，写1置位（且只在Guest bridge模式下有效）。
- CPU Doorbell：CPU Doorbell中断由BM3803MG写CPU_ICMD对应位触发。若此时PCI_IMASK寄存器中断源被使能，则处理器可以通过PCI总线INTA#发送用户自定义的中断，并访问PCI_ISTATUS寄存器查询中断类型（只在Guest bridge模式下有效）。

- Read DMA end 和 Write DMA end的使用和CPU Doorbell类似，也是通过PCI总线的INTA#信号向PCI总线Host设备发出中断信号，中断由DMA完成事件触发。
- PCI_VERSION 寄存器与 CPU_VERSION 寄存器一致。(详细内容参见 CPU_VERSION 寄存器)。

七 GPIO、UART 和定时器

7.1 GPIO

基本功能

通用接口 GPIO，为 32 位宽的双向 I/O 端口，分为高 16 位和低 16 位，其中高 16 位与数据总线复用，低 16 位由并行 I/O 端口访问。

低 16 位

GPIO 低 16 位对应 PIO[15:0]，端口可与其他功能复用，当用作通用 I/O 端口时，每个端口都具有读写功能，相互独立且可被改写。 PIO[15:0]的每个引脚连接两个控制寄存器：IODIR 和 IODAT，寄存器 IODIR 中的 IODIR*x* 位选择端口 *x* 的方向，若置 1，则相应的引脚配置成输出，置 0 则相应的引脚配置成输入。当引脚配置成输入时，读 IODATx 位返回当前引脚值。当引脚配置成输出时，如果 IODATx 位置 1，端口 x 拉高；反之 IODATx 位置 0，端口 x 拉低。

当通过转换 IODIRx 值使端口 x 从输入转换到输出模式时，相应引脚的 IODATx 值立即被驱动。当触发 IODIRx 位使得端口 x 从输出转换到输入模式时，引脚值立即被写入 IODATx。

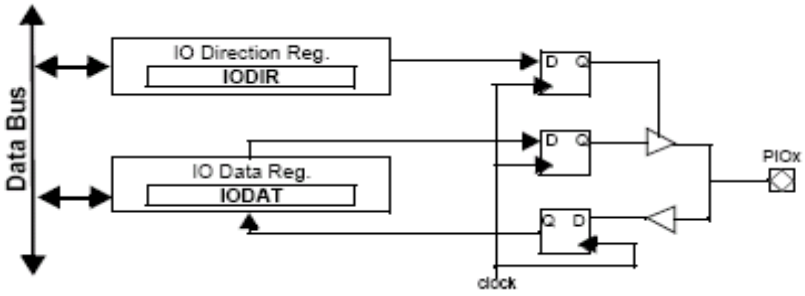


图 7-1 PIO[15:0]结构框图

高 16 位

GPIO 高 16 位对应 D[15:0]。他们仅当所有的存储器（ROM，RAM，I/O)为 8 位或 16 位时被应用。如果 SDRAM 控制器使能，则高 16 位不可用。

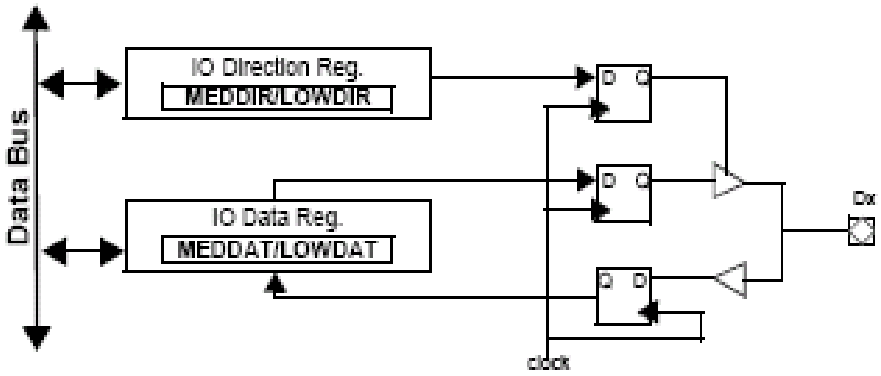


图 7-2 D[15:0]结构框图

高 16 位仅能配置成基于字节的输出或输入。当 D[7:0]被参考作低字节时，D[15:8]被参考作中间字节。

D[15:0]的每个字节连接两个寄存器域。Direction 域在 IODIR 地址位被访问，数据域在 IODAT 地址位被访问。

IODIR 寄存器中的 MEDDIR 和 LOWDIR 位分别为中间字节 D[15:8]和低字节 D[7:0]选择 Direction。

如果 MEDDIR 或 LOWDIR 被写入逻辑 1，则 D[15:0]中相应的字节配置成输出，反之置 0，被配置成输入。

当配置成输入时，读 IODAT 寄存器中的 MEDDAT 值返回 D[15:8]的当前值，读 IODAT 寄存器的 LOWDAT 值返回 D[7:0]的当前值；当配置成输出时，MEDDAT 的逻辑值被译成 D[15:8]总线的物理值，LOWDAT 逻辑值被译成 D[7:0]的物理值。

当通过转换 MEDDIR 或 LOWDIR 值，把中间字节（或低字节）从输入转换到输出时，MEDDAT 或 LOWDAT 的值立即在相应的引脚被驱动。当通过触发 MEDDIR（或 LOWDIR)从输出转换到输入时，引脚值立即被写入 MEDDAT(或 LOWDAT)。

复用功能

复用功能如下所示：

I/O 端口	功能	类型	描述
PIO [15]	TXD1	输出	UART1 输出数据
PIO [14]	RXD1	输入	UART1 输入数据
PIO [13]	RTS1	输出	请求发送
PIO [12]	CTS1	输入	允许发送

PIO [11]	TXD2	输出	UART2 输出数据
PIO [10]	RXD2	输入	UART2 输入数据
PIO [9]	RTS2	输出	请求发送
PIO [8]	CTS2	输入	允许发送
PIO [3]	UART CLOCK	输入	可选的外部时钟
PIO [1:0]	PROM 宽度	输入	初始化 PROM 的宽度（启动时）

除了用作通用 I/O，绝大多数 GPIO 引脚具有复用功能，如串行通信，中断输入以及相应配置都可以通过这些管脚实现。除了上述复用功能外，每个 GPIO 接口引脚都可以配置成输入从而捕获外部器件中断，通过编程 I/O 中断寄存器，GPIO 接口能够配置成四个中断。

通过读写地址空间 0x8000-00A0 对 I/O 端口数据进行操作，通过读写地址空间 0x8000-00A4 对 IODIR（仅有低 18 位有效）进行操作，通过读写地址空间 0x8000-00A8 访问 I/O 端口中断配置寄存器。中断配置寄存器如下所示，每 8 比特一组，包括使能位 EN，边沿/电平触发方式（LE）、极性（PL）、以及 32 位 I/O 端口中哪个管脚信号作为中断输入的选择。

7.2 UART

UART 是非常灵活的串行通信模块，BM3803MG 实现了三个通用异步收发器。其中两个的管脚是与 GPIO 复用。每个 UART 由发送保持寄存器、接收保持寄存器、发送移位寄存器和接收移位寄存器组成；控制主要通过控制寄存器、状态寄存器、分频器寄存器、数据寄存器组成。串口数据的发送和接收通过对上述寄存器的操作实现。

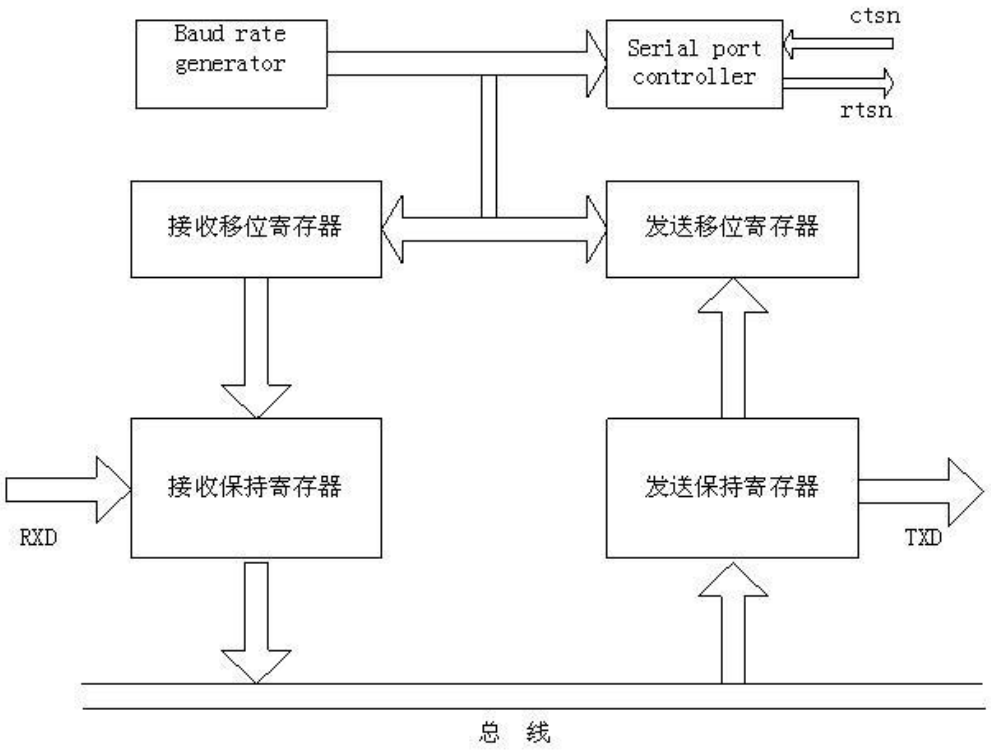
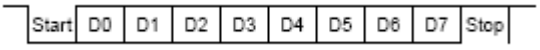


图 7-3 结构框图

发送/接收操作

在 UART 控制寄存器中置 TE 位，可以进行发送操作。当进行发送时，数据将从发送保持寄存器传输到发送移位寄存器，并在串行输出口（TXD）转化为串行数据流，同时自动在每 8 位数据前增加一位 start 位，在数据后增加可选的奇偶校验位和 stop 位。首先发送的是数据最低位：

数据桢，无奇偶校验：



数据桢，包含奇偶校验：

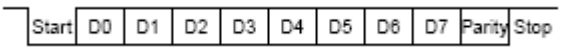


图 7-4 UART 数据桢格式

在 UART 控制寄存器置 RE 位，可以进行接收操作。接收端等待在接收数据输入引脚上出现一个从高电平到低电平变化的起始位。若接收使能，则将在半个时钟之后采样串行输入。若串行输入采样为高，则起始位无效，将继续等待有效的起始位；若串行输入为低，起始位有效，接收端在每个时钟间隔内采样串行输入，直到发现数据位，以及奇偶校验位和 stop 位。然后将串行输入数据转移到 8 位接收移位寄存器，此时

需寄存器中的数据具有相同的值，才能确保获得新的数据。

接收时，首先接收最低位，然后数据被转发到接收保持寄存器（RHR），并在 UART 状态寄存器中置数据准备好（DR）位，同时设置奇偶校验位错、帧错以及溢出错误位等。若有新的数据接收，而此时接收保持和移位寄存器中都包含未读数据，则接收移位寄存器中的数据将丢失，同时置 UART 状态寄存器中的溢出错误位。

流控/回送方式

若流控使能（FL），则 CTSN 输入必须为低电平时才能进行发送。若发送过程中，CTSN 不为低，将继续发送移位寄存器中的数据，且串行输出口保持无效状态，直到 CTSN 重新为低电平；接收时，当监测到有效的起始位，RTSN 将无效，且接收保持寄存器包含未读的字符。当保持寄存器被读时，RTSN 将自动重新监测。如果 CTSN 连接到了接收端的 RTSN，则可以有效地避免数据溢出。

若 LB 位置位， UART 将进入回送模式。在该模式下，发送端输出将在内部与接收端输入相连，RTSN 也将连接 CTSN，从而可进行回送测试，验证接收/发送端以及相关软件的工作情况。在该模式下，输出仍然保持无效状态，避免向外发送数据。

波特率产生

每个 UART 包含一 12 位的减法计数，可产生期望的波特率。分频器由系统时钟计时，在每次下溢时产生 UART TICK 信号，同时根据 UART 分频寄存器中的值重载 scaler。UART TICK 信号的频率是所需波特率的 8 倍。若 EC 位置位，则时钟由 PIO[3]触发。在这种情况下，PIO[3]的频率必须小于系统时钟频率的一半。

波特率产生公式：

分频器重载值=（（系统时钟*10）/（波特率*8）- 5）/10

中断产生

在以下的情况下，UART 将产生一个中断：当发送使能、发送中断使能，发送保持寄存器从满到空；当接收使能，接收中断使能，接收保持寄存器从空到满；当接收使能，接收中断使能，产生了奇偶校验位错、帧错或溢出错误等。

7.3 定时器和看门狗

定时器

BM3803MG 具有两个定时器，一个看门狗，它们共用同一预分频器。预分频器为一个 10 位的减一计数器，由系统时钟定时，每个时钟周期减一，当产生下溢时，由预分频重载寄存器自动重新装载，同时为两个定时器和看门狗产生一计数 TICK 信号，所以有效的分频值等于预分频重载寄存器值加 1。定时器和看门狗共用同一减一计数器。

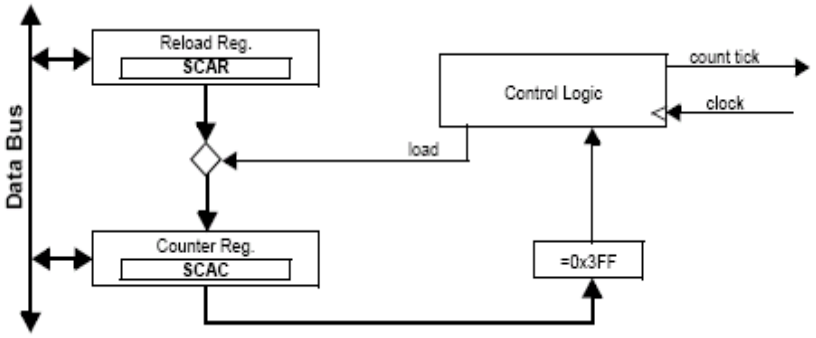


图 7-5 预分频器结构图

两个定时器是通用的 24 位定时器，当预分频器产生定时脉冲时，定时器的值就减一，定时器操作是由专门的定时器控制寄存器控制，通过设置定时器控制寄存器使能位来控制是否使能。每当定时器下溢，就产生一中断，这些中断就能够被中断屏蔽和优先级寄存器屏蔽。发生下溢之后，通过设置定时器控制寄存器的加载位可以自动重载寄存器的值，同时定时器继续运行。

每个定时器具有 3 个寄存器：计数器，重载计数值寄存器，控制寄存器。

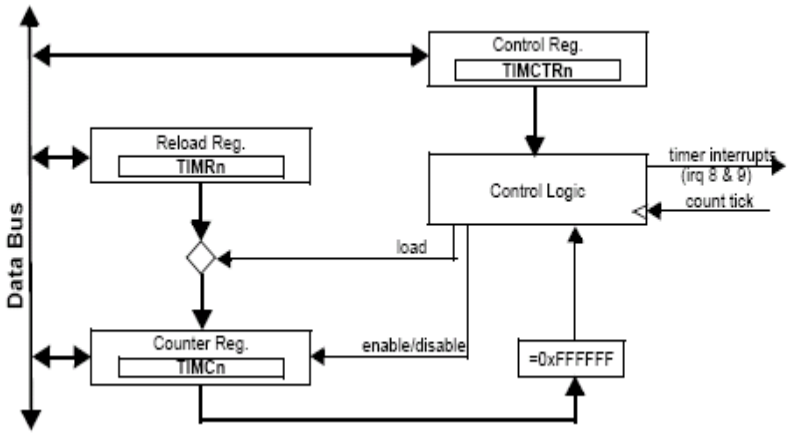


图 7-6 定时器结构框图

看门狗

系统复位后，看门狗处于不使能状态，通过设置控制寄存器中的 Watchdog 使能位使能看门狗，在每次下溢时都能产生一个外部信号 WDOG，这个信号可以用来产生系统复位。在计数器到 0 之前，如果看门狗计数器更新值被写入到 WDOG 寄存器，则计数器重新置计数值。

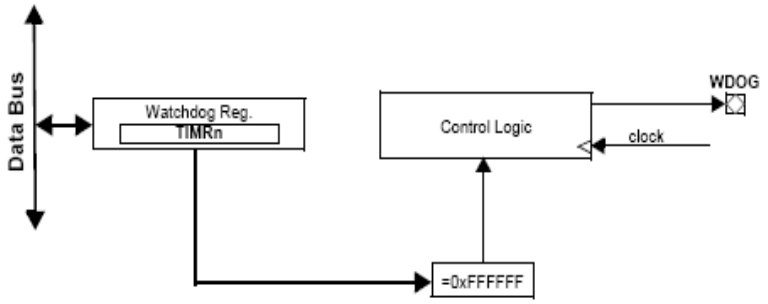


图 7-7 看门狗结构框图

八 调试接口

8.1 概述

BM3803MG 处理器具有调试支持单元，辅助软件在目标硬件上进行调试。此单元包括两个模块：调试支持单元 DSU 和调试通讯链接模块 DCL。通过 DSU 可以将处理器置于调试模式，允许处理器中所有寄存器及 Cache 存储器的读写访问。DSU 同时具有一大小为 512*16 字节的缓冲区，保存被执行的指令或总线上的传输数据。通过串口控制片上调试单元。

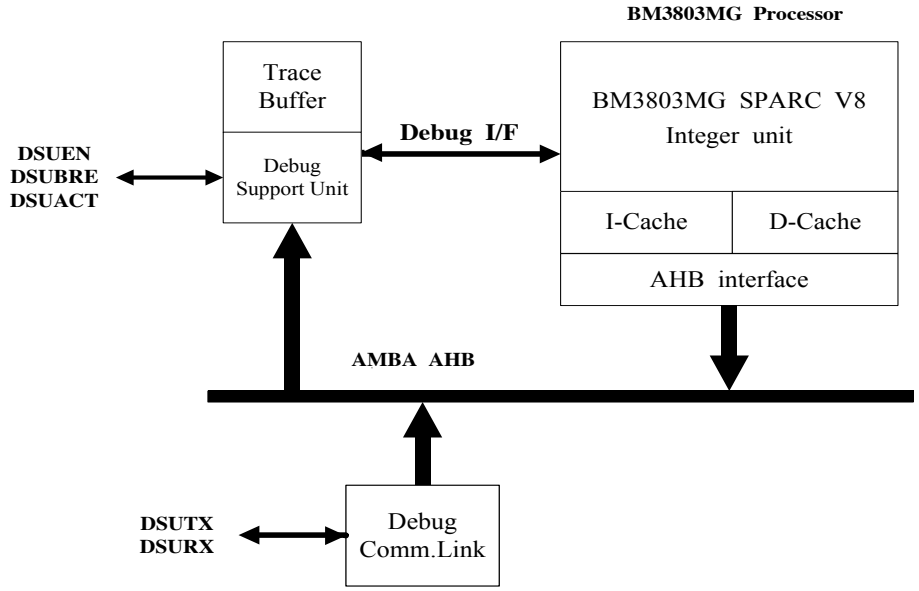


图 8- 1 调试支持单元和通讯连接

可能通过任何在内部总线上的主设备来调试处理器。PCI接口被固定作一个内部总线上的主设备，从任何PCI主设备可以得到所有的调试特征。

8.2 调试支持单元

调试支持单元用来控制跟踪缓冲区和处理器调试模式，DSU 主设备在内部总线上占有一个地址空间。通过这个地址空间，任何其他主设备都可以访问处理器的寄存器和跟踪缓冲区的内容。

DSU 控制寄存器在任何时候都可以被访问。而处理器的寄存器和

cache 只能在处理器进入调试模式时访问。跟踪缓冲区只能在跟踪被禁用或者完成时被访问。在调试模式下，DSU 控制处理器，流水线暂停。

通过下列事件可以进入调试模式：

- 执行一条断点指令（ta.1）；
- 观察点命中；
- 外部中断信号的上升沿（DSUBRE）；
- 设置 DSU 控制寄存器中的 BN 位；
- 导致处理器进入错误模式的陷阱；
- DSU 控制寄存器中定义的陷阱发生；
- 在一条单步操作后；
- DSU 断点命中。

DSU 的外部引脚（DSUEN）使能后才能进入调试模式。DSUEN 引脚拉高，使能调试模式。当进入调试模式后，发生随后动作：

- 程序计数器（PC）和 nPC 被存入临时寄存器（可通过调试单元访问）；
- 输出信号 （DSUACT）指示出（进入）调试状态；
- 时钟单元停止，冻结定时器和看门狗。

引起处理器进入调试模式的指令不被执行，且处理器状态保持不变。通过清除 DSU 控制寄存器中的 BN 位或 DSUEN 位可以使处理器重新恢复执行。定时器单元将被重新使能且从保存的 PC 和 nPC 处继续执行。调试模式也可以在处理器错误模式后进入，比如一个应用程序终止后挂起了处理器。错误模式可以被复位，处理器从任意地址重新开始执行。

时间标志

DSU 有一个时间标志计数器。跟踪中，时间标志计数器值被记录，作为执行操作的时间参考。处理器运行时，计数器每时钟周期减一。处理器进入调试模式时计数器停止，重新开始执行时计数重新开始。

跟踪缓冲区

跟踪缓冲区是一个环形缓冲区，存储执行的指令或者内部总线上传输的数据，其大小为 512 行，每行 16 字节。跟踪缓冲区由 DSU 控制寄存器和跟踪缓冲区控制寄存器控制。当处理器进入调试模式时，跟踪暂停。

跟踪缓冲区有三种工作模式：

- 指令跟踪模式（只记录执行的指令）
- 总线跟踪模式（只记录内部总线上传输的数据）
- 混合跟踪模式（同时记录执行指令和总线数据）

跟踪缓冲区控制寄存器包含了两个计数器（总线计数器和指令计数器），表示下一次跟踪记录写入跟踪缓冲区的位置。由于缓冲区是环形的，因此计数器值实际上指向缓冲区最旧记录位置，每次对存储跟踪缓冲区完成后，计数器指自动增加。

指令跟踪

缓冲区控制寄存器中的指令跟踪使能位置 1，使能指令跟踪模式。

在指令跟踪期间，除多周期指令，一条指令占用缓冲区的一行。多周期指令占用跟踪缓冲区两行或三行：

- 对于存储指令，缓冲区中第一行[95:64]用于记录存储地址；第二行相同字段存储写入的数据。对 STD 指令，需要使用三行记录，第一行[95:64]字段，记录存储地址，第二行相同字段存储写入数据的高 32bit，第三行相同字段存储写入数据的低 32bit。当一行的 bit 126 置 1 时，表示这是一条指令的第二行或第三行记录。
- 双字加载指令(LDD)使用两行跟踪缓冲区，第一行[95:64]字段存储读出数据高 32bit，第二行相同字段存储读出数据的低 32bit。
- 乘法和除法指令使用两行跟踪缓冲区。对乘法指令，第一行[95:64]字段记录计算结果的高 32bit，第二行[95:64]字段记录计算结果的低 32bit；对除法指令，第一行[95:64]字段记录为符号扩展，第二行[95:64]字段记录 32bit 运算结果。Bit 126 置位，表示同一指令记录的第二行。
- 双精度浮点指令使用两行跟踪缓冲区。第一行[95:64]字段存储计算结果的高 32bit，第二行[95:64]字段存储计算结果的低 32bit。Bit 126 置位，表示同一指令记录的第二行。

记录格式如下：

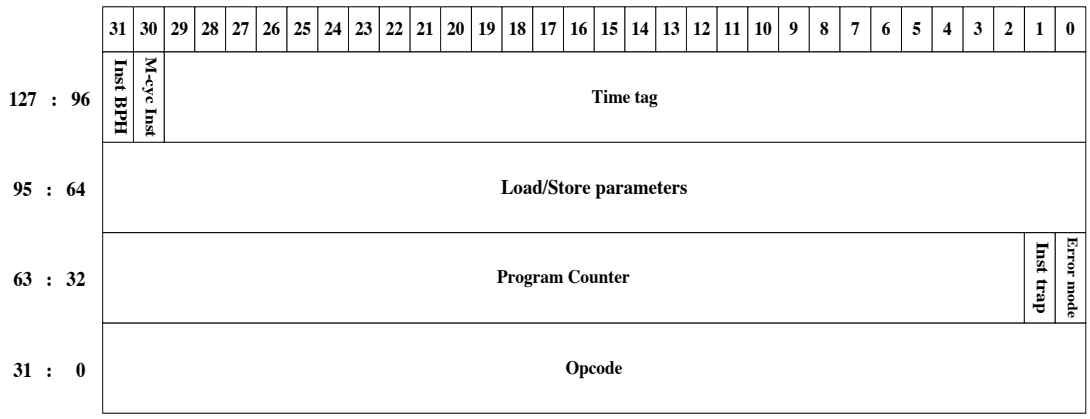


表 8-1 跟踪缓冲区数据分配，指令跟踪模式

Bits	Name	定义
127	指令断点命中	如果出现DSU指令断点命中，设置为‘1’
126	多周期指令	多周期指令(LDD,ST或FPOP)的第二、三instance时设为‘1’
125:96	时间标志	时间标志计数器的值
95:64	Load/Store参数	指令结果、存储地址或者存储数据
63:34	程序计数器	程序计数器（不记录最低两位，因为它们总是0）
33	指令陷阱	如果被跟踪的指令进入陷阱则设置为‘1’
32	处理器错误模式	如果被跟踪的指令引起处理器进入错误模式则设置为‘1’
31:0	Opcode	指令操作码

当跟踪被冻结时，产生11号中断。

总线跟踪

通过使跟踪缓冲区控制寄存器的跟踪 AHB 使能位为 1 使能总线跟踪模式。

在总线跟踪期间，跟踪缓冲区的一行记录一次内部总线上的操作。记录格式如下：

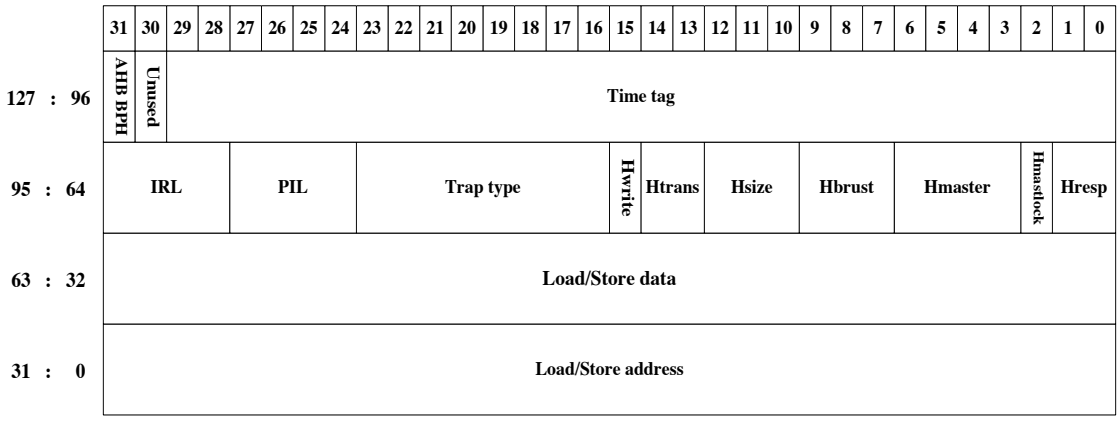


表 8-2 跟踪缓冲区数据分配，内部总线跟踪模式

Bits	Name	定义
127	AHB 断点命中	如果 DSU AHB 断点命中则设为‘1’
126	-	未使用
125:96	时间标志	时间标志计数器的值
95:92	IRL	处理器中断请求输入
91:88	PIL	处理器中断级别(psr.pil)
95:80	trap type	处理器陷阱类型(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

混合跟踪

在混合模式下，缓冲区被分为两半：指令存在低一半，总线传输存在高一半。AHB 索引计数器的 MSB 位自动保持为高，指令索引计数器的 MSB 保持为低，即指令跟踪记录的起始地址为 0x9001-0000，AHB 跟踪记录的起始地址为 0x9001-1000。

DSU 存储器映射

表8-3 DSU存储器映射表

Address	**Register**
0x8000-00c4	DSU串口状态寄存器
0x8000-00c8	DSU串口控制寄存器
0x8000-00cc	DSU串口分频寄存器
0x9000-0000	DSU控制寄存器
0x9000-0004	Trace buffer控制寄存器
0x9000-0008	Time tag counter
0x9000-0010	AHB断点地址1
0x9000-0014	AHB掩码1
0x9000-0018	AHB断点地址2
0x9000-001C	AHB掩码2

0x9001-0000 - 0x9001-1FFF	跟踪缓冲区
...0	Trace bits 127 - 96
...4	Trace bits 95 - 64
...8	Trace bits 63 - 32
...C	Trace bits 31 - 0
0x9002-0000 - 0x9002-029F	IU/FPU寄存器堆（详细说明见附录）
0x9008-0000	Y寄存器
0x9008-0004	PSR寄存器
0x9008-0008	WIM寄存器
0x9008-000C	TBR寄存器
0x9008-0010	PC寄存器
0x9008-0014	NPC寄存器
0x9008-0018	FSR寄存器
0x9008-001C	DSU陷阱寄存器 ③
0x9008-0048	ASR18
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	指令cache tags ①
0x9010-0000-0x9010-1FFF	第一路
0x9010-2000-0x9010-3FFF	第二路
0x9010-4000-0x9010-5FFF	第三路
0x9010-6000-0x9010-7FFF	第四路
0x9014-0000 - 0x9014-7FFF	指令cache数据
0x9014-0000-0x9014-1FFF	第一路
0x9014-2000-0x9014-3FFF	第二路
0x9014-4000-0x9014-5FFF	第三路
0x9014-6000-0x9014-7FFF	第四路
0x9018-0000 - 0x9018-3FFF	数据cache tags ②
0x9018-0000-0x9018-1FFF	第一路
0x9018-2000-0x9018-3FFF	第二路
0x901C-0000 - 0x901C-3FFF	数据cache数据
0x901C-0000-0x901C-1FFF	第一路
0x901C-2000-0x901C-3FFF	第二路

注①：由于 Cache 结构，一行指令 Cache 的 TAG 在跟踪缓冲区中的映射为连续的 8 个字，读写 8 个字中的任何一个都是对同一个 TAG 操作

注②：由于 Cache 结构，一行数据 Cache 的 TAG 在跟踪缓冲区中的映射为连续的 4 个字，读写 4 个字中的任何一个都是对同一个 TAG 操作

注③：DSU 陷阱寄存器(DTR)是一个只读寄存器，指出哪种 SPARC 陷阱导致处理器进入调试模式。当通过设置 DSU 控制寄存器中的 BN 位强制进入调试模式时，陷阱类型是 0x0B。

DSU 断点

DSU 包含了两个断点寄存器来匹配任何内部的总线地址或者执行的指令。断点命中的典型应用是冻结跟踪缓冲区，但也可以把处理器置于调试模式。

冻结操作可以被延迟，通过把 DSU 控制寄存器中的 TDELAY 字段置为一非零的值来实现。这种情况下，TDELAY 的值会在每一次额外的跟踪后减一，递减至零时冻结跟踪缓冲区。如果 DSU 控制寄存器中的 BT 置 1，则跟踪缓冲区冻结时，DSU 强制处理器进入调试模式。

注：因为流水延迟，在处理器被置于调试模式前，最多有 4 个额外的指令可能被执行。

每个断点包含一个掩码寄存器，通过该寄存器可以允许在一段地址上触发断点。断点检测过程中，掩码寄存器中置“1”的掩码位与地址比较，控制断点位置。

8.3 DSU 通讯连接

DSU 通讯连接包含一个串口，该串口在内部总线上作为一个主设备。

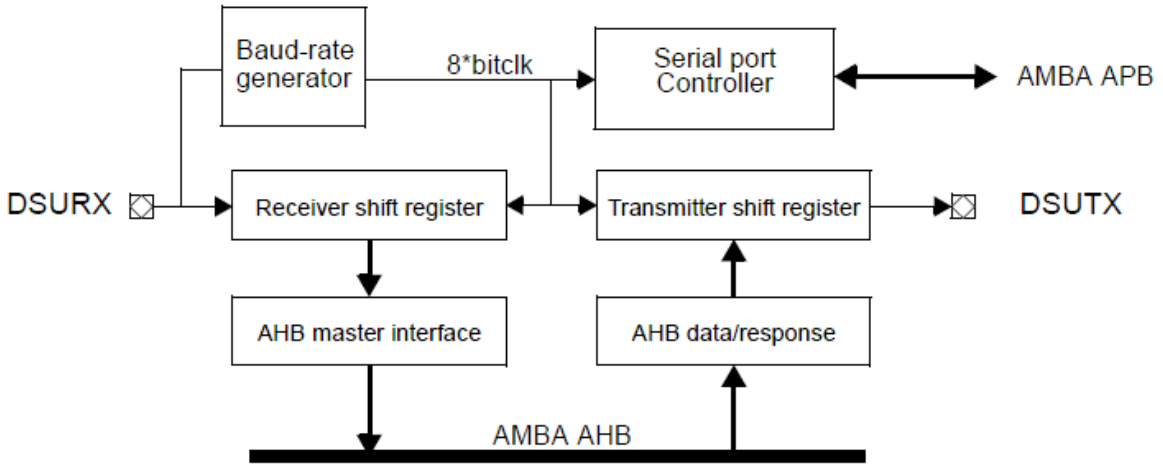


图 8- 2 DSU 通讯连接块图

该串口支持简单的通信协议传输访问参数和数据。连接命令由一个控制字节、32 位地址和可选的写入数据组成。如果 DSU 控制寄存器的

LR 位置位，则每次 AHB 传输之后发送响应的字节。如果未设置 LR 位，则写访问不返回任何数据，读访问只返回读到的数据。

数据帧

数据以 8-bit 为基础进行发送：

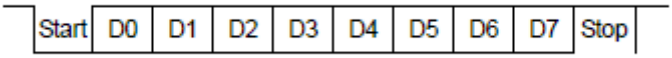


图 8- 3 DSU 串口数据帧

DSU 控制命令

通过通讯链路，可以对内部总线上任何地址进行读/写操作。当处理器从执行模式进入调试模式时，可以选择是否发送一个响应字节。通过设置长度字段为 n-1（其中 n 标明传输字的长度），可以进行块传输。写访问时，发送控制字节、目标地址段首地址以及需要写入的数据，数据写入地址以目标地址段首地址为基地址，以字为单位自动递增。读访问时，发送控制字节和目标地址段首地址，返回相应大小的数据。

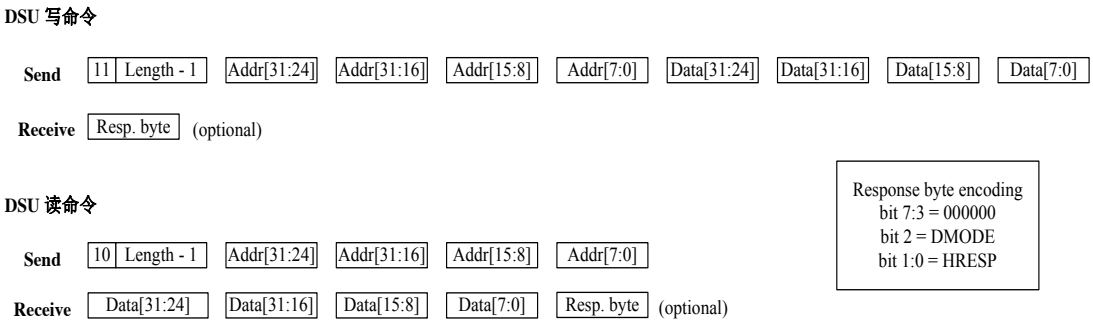


图 8- 4 DSU 命令

串口通信时钟产生

串口通过一个 18 位递减计数的分频器来产生需要的波特率。分频器由系统时钟控制，每次下溢时产生一个串口 tick。每次下溢之后分频器重新载入串口重载寄存器的值。串口 tick 频率应该是期望波特率的 8 倍。

如果不进行软件编程，波特率能够被自动检测。这通过寻找接收数据(相应于两个 bit 周期)的两个下降沿的最短周期实现。当发现三个相同的 2-bit 周期时，相应的分频器重载值锁存入重载寄存器，设置 UART 控制寄存器中的 BL 位。如果 BL 位通过软件复位，波特率发现过程重新开

始。当接收器收到一个‘break’时，波特率发现重新开始，这允许外部的发送器改变波特率。为了正确的监测波特率，在复位或者发送‘break’之后应该向接收器发送值 0x55。

分频值计算公式如下：

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

8.4 从 DSU 引导

在复位时使 DSUEN 和 DSUBRE 有效，将使处理器直接进入调试模式而不执行任何指令。此时可以通过通讯连接初始化系统，下载程序进行调试。另外，可以编程外部引导 PROM（flash）。

九 片内寄存器

9.1 片内控制寄存器的地址

地址	寄存器描述	地址	寄存器描述
0x8000-0000	存储器控制寄存器 MCFG1	0x8000-00A0	I/O 端口输入输出寄存器
0x8000-0004	存储器控制寄存器 MCFG2	0x8000-00A4	I/O 端口方向寄存器
0x8000-0008	存储器控制寄存器 MCFG3	0x8000-00A8	I/O 端口中断配置寄存器
0x8000-000C	AHB 总线失效地址寄存器		
0x8000-0010	AHB 状态寄存器		
0x8000-0014	Cache 控制寄存器		
0x8000-0018	Power-down 寄存器		
0x8000-001C	写保护寄存器 1		
0x8000-0020	写保护寄存器 2	0x8000-00C4	DSU 串口状态寄存器
0x8000-0024	产品配置寄存器	0x8000-00C8	DSU 串口控制寄存器
0x8000-0040	定时器 1 计数寄存器	0x8000-00CC	DSU 串口通信分频寄存器
0x8000-0044	定时器 1 总数重载寄存器		
0x8000-0048	定时器 1 控制寄存器	0x8000-00D0	PCI 复位控制
0x8000-004C	看门狗寄存器		
0x8000-0050	定时器 2 计数寄存器		
0x8000-0054	定时器 2 总数重载寄存器		
0x8000-0058	定时器 2 控制寄存器	0x8000-00E0	UART3 数据寄存器
0x8000-0060	预分频 计数寄存器	0x8000-00E4	UART3 状态寄存器
0x8000-0064	预分频 重载寄存器	0x8000-00E8	UART3 控制寄存器
0x8000-0070	UART1 数据寄存器	0x8000-00EC	UART3 预分频寄存器
0x8000-0074	UART1 状态寄存器		
0x8000-0078	UART1 控制寄存器	0x8000-0100	存储器容错配置寄存器 MECFG1
0x8000-007C	UART1 预分频寄存器	0x8000-0104	存储器容错配置寄存器 MECFG2
0x8000-0080	UART2 数据寄存器	0x8000-0108	存储器容错配置寄存器 MECFG3
0x8000-0084	UART2 状态寄存器	0x8000-010C	存储器容错配置寄存器 MECFG4
0x8000-0088	UART2 控制寄存器		
0x8000-008C	UART2 预分频寄存器	0x8000-0110	扩展 Cache 控制器 1
0x8000-0090	中断屏蔽和优先寄存器	0x8000-0114	扩展 Cache 控制器 2
0x8000-0094	中断等待处理寄存器	0x8000-0118	扩展 Cache 控制器 3
0x8000-0098	中断强制寄存器	0x8000-011C	保留
0x8000-009C	中断清除寄存器		

9.2 AHB 状态寄存器

包括 AHB 失效地址寄存器（0x8000-000C）和 AHB 状态寄存器（0x8000-0010），失效地址寄存器存储访问的地址，而状态寄存器则存储访问的错误类型，当错误发生时寄存器将被更新，并且 NE 位被置位。当 NE 位被置位，中断 1 将产生，通知处理器这个错误，一个错误处理完成之后，NE 位必须被软件重新复位。

0x8000-000C 记录当访问发生错误时的总线地址。

0x8000-0010 记录如下的信息：

- HSIZE（2:0）：HSIZE 传输的大小；
- HMAST（6:3）：记录访问失效时的 HMASER，即总线主设备的值;
- RW（7）：读写，当读周期发生错误时，此位置 1，否则清零；
- NE（8）：错误有效位，当错误发生时，置位；
- （31:9）：保留。

任意时刻读写这两个寄存器可以获得当前总线的读写状态，读寄存器 0x8000-000C 地址的值就是 0x8000-000C，读写 0x8000-0010 地址将得到 0x0000-0082，表明是读状态，总线读写的大小是 2 表明是四个字节。

9.3 Power-down 寄存器

idle（31:0）（只写）

通过对地址 0x8000-0018 写入任意值可以实现此功能，写入任意值之后，将进入到 LOAD/STORE 指令模式。为了更快的进入到 Power-down 模式，建议向 0x8000-0018 写入任意值之后，再跟一“dummy”load 指令。在 Power-down 模式下，IU 单元处于悬挂状态，当未屏蔽的中断进入时，Power-down 模式才会被终止，在 Power-down 模式下其他功能和外设都处于正常状态，此时 IU 单元将停止。实际的工作原理是停止了 DCache 对总线上存储器的访问过程，让 DCache 处于数据存取状态中，从而让 IU 停下来。因此对于不可 Cacheable 的存储空间的访问，Power-down 产生的效果才更为明显

9.4 写保护寄存器 WPR

地址为：0x8000-001C 和 0x8000-0020

EN（31）	BP（30）	TAG（29:15）	MASK（14:0）

- MASK（14:0）：地址掩码位选择块的大小（4）；

- TAG（29:15）：比较地址（3）；
- BP（30）：如果设置则确定块保护模式（2）；
- EN（31）：使能位（1）。

仅仅对地址空间从 0x4000-0000 开始到 0x7FFF-FFFF 之间的地址段具有写保护作用。ROM 的写使能通过设置 MCFG1 中的写使能位进行设置。

说明：

- 设置使能保护单元；
- 选择块保护模式；
- 比较地址用于选择进行保护的地址；
- 掩码地址用于选择不同的块的大小。

写保护信号有效的基本条件是，首先对读写的地址与 TAG 进行 XOR 运算，然后与 MASK 进行 AND 运算，当最后地址等于零的情况下称为命中。当使能位置 1，下列两种情况将使得写保护信号有效：

- 如果没有命中，也没有在块模式下，写保护信号有效；
- 命中，处于块保护模式下，写保护信号有效。

注意在这样的设计中，前一个写保护信号给出的条件，这样的一个结果说明当命中某一地址，而不是块模式时，写保护无法给出有效信号。当写保护使能有效，并且在存储器的操作中命中相应的存储器，将进入 0x2b 陷阱。

9.5 产品配置寄存器 PCR

地址为：0x8000-0024

31	30	29	28-26	25	24-20	19-17	16-15	14-12	11-10	9	8	7	6	5-4	3-2	1-0
mmu	dsu	sdctrl	wtpnb	imac	nwindows	icsz	ilsz	dcsz	dlsz	divinst	mulinst	wdog	memstat	fpu	pci	wprt

- wprt (1:0)：WPROTEN 存在与否的标识。初始值是“01”；
- pci (3:2)：“11”，PCI 核的类型；
- fpu (5:4)：“01”浮点处理器是 meiko 类型；
- memstat(6)：“1”，有总线状态寄存器；
- wdog(7)：“1”，有看门狗；

- mulinst(8)：“1”，有硬件乘法器；
- divinst(9)：“1”，有硬件除法器；
- dlsz(11:10)：“10”数据 Cache 的 LINE 的大小 log2 值；
- dcsz(14:12)：“011”数据 Cache 的大小，log2 值 8K；
- ilsz(16:15)：“11” 指令 Cache 的 LINE 的大小 log2 值；
- icsz(19:17)：“011”指令 Cache 的大小，log2 值 8K；
- nwindows(24:20)：“00111” 具有 8 寄存器窗口数目；
- imac(25)：‘1’支持 MAC 指令；
- wtpnb(28:26)：“100” 4 个 Watchpoint；
- sdrctrl(29)：‘1’表明存在 SDRAM 功能；
- dsu(30)：‘1’表明具有 DSU；
- mmu(31)：‘0’，表明没有 MMU。

整个寄存器的初始值是：0x7277-bbdd。

9.6 其他寄存器的设置

其他各个寄存器可以参看所对应模块章节。

十 片内锁相环和时钟

BM3803MG 具有两个完全独立的输入时钟信号，CLK 和 PCI_CLK。

CLK 是处理器工作时时钟信号的输入管脚，在片内连接一锁相环进行倍频，可以通过外部四个管脚（PLL_MIN）设置不同的倍频系数，同时有 BP 信号输入，可以选择不用锁相环而直接采用外部时钟输入。BP 低电平采用锁相环，高电平选择输入的时钟信号，这时锁相环被旁路。PCI_CLK 直接输入到 PCI 模块。

PLL_MIN[3:0]表示锁相环的倍频系数，当 BP 为 0 时，PLL_MIN 不能为零（取零时无法保证锁相环频率的锁定）。PLL_MIN 倍频数从 1 到 15，按照相应的二进制取值。通过 CLK 输入的外部时钟的工作频率是 2MHz-30MHz，内部能够提供的最高工作频率是 250MHz，锁相环的接口电平采用 3.3V，可以满足内部设计的要求。

电源稳定到锁相环稳定时间为 10ms。

附 录

A 寄存器描述

表 A-1 寄存器图例

位号	31	30	29	28	27	26	25	24	23	...	...	...	...	9	8	7	6	5	4	3	2	1	0
域名	域				保留										位								
读/写	r =只读				w =只写										r/w =可读可写								
复位后的初始值	0	4			1	x = 未定义或复位后无影响																	

IU 相关的内部寄存器

表 A-2 处理器状态寄存器（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	保留						EC	EF	PIL				S	PS	ET	CWP				
r				r				r/w				r/w						r	r	r/w				r/w			r/w				
B				3				0				0	0	1	0	0	0	0	1	0				1	0	0	0				

位号	助记符	描述
31:28	IMPL	规范实现标识 定义为 B
27:24	VER	处理器版本号，定义为 3
23	N	整数单元的条件码，表示ALU 的逻辑运算单元的32 位结果是否为负数 1 = 负数 0 = 非负
22	Z	整数单元的条件码，表示运算结果是否是零 1 = 零 0 = 非零
21	V	整数单元的条件码，表示运算结果是否超出32 位数的范围 1 = 溢出 0 = 不溢出
20	C	整数单元的条件码，表示运算的结果是否引起进位 1 = 进位 0 = 不进位
13	EC	表示是否使能协处理器 1 = 使能 0 = 禁能
12	EF	表示是否使能 浮点处理器 1 = 使能 0 = 禁能
11:8	PIL	指定处理器可响应中断的级别

位号	助记符	描述
7	S	表示处理器处于管理模式还是用户模式 1 = 管理模式 0 = 用户模式
6	PS	保存最近一次陷阱发生时S位的值
5	ET	表示陷阱是否使能 1 = 陷阱使能 0 = 陷阱禁用
4:0	CWP	当前窗口指针

表 A-3 窗口无效屏蔽寄存器（WIM Windows Invaild Mask Register，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
保留																								Windows							
																								7	6	5	4	3	2	1	0
r																								r/w							
0																								0							

位号	助记符	描述
0 < n < 7	Windows[n]	表示窗口是否有效 0 = 有效 1 = 无效

表 A-4 异常基址标志寄存器（Trap Base Register ，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/w			
0																												0	0	0	0

位号	助记符	描述
31:12	TBA	陷阱的基地址 由软件设置，包含陷阱表基地址的高20位
11:4	TT	陷阱类型字段 陷阱发生由硬件写入，并保存到下一个陷阱发生
3:0	ZERO	全零

表 A-5 陷阱类型

陷阱	TT	优先级	描述
Reset	0x00	1	上电复位
Write error	0x2B	2	写buffer错误
Instruction_access_error	0x01	3	取指时出错
Illegal_ instruction	0x02	5	UNIMP 或其他未实现指令
Privileged_ instruction	0x03	4	用户模式下特殊指令的执行
Fp_ disabled	0x04	6	禁能FPU 时使用浮点指令
Cp_ disabled	0x24	6	禁能协处理器时使用协处理器指令
Watchpoint_ detected	0x0B	7	硬件断点匹配
Window_ overflow	0x05	8	SAVE 进入无效窗口
Window_ underflow	0x06	8	RESTORE进入无效窗口
Register_ hardware_error	0x20	9	寄存器堆EDAC错误
Mem_ address_not_ aligned	0x07	10	访问寄存器时地址未对齐
Fp_ exception	0x08	11	FPU异常
Cp_ exception	0x28	11	协处理器异常
Privileged_ instruction	0x03	4	用户模式下特殊指令的执行
Fp_ disabled	0x04	6	禁能FPU 时使用浮点指令
Cp_ disabled	0x24	6	禁能协处理器时使用协处理器指令
Watchpoint_ detected	0x0B	7	硬件断点匹配
Window_ overflow	0x05	8	SAVE 进入无效窗口
Window_ underflow	0x06	8	RESTORE进入无效窗口
Reset	0x00	1	上电复位
Write error	0x2B	2	写buffer错误
Instruction_access_error	0x01	3	取指时出错
Illegal_ instruction	0x02	5	UNIMP 或其他未实现指令
Privileged_ instruction	0x03	4	用户模式下特殊指令的执行
Fp_ disabled	0x04	6	禁能FPU 时使用浮点指令
Cp_ disabled	0x24	6	禁能协处理器时使用协处理器指令
Watchpoint_ detected	0x0B	7	硬件断点匹配
Window_ overflow	0x05	8	SAVE 进入无效窗口
Window_ underflow	0x06	8	RESTORE进入无效窗口
Watchpoint_ detected	0x0B	7	硬件断点匹配
Window_ overflow	0x05	8	SAVE 进入无效窗口
Window_ underflow	0x06	8	RESTORE进入无效窗口

表 A-6 乘法/除法寄存器（Multiply/Divide 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																															
0																															

表 A-7 辅助状态寄存器（Ancillary State Registers， ASR）

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				**R**					**TCB**							**CNT**															
r/w				**r**					**r/w**							**r**															
1111				**0**					**x**							**x**															

位号	名称	描述
31:28	**CB**	**0000 = 错误计数** **0100 = 错误计数清零** **1010 = 造错使能，即对及数据位和校验位造错**
22:16	**TCB**	**对校验位造错**
15:0	**CNT**	**记录发生一位错的次数**

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																															

位号	助记符	描述
31:0	DCB	数据位造错

观察点寄存器（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																														保留	IF
r/w																														r	r/w
x																														0	0

位号	助记符	描述
31:2	WADDR	表示观察点的地址范围
0	IF	取值使能 1 = 使能 0 = 禁能

屏蔽地址寄存器（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

位号	助记符	描述
31:2	WMASK	表示与观察点的比较位 1 = 比较使能 0 = 比较禁能
1	DL	表示是否使能加载数据命中 1 = 使能加载 0 = 禁能加载
0	DS	表示是否使能存储数据命中 1 = 使能 0 = 禁能

浮点寄存器

表 A-8 浮点状态寄存器（Floating Point State 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		保留		TEM					NS	保留		VER			FTT			保留		FCC		AEXEC					CEXEC				
r/w		r/w		r/w					r	r/w		r			r			r/w		r		r					r				
x		x		0					x	x		0			x			0		x		x					0				

位号	助记符	描述
31:30	RD	舍入方向
27:23	TEM	陷阱使能掩码 1 = 陷阱使能 0 = 陷阱禁能
22	NS	FPU 实现标志位
19:17	VER	表示一个或多个FPU的实现位
16:14	FTT	浮点陷阱类型
11:10	FCC	浮点条件代码位
9:5	AEXEC	TEM禁能fp_exception异常时保存IEEE浮点异常。当浮点操作执行完，TEM和cexc逻辑与，结果非0，产生fp_exception异常，否则，新的cexc值与aexc相或并保存在aexc中。因此，当屏蔽陷阱时，异常保存在aexc中
4:0	CEXEC	表示最近执行Fpop指令所引起的一个或多个IEEE浮点异常

表 A-9 FSR 的浮点陷阱类型（FTT）字段

FTT	陷阱类型	描述
0	None	无陷阱
1	IEEE_754 exception	表示产生一个符合IEEE 754－1985标准的陷阱
2	Unfinished_FPop	表示一个实现的FPU不能产生ANSI/IEEE 754-1985标准所定义的正确结果或陷阱
3	Unimplemented_FPop	表示未被实现的FPop解码的一个FPU实现
4	Sequence_error	表示FPU中三种异常错误条件之一
5	Hardware_error	表示FPU发现发生了严重内部错误
6	Invaild_fp_register	表示FPop中的一个（或多个）操作数未对齐

表 A-10 FSR 的浮点数条件码字段（FCC）

FCC	描述
0	f_{rs1} = f_{rs2}
1	f_{rs1} < f_{rs2}
2	f_{rs1} > f_{rs2}
3	f_{rs1} ? f_{rs2} （无序） 如果f_{rs1} 或者 f_{rs2} 是signaling NaN或quiet NaN，表示两者无法比较
f_{rs1} 和 f_{rs2} 指存放在f寄存器中单精度或双精度浮点数的值。 注意：当FCMP或者FCMPE指令产生IEEE_exception陷阱时FCC并不改变。	

表 A-11 浮点数异常字段

AEXC	CEXC	名称	描述
NVA	NVC	Invalid	表示将要执行操作的操作数是否为无效操作数 1 = 无效操作数　0 = 合法操作数
OFA	OFC	Overflow	表示在数据格式中舍入后的结果是否会上溢 1 = 上溢　0 = 未上溢
UFA	UFC	Underflow	表示在制定的数据格式中舍入的结果是否会下溢 1 = 下溢　0 = 不下溢
DZA	DZC	Division-by-zero	X÷0，X是欠规格化或规格化的 注意：0÷0不设置dzc位 1=被零除，0=未被零除
NXA	NXC	Inexact	与无限精确的正确结果相比，表示舍入结果是否可认为是精确结果 1 = 不精确　0 = 精确

存储器接口相关寄存器

表 A-12 存储器控制寄存器 1（MCFG 1）

Address = 0x8000-0000

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
保留			IOWDH		BRDY	BEXC	保留	IOWS				IOEN	保留		PRWWS						PRWEN	保留	PRWDH		保留					PRRWS			
r			r/w		r/w	r/w	r	r/w				r/w	r		r/w						r/w	r	r/w		r					r/w			
0			x		0	0	0	F				0	0		11111						0	0	x		0					11111			

位号	助记符	描述
31:29 24 7:5		保留
28:27	IOWDH	I/O设备总线宽度（”00”=8，“01”=16，“1x”=32）
26	BRDY	总线准备好(BRDYN) 使能，用于I/O设备。高电平有效。
25	BEXC	总线错误(BEXCN)使能，用于全部外部存储器空间（PROM、I/O和RAM），高电平有效。
23:20	IOWS	I/O设备访问等待周期数。初始值为15。(“0000”=0,“0001”=1, “0010”=2,…, “1111”=15)
19	IOEN	I/O设备访问使能位 高电平有效
16:12	PRWWS	PROM写等待周期数。初始值为31。(“00000”=0,“00001”=1,“00010”=2,…,“11111”=31)
11	PRWEN	PROM写使能 高电平有效
9:8	PRWDH	PROM 数据宽度(“00”=8,“01”=16, “1x”=32)。此字段的初值由GPIO的第1、0两位配置。
4:0	PRRWS	PROM读等待周期数。初始值为31。(“00000”=0,“00001”=1,“00010”=2,…,“11111”=31)。

表 A-13 存储器控制寄存器 2（MCFG 2）

Address = 0x8000-0004

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				保留	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			
x	x	x			x	x			x		x		F				0	0	x				0	x	x	x		F			

位号	助记符	描述
31	SDREF	SDRAM 刷新，如果置位，SDRAM刷新将被使能；
30	TRP	SDRAM TRP时间。如果清零，则 TRP为2个时钟周期，如果置位，则 TRP为3个时钟周期。
29:27	TRFC	SDRAM TRFC 时间。 TRFC为（设置数值＋3）个系统时钟周期。
26	SDRCAS	SDRAM CAS 延时。如果清零，则 CAS 延时为2个时钟周期，如果置位，则CAS 延时为3个时钟周期。
25:23	SDRBS	SDRAM BANK的大小 “000”=4 Mbyte,“001”=8 Mbyte, “010”=16 Mbyte .... “111”=512 Mbyte
22:21	SDRCLS	SDRAM Column的大小。 “00”=256, “01”=512, “10”=1024, 当[22:21]=“11”并且[25:23]=“111”时，为4096 当[22:21]=“11”并且[25:23]/=“111”时, 为2048。
20:19	SDRCMD	SDRAM 命令。当写入非0数值时，将产生SDRAM命令。 “01”=PRECHARGE,“10”=AUTO-REFRESH,“11”=LOAD-COMMAND-REGISTER.命令执行后，此数值清零。
18:15	SRWWS	SRAM写等待周期数，初始值为15。(“0000”=0,“0001”=1,“0010”=2,…, “1111”=15)。
14	SDREN	SDRAM使能 1 = 使能 0 =禁能
13	SRDIS	SRAM禁能 1 = 禁能 0 =使能
12:9	SRBS	SRAM bank 大小。定义每个SRAM大小。 “0000”=8 Kbyte, “0001”=16 Kbyte , … ,“1111”=256 Mbyte
7	BRDY	总线准备好使能 1 = 使能并用于第5 BANK的 SRAM
6	RMW	读改写方式使能位，高电平有效
5:4	SRWDH	SRAM数据总线宽度 ‘00’=8 ‘01’=16 ‘1x’=32
3:0	SRRWS	SRAM读等待周期数 初始值为15 ‘0000’=0 ‘0001’=1 ‘0010’=2 ...‘1111’=15

表 A-14 存储器控制寄存器 3（MCFG 3）

Address = 0x8000-0008

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
保留					SDRRV															保留											
r					r/w															r											
0					x															0											

位号	助记符	描述
26:12	SDRRV	SDRAM刷新计数器重载值 计算AUTO-REFRESH命令间隔时间公式为：$T_{REFRESH}$ = ((reload value) + 1) / SYSCLK

表 A-15 存储器容错配置寄存器 1 （MECFG 1）

Address = 0x8000-0100

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	保留			EWB2	ERB	EWB1	SDEEN	EWB0	PRBS				保留				TCBAREA		保留	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						

位号	助记符	描述
31	SR1EEN	SRAM BANK1 EDAC 使能 1 = 使能 0 = 禁能
30	SR2EEN	SRAM BANK2 EDAC 使能 1 = 使能 0 = 禁能
29	SR3EEN	SRAM BANK3 EDAC 使能 1 = 使能 0 = 禁能
28	SR4EEN	SRAM BANK4 EDAC 使能 1 = 使能 0 = 禁能
27	SR5EEN	SRAM BANK5 EDAC 使能 1 = 使能 0 = 禁能
26	ROMEEN	Prom EDAC使能，1 = 使能 0 = 禁能 此字段的初值在上电复位时由GPIO的第2位配置。
22	EWB2	EDAC写旁路 EWB2、1、0 = “011”有效
21	ERB	EDAC读旁路使能， 1 = 使能 0 = 禁能
20	EWB1	EDAC写旁路使能，EWB2、1、0 = “011”有效
19	SDEEN	SDRAM EDAC使能，1 = 使能 0 = 禁能
18	EWB0	EDAC写旁路，EWB2、1、0 = “011”有效
17:14	PRBS	PROM bank size。定义每个PROM大小。 “0000”=8 Kbyte, “0001”=16 Kbyte , … , “1111”=256 Mbyte 用于8位prom的EDAC。此字段的初值在上电复位时由外部管脚ROMBSD [3:0]配置。
9:8	TCBAREA	TCB area。将每个bank的SRAM和每个bank的SDRAM平均分成4份区域。在通过设置此字段而指向的区域中，被访问的数据的校验位才可以存入TCB字段。第5个bank的SRAM除外，其不受TCBarea字段控制。
6:0	TCB	Test check bits(TCB)。用于测试校验，如果MECFG1中ERB被使能，且对于32位数据宽度SRAM和SDRAM，那么在读数据时，根据TCBarea字段，读得的正常的校验位存入此字段。

表 A-16 存储器容错配置寄存器 2 （MECFG 2）

Address = 0x8000-0104

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																															

位号	助记符	描述
31:0	DR	Data Reversal数据位翻转使能位 1 = 使能 0 = 禁能

表 A-17 存储器容错配置寄存器 3 （MECFG 3）

Address = 0x8000-0108

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
？？																									PR						
r																									r/w						
0																									x						

？？	？？？	？？
6:0	PR	Pardata Reversal？？？？？？？？ 1 = ？？ 0 = ？？

表 A-18 存储器容错配置寄存器 4 （MECFG 4）

Address = 0x8000-010C

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

位号	助记符	描述
31:0	MCTRL	版本号

表 A-19 写保护寄存器 1（Write Protect Register1，WPR1）

Address = 0x8000-001C　0x8000-0020

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														

位号	助记符	描述
31	EN	使能位　1 = 使能　0 = 禁能
30	BP	比较地址位
29:15	TAG	块保护模式设置位　若设置则块模块使能
14:0	MASK	地址掩码位 选择块的大小

表 A-20 写保护寄存器 2（Write Protect Register2，WPR2）

Address = 0x8000-0020

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														

位号	助记符	描述
31	EN	使能位　1 = 使能　0 = 禁能
30	BP	比较地址位
29:15	TAG	块保护模式设置位　若设置则块模块使能
14:0	MASK	地址掩码位 选择块的大小

系统寄存器

表 A-21 产品配置寄存器 （PCR）

Address = 0x8000-0024

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		1	0	1	1	1	0	1	1	1	1		0	1	1	1	0	1	1	1	1	0	1	1	1	0	1

位号	助记符	描述
31	MMU	0 = 表示没有MMU
30	DSU	1 = 表示具有DSU
29	SDRCTRL	1 = 表示存在SDRAM功能
28:26	WTPNB	100 = 表示具有4个Watchpoint
25	IMAC	1 = 表示支持MAC指令
24:20	NWINDOWS	00111 = 具有8个寄存器窗口数目
19:17	ICSZ	011 = 表示指令Cache的大小
16:15	ILSZ	11 = 表示指令Cache的LINE大小
14:12	DCSZ	011 = 表示数据Cache的大小
11:10	DLSZ	10 = 数据Cache的LINE大小
9	DIVINST	1 = 包含硬件除法器
8	MULINST	1 = 包含有硬件乘法器
7	WDOG	1 = 包含有看门狗
6	MEMSTAT	1 = 包含有总线状态寄存器
5:4	FPU	01 = 表示浮点处理器是meiko类型
3:2	PCI	11 = 表示PCI的类型
1:0	WPRT	表示WPROTEN存在与否， 初始值为‘01’

表 A-22 AHB 失效地址寄存器

Address = 0x8000-000C

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																															

位号	助记符	描述
31:0	FA	失效地址位　表示访问发生错误时的总线地址

表 A-23 AHB 状态寄存器

Address = 0x8000-0010

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
保留																							NE	RW	HMAST				HSIZE		
r/w																							r/w	r	r				r		
x																							0	0	0				0		

位号	助记符	描述
8	NE	错误有效位　　1 = 错误发生
7	RW	读写错误有效位　1 = 读周期发生错误　0 = 读周期未发现错误
6:3	HMAST	记录访问失效时HMASER值，即总线主设备值
2:0	HISZE	HSIZE传输大小

Cache 寄存器

表 A-24 Cache 控制寄存器（CCR）

Address = 0x8000-0014

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
LRP		IRP		ISTS		DSTS		DS	FD	FI	保留				IB	IP	DP	保留								DF	IF	DC		ICS	
r		r		r		r		r/w	r/w	r/w	r/w				r/w	r	r	r/w								r/w	r/w	r/w		r/w	
x		x		x		x		x	0	0	x				x	x	x	x								x	x	0		0	

位号	助记符	描述
31:30	LRP	数据Cache的替换策略
29:28	IRP	指令Cache的替换策略
27:26	ISTS	指令Cache的相联数
25:24	DSTS	数据Cache的相联数
23	DS	数据Cache的Snoop使能位
22	FD	数据Cache刷新使能位
21	FI	指令Cache刷新使能位
16	IB	指令取值使能位
15	IP	指令Cache正在刷新
14	DP	数据Cache正在刷新
5	DF	中断时数据Cache冻结控制位
4	IF	中断时指令Cache冻结控制位
3:2	DC	数据Cache状态位　‘01’=Cache冻结　‘11’=Cache使能　‘x0’=禁止
1:0	ICS	指令Cache状态位　‘01’=Cache冻结　‘11’=Cache使能　‘x0’=禁止

表 A-25 Cache 容错控制寄存器（CCR1）

Address = 0x8000_0110

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			保留															CMEEN			CNTRST	CPSEL		保留				EPOS			
r			r															r/w			r/w	r/w		r				r/w			
0	0	1	0															0			0	0		0				0			

位号	助记符	描述
31:9	VER	Cache寄存器版本号，定义为001
13:11	CMEEN	Cache造错使能标志位 ‘011’=造错开启 否则造错无效
10	CNTRST	Cache校验计数器清零标志位 1 =清零所有错误计数器
9:8	CPSEL	Cache造错目标区选择位 ‘00’=指令Cache的data区 ‘01’=指令Cache的tag区 ‘10’=数据Cache的data区 ‘11’=数据Cache的tag区
3:0	EPOS	Cache造错位置的高4位 若选择为data区，对应于数据的4个校验位；若选择为tag区，对应于tag的4个校验位

表 A-26 Cache 造错寄存器（CCR2）

Address = 0x8000_0114

当Cache造错使能，且目标区域为Data区时：

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																															

位号	助记符	描述
31:0	EPOS	造错位置标识低32位

当Cache造错使能，且目标区域为Tag区时：

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									

位号	助记符	描述
31	LOCK	Cache LOCK位
30:12	TAG	Cache TAG区域数据
11:8	VALID	无效位

表 A-27 Cache 错误计数寄存器（CCR3）

Address = 0x8000-0118

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									

位号	助记符	描述
31:26	DTAGCNT	DCache Tag 错误计数器
25:16	DDATACNT	DCache Data错误计数器
15:10	ITAGCNT	ICache Tag 错误计数器
9:0	IDATACNT	ICache Data错误计数器

Power-down 寄存器

表 A-28 Power-down 寄存器

Address = 0x8000-0018

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
IDLE																															
w																															
x																															

位号	助记符	描述
31:0	IDLE	写入任意值 进入LOAD/STORE模式

定时器相关寄存器

表 A-29 定时器 1 计数器寄存器

Address = x8000-0040

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
保留								TIM1VAL																							
r/w								r/w																							
x								x																							

位号	助记符	描述
23:0	TIM1VAL	计数值

表 A-30 定时器 1 重载计数器

Address = 0x8000-0044

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
保留								TIM1RLD																							
r/w								r/w																							
x								x																							

位号	助记符	描述
23:0	TIM1RLD	重载计数值

表 A-31 定时器 1 控制寄存器

Address = 0x8000-0048

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
保留																											WTDEN1	保留	LD1	RL1	EN1
r/w																											r/w	r/w	r/w	r/w	r/w
x																											x	x	x	x	0

位号	助记符	描述
4	WTDEN1	Watchdog使能位
2	LD1	重载计数器使能位　1 = 重载使能　0 = 重载禁能
1	RL1	自动重载使能位　1 = 计数结束时自动重载使能
0	EN1	定时器使能设置位

表 A-32 看门狗寄存器

Address = 0x8000-004C

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
保留								WDC																							
r/w								r/w																							
x								FFFFFF																							

位号	助记符	描述
23:0	WDC	重载计数值

表 A-33 定时器 2 计数器寄存器

Address = 0x8000-0050

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
保留								TIM2VAL																							
r/w								r/w																							
x								x																							

位号	助记符	描述
23:0	TIM2VAL	计数值

表 A-34 定时器 2 重载计数器值

Address = 0x8000-0054

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
保留								TIM2RLD																							
r/w								r/w																							
x								x																							

位号	助记符	描述
23:0	TIM2RLD	重载计数值

表 A-35 定时器 2 控制寄存器

Address = 0x8000-0058

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
保留																													LD2	RL2	EN2
r/w																													r/w	r/w	r/w
x																													x	x	0

位号	助记符	描述
2	LD2	重载计数器使能位　1 = 重载使能　0 = 重载禁能
1	RL2	自动重载使能位　1 = 计数结束时自动重载使能
0	EN2	定时器使能设置位

表 A-36 预分频器计数器寄存器

Address = 0x8000-0060

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
保留																						COUNTER VALUE									
r/w																						r/w									
x																						x									

位号	助记符	描述
9:0	COUNTER VALUE	SDRAM刷新计数器重新加载值

表 A-37 预分频器重载值寄存器

Address = 0x8000-0064

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
保留																						RELOAD VALUE									
r/w																						r/w									
x																						3FF									

位号	助记符	描述
9:0	RELOAD VALUE	重载计数值

UART 口寄存器

表 A-38 UART1 收发数据寄存器

Address = 0x8000-0070

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
保留																								RTD							
r/w																								r/w							
0																								x							

位号	助记符	描述
7:0	RTD	数据位（发送数据、接收数据）

表 A-39 UART1 状态寄存器

Address = 0x8000-0074

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
保留																									FE	PE	OV	BR	TH	TS	DR
r/w																									r/w	r/w	r/w	r/w	r	r	r
x																									x	x	x	x	x	x	x

位号	助记符	描述
6	FE	表示检测到一个帧错误
5	PE	表示检测到一个校验错误
4	OV	表示检测到一个或多个overrun
3	BR	表示受到一个break
2	TH	表示传输保持寄存器为空
1	TS	表示传输移位寄存器为空
0	DR	表示可获得新的数据

表 A-40 UART1 控制寄存器

Address = 0x8000-0078

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
保留																							EC	LB	FL	PE	PS	TI	RI	TE	RE
r/w																							r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
x																							x	x	x	x	x	x	x	x	x

位号	助记符	描述
8	EC	外部时钟使能位 若设置，则串口分频器时钟采用PIO[3]
7	LB	环回测试使能位 若设置，则串口进入环回测试模式
6	FL	流控使能位 若设置，则采用CTS、RTS实现流控
5	PE	校验使能位 若设置，则产生校验
4	PS	校验极性选择位 1 = 奇极性 0 = 偶极性
3	TI	发送中断使能位
2	RI	接收中断使能位
1	TE	发送使能位
0	RE	接收使能位

表 A-41 UART1 分频寄存器

Address = 0x8000-007C

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
保留																							SCALER VALUE								
r/w																							r/w								
x																							x								

位号	助记符	描述
8:0	SCALER VALUE	分频器重载值

表 A-42 UART2 收发数据寄存器

Address = 0x8000-0080

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
保留																								RTD							
r/w																								r/w							
0																								x							

位号	助记符	描述
7:0	RTD	数据位（发送数据、接收数据）

表 A-43 UART2 状态寄存器

Address = 0x8000-0084

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
保留																									FE	PE	OV	BR	TH	TS	DR
r/w																									r/tw	r/tw	r/tw	r/tw	r	r	r
x																									x	x	x	x	x	x	x

位号	助记符	描述
6	FE	表示检测到一个帧错误
5	PE	表示检测到一个校验错误
4	OV	表示检测到一个或多个overrun
3	BR	表示受到一个break
2	TH	表示传输保持寄存器为空
1	TS	表示传输移位寄存器为空
0	DR	表示可获得新的数据

表 A-44 UART2 控制寄存器

Address = 0x8000-0088

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
保留																							EC	LB	FL	PE	PS	TI	RI	TE	RE
r/w																							r/tw	r/tw	r/tw	r/tw	r/tw	r/tw	r/tw	r/tw	r/tw
x																							x	x	x	x	x	x	x	x	x

位号	助记符	描述
8	EC	外部时钟使能位 若设置，则串口分频器时钟采用PIO[3]
7	LB	环回测试使能位 若设置，则串口进入环回测试模式
6	FL	流控使能位 若设置，则采用CTS、RTS实现流控
5	PE	校验使能位 若设置，则产生校验
4	PS	校验极性选择位 1 = 奇极性 0 = 偶极性
3	TI	发送中断使能位
2	RI	接收中断使能位
1	TE	发送使能位
0	RE	接收使能位

表 A-45 UART2 分频寄存器

Address = 0x8000-008C

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
保留																								SCALER VALUE							
r/w																								r/w							
x																								x							

位号	助记符	描述
8:0	SCALER VALUE	分频器重载值

表 A-46 UART3 收发数据寄存器

Address = 0x8000-00E0

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
保留																								RTD							
r/w																								r/w							
0																								x							

位号	助记符	描述
7:0	RTD	数据位（发送数据、接收数据）

表 A-47 UART3 状态寄存器

Address = 0x8000-00E4

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
保留																									FE	PE	OV	BR	TH	TS	DR
r/w																									r/tw	r/tw	r/tw	r/tw	r	r	r
x																									x	x	x	x	x	x	x

位号	助记符	描述
6	FE	表示检测到一个帧错误
5	PE	表示检测到一个校验错误
4	OV	表示检测到一个或多个overrun
3	BR	表示受到一个break
2	TH	表示传输保持寄存器为空
1	TS	表示传输移位寄存器为空
0	DR	表示可获得新的数据

表 A-48 UART3 控制寄存器

Address = 0x8000-00E8

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
保留																							EC	LB	FL	PE	PS	TI	RI	TE	RE
r/w																							r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
x																							x	x	x	x	x	x	x	x	x

位号	助记符	描述
8	EC	外部时钟使能位 若设置，则串口分频器时钟采用PIO[3]
7	LB	环回测试使能位 若设置，则串口进入环回测试模式
6	FL	流控使能位 若设置，则采用CTS、RTS实现流控
5	PE	校验使能位 若设置，则产生校验
4	PS	校验极性选择位 1 = 奇极性 0 = 偶极性
3	TI	发送中断使能位
2	RI	接收中断使能位
1	TE	发送使能位
0	RE	接收使能位

表 A-49 UART3 分频寄存器

Address = 0x8000-00EC

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
保留																							SCALER VALUE								
r/w																							r/w								
x																							x								

位号	助记符	描述
8:0	SCALER VALUE	分频器重载值

中断寄存器

表 A-50 中断级别/屏蔽控制寄存器

Address = 0x8000-0090

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															保留	IMASK															保留
r/w															r/w	r/w															r/w
x															x	x															x

位号	助记符	描述
31:17	ILEVEL	中断级别位
15:1	IMASK	中断屏蔽位　1 = 中断使能　0 = 中断屏蔽

表 A-51 中断请求寄存器

Address = 0x8000-0094

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
保留																IPEND															保留
r/w																r															r/w
x																0															x

位号	助记符	描述
15:1	IPEND	中断请求位　1 = 存在一个相应的中断

表 A-52 强制中断寄存器

Address = 0x8000-0098

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
保留																IFORCE															保留
r/w																r/w															r/w
x																0															x

位号	助记符	描述
15:1	IFORCE	强制中断位　1 = 进入中断

表 A-53 中断清除寄存器

Address = 0x8000-009C

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
保留																ICLEAR															保留
r/w																r															r/w
x																0															x

位号	助记符	描述
15:1	ICLEAR	清除中断位　1 = 清除某一位中断位

GPIO 相关寄存器

表 A-54 I/O 端口数据存储器

Address = 0x8000-00A0

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															

位号	助记符	描述
31:24	MEDDAT	与数据总线D[15:8]连接
23:16	LOWDAT	与数据总线D[7:0]连接
15:0	IODATA	I/O 端口数据

表 A-55 I/O 端口方向寄存器

Address = 0x8000-00A4

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
保留														MEDDIR	LOWDIR	IODIR[15：0]															
r/w														r/w	r/w	r/w															
x														x	x	x															

位号	助记符	描述
17	MEDDIR	表示D[15:8]的方向
16	LOWDIR	表示D[7:0]的方向
15:0	IODIR	I/O端口[15:0]的方向 1 = 输出 0 = 输入

表 A-56 IO 端口中断控制寄存器

Address = 0x8000-00A8

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				
x	x	x	x					x	x	x	x					x	x	x	x					x	x	x	x				

位号	助记符	描述
31 23 15 7	EN	使能信号 1 = 相应中断使能 0 = 相应中断屏蔽
30 22 14 6	LE	触发方式 1 = 边沿触发 0 = 电平触发
29 21 13 5	PL	极性 1 = 相应中断上升沿有效 0 = 相应中断下降沿有效
28...24 20...16 12...8 4...0	ISEL	I/O端口选择 确定哪个I/O端口（31：0）产生GPIO中断n

PCI 相关寄存器

表 A-57 PCI 总线复位寄存器（PCI_RESET）

Address = 0x8000-00D0

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
保留																								RESET							
x																								w							
x																								x							

位号	助记符	描述
7:0	RESET	PCI接口复位 05 = 复位

表 A-58 PCI 总线 DMA 写操作起始地址寄存器（WDMA_PCI_ADDR）

Address = 0xC000-0000

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																															
x																															

位号	助记符	描述
31:0	ADDR	DMA写操作在PCI总线上的目标缓存器缓冲区地址 建议使用时，寄存器后两位设为0，此寄存器数据在DMA操作时自动增长

表 A-59 AHB 总线 DMA 写操作起始地址寄存器（WDMA_AHB_ADDR）

Address = 0xC000-0004

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																															
x																															

位号	助记符	描述
31:0	ADDR	DMA写操作在AHB总线上的目标缓存器缓冲区地址 建议使用时，寄存器后两位设为0，此寄存器数据在DMA操作时自动增长

表 A-60 DMA 写操作控制寄存器（WDMA_CONTROL）

Address = 0xC000-0008

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	保留	STOP TRANSFER	START TRANSFER
r/w																								r/w				r/w	r/w	r/w	r/w
x																								x				x	x	x	x

位号	助记符	描述
31:8	TRANSFER SIZE	表示DMA传送的字节数　范围是1－FFFFCH字节 建议进行IO、配置、中断应答DMA操作时使用单字读写方式
7:4	COMMAND	表示在PCI总线上传送数据的DMA命令 DMA写操作传送数据大小和控制寄存器只对WRITE命令敏感
3	PCI INTERRUPT	表示当DMA操作完成时将在PCI总线上产生中断请求　DMA abort不产生中断请求
1	STOP TRANSFER	表示在DMA操作时强行推出，可视为last-resort（最后的手段）
0	START TRANSFER	表示DMA通告是否忙　1 ＝ 发起DMA操作，DMA完成后此位自动清零

表 A-61 在 PCI 总线上传送的 DMA 命令（COMMAND）

通道	命令	描述
读 DMA	0000	中断应答
	0010	I/O 读
	0110	存储器读
	1010	配置读
	1100	存储器多行读
写 DMA	1100	I/O 读
	1100	I/O 写
	1100	存储器写
	1100	配置读

表 A-62 PCI 总线 DMA 读操作起始地址寄存器（RDMA_PCI_ADDR）

Address = 0xC000-0020

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																															
x																															

位号	助记符	描述
31:0	ADDR	DMA读操作在PCI总线上的目标缓存器缓冲区地址 建议使用时，寄存器后两位设为0，此寄存器数据在DMA操作时自动增长

表 A-63 AHB 总线 DMA 读操作起始地址寄存器（RDMA_AHB_ADDR）

Address = 0xC000-0024

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																															
x																															

位号	助记符	描述
31:0	ADDR	DMA读操作在AHB总线上的目标缓存器缓冲区地址 建议使用时，寄存器后两位设为0，此寄存器数据在DMA操作时自动增长

表 A-64 DMA 读操作控制寄存器（RDMA_CONTROL）

Address = 0xC000-0028

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	保留	STOP TRANSFER	START TRANSFER
r/w																								r/w				r/w	r/w	r/w	r/w
x																								x				x	x	x	x

位号	助记符	描述
31:8	TRANSFER SIZE	表示DMA传送的字节数　范围是1－FFFFCH字节 建议进行IO、配置、中断应答DMA操作时使用单字读写方式
7:4	COMMAND	表示在PCI总线上传送数据的DMA命令 DMA读操作传送数据大小和控制寄存器只对WRITE命令敏感
3	PCI INTERRUPT	表示当DMA操作完成时将在PCI总线上产生中断请求　DMA abort不产生中断请求
1	STOP TRANSFER	表示在DMA操作时强行推出，可视为last-resort（最后的手段）
0	START TRANSFER	表示DMA通告是否忙　1 ＝ 发起DMA操作，DMA完成后此位自动清零

表 A-65 CPU 中断掩码寄存器

Address = 0xC000-0040

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
保留			PCI SYSTEM ERROR	PCI INTERRUPT				保留				PCI DOORBELL				保留	AHB-PCI WINDOW			READ DMA				保留	PCI-AHB WINDOW						
																	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/w				r				r/w				r	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
0			0	0				0				0				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

位号	助记符	描述
28	PCI SYSTEM ERROR	在主桥模式下PCI系统错误
27:24	PCI INTERRUPT	在主桥模式下PCI中断
19:16	PCI DOORBELL	在简单桥模式下PCI DOORBELL位
14	AHB-PCI WINDOW DISCARD	AHB-PCI窗口丢弃
13	AHB-PCI WINDOW FETCH ERROR	AHB-PCI窗口取值错误
12	AHB-PCI WINDOW POST ERROR	AHB-PCI窗口存储错误
11	READ DMA AHB ERROR	读DMA模式AHB错误
10	READ DMA PARITY ERROR	读DMA奇偶校验错误位
9	READ DMA ABORT	读DMA中止
8	READ DMA END	读DMA结束
6	PCI-AHB WINDOW DISCARD	PCI-ABH窗口丢弃
5	PCI-AHB WINDOW FETCH ERROR	PCI-AHB窗口取值错误
4	PCI-AHB WINDOW POST ERROR	PCI-AHB存储错误
3	WRITE DMA AHB ERROR	写DMA模式AHB错误
2	WRITE DMA PARITY ERROR	写DMA奇偶校验错误位
1	WRITE DMA ABORT	写DMA中止
0	WRITE DMA END	写DMA结束

表 A-66 CPU 中断状态寄存器

Address = 0xC000-0044

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
保留			PCI SYSTEM ERROR	PCI INTERRUPT				保留				PCI DOORBELL				保留	AHB-PCI WINDOW			READ DMA				保留	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			写1清0 写0无效					r				写1清0 写0无效				r	写1清0 写0无效							r	写1清0 写0无效						
0			0					0				0				0	0							0	0						

位号	助记符	描述
28	PCI SYSTEM ERROR	在主桥模式下PCI系统错误
27:24	PCI INTERRUPT	在主桥模式下PCI中断
19:16	PCI DOORBELL	在简单桥模式下PCI DOORBELL位
14	AHB-PCI WINDOW DISCARD	AHB-PCI窗口丢弃
13	AHB-PCI WINDOW FETCH ERROR	AHB-PCI窗口取值错误
12	AHB-PCI WINDOW POST ERROR	AHB-PCI窗口存储错误
11	READ DMA AHB ERROR	读DMA模式AHB错误
10	READ DMA PARITY ERROR	读DMA奇偶校验错误位
9	READ DMA ABORT	读DMA中止
8	READ DMA END	读DMA结束
6	PCI-AHB WINDOW DISCARD	PCI-ABH窗口丢弃
5	PCI-AHB WINDOW FETCH ERROR	PCI-AHB窗口取值错误
4	PCI-AHB WINDOW POST ERROR	PCI-AHB存储错误
3	WRITE DMA AHB ERROR	写DMA模式AHB错误
2	WRITE DMA PARITY ERROR	写DMA奇偶校验错误位
1	WRITE DMA ABORT	写DMA中止
0	WRITE DMA END	写DMA结束

表 A-67 CPU 中断命令寄存器

Address = 0xC000-0048

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
保留								CPU DOORBELL				保留																			
r/w								r				r/w																			
x								x				x																			

位号	助记符	描述
23:20	CPU DOORBELL	在简单桥模式下CPU DOORBELL位

表 A-68 CPU 状态及版本寄存器

Address = 0xC000-004C

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
保留																TYPE				VERSION											
r/w																r				r											
x																x				x											

位号	助记符	描述
15:12	TYPE	CPU状态 0 = 未定义 1 = 32位AHB-PCI简单桥模式 2 = 32位AHB-PCI主桥模式 3...F = 保留
11:0	VERSION	CPU版本 由IC厂商定义

表 A-69 不可预取范围 PCI-AHB 窗口控制寄存器（PCIAHB_ADDR_NP）

Address = 0xC000-0070

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																								保留							EN
r/w																								r/w							r/w
x																								x							0

位号	助记符	描述
31:8	AHB BASE ADDRESS	简单桥模式下,PCI-AHB基址;主桥模式下，PCI窗口基地址 此地址必须32bit对齐，且可以和AHB设备或AHB总线上未分配地址空间重合
0	WINDOWS ENABLE	PCI总线操作转换位 1 = 使能PCI总线操作转换 0 = 所有PCI操作以Abort结束

表 A-70 PCI-AHB 窗口可预取范围控制寄存器（PCIAHB_ADDR_PF）

Address = 0xC000-0074

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																								保留							EN
r/w																								r/w							r/w
x																								x							0

位号	助记符	描述
31:8	AHB BASE ADDRESS	简单桥模式下,PCI-AHB基址;主桥模式下，PCI窗口基地址 此地址必须32bit对齐，且可以和AHB设备或AHB总线上未分配地址空间重合
0	WINDOWS ENABLE	PCI总线操作转换位 1 = 使能PCI总线操作转换 0 = 所有PCI操作以Abort结束

表 A-71 PCI-AHB 窗口定时丢弃寄存器（PCIAHB_TIMER）

Address = 0xC000-0078

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
保留																				DISCARD TIMER											
r																				r/w											
x																				100											

位号	助记符	描述
11:0	DISCARD TIMER	清空数据 为防止 总线锁死，PCI-AHB窗口定时（PCIAHB_TIMER定义AHB时钟周期的个数）清空数据

表 A-72 AHB-PCI 窗口的 discard 定时器寄存器（AHBPCI_TIMER）

Address = 0xC000-007C

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
保留																				DISCARD TIMER											
r																				r/w											
x																				100											

位号	助记符	描述
11:0	DISCARD TIMER	清空数据 定时清空PCI总线预取数据

表 A-73 PCI 控制寄存器

Address = 0xC000-0080

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
保留																													MasterEN	MemoryEN	BridgePCIReady
r/w																													r/w	r/w	r/w
x																													x	x	x

位号	助记符	描述
2	MasterEN	表示桥是否允许执行DMA 总桥模式总置1
1	MemoryEN	表示访问BAR地址空间是否使能 CPU通过检查它来确定桥是否被初始化 总桥模式总置1
0	BridgePCIrReady	在主桥 模式下，CPU在开始对PCI总线配置访问之前必须等此位置位，PCI reset后此位清除，225个PCI时钟周期之后置位

表 A-74 PCI 设备和供应商商配置寄存器（PCI_DV）

Address = 0xC000-0084

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																															
55551556																															

位号	助记符	描述
31:0	DV	PCI设备和供应商信息位

表 A-75 PCI 子系统设备和子系统供应商配置寄存器（PCI_SUB）

Address = 0xC000-0088

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
SUB_ID																															
r/w																															
0																															

位号	助记符	描述
31:0	SUB_ID	PCI子系统设备和子系统供应商信息位

表 A-76 PCI 分类代码和版本配置寄存器（PCI_CREV）

Address = 0xC000-008C

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
CREV																															
r/w																															
0x04000000																															

位号	助记符	描述
31:0	CREV	PCI分类代码和版本信息位

表 A-77 PCI 仲裁状态寄存器（PCI_BROKEN）

Address = 0xC000-0090

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
保留																								STATUS							
r																								r/w							
x																								x							

位号	助记符	描述
7:0	STATUS	在主桥模式下，表示PCI总线上Master设备的状态 1 = 相应的PCI Master设备禁止，表明严重的系统错误 写1清零 = 使能相应的PCI Master设备

表 A-78 PCI-AHB 窗口不可预取范围屏蔽寄存器（PCIAHB_SIZ_NP）

Address = 0xC000-0094

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																															
FFFC00																								0							

位号	助记符	描述
31:0	SIZ	表示可预取空间大小 低8位恒为0，最小范围为0x100字节

表 A-79 PCI-AHB 窗口可预取范围屏蔽寄存器（PCIAHB_SIZ_PF）

Address = 0xC000-0098

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																															
FFFC00																								0							

位号	助记符	描述
31:0	SIZ	表示可预取空间大小 低8位恒为0，最小范围为0x100字节

表 A-80 PCI 中断掩码寄存器

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
保留								CPU DOORBELL				保留											READ DMA END	保留							WRITE DMA END
r/w								r				r/w											r	r/w							r
x								x				x											x	x							x

位号	助记符	描述
23:20	CPU PC DOORBELL	在简单桥模式下CPU DOORBELL位
8	READ DMA END	在简单桥模式下读DMA结束
0	WRITE DMA END	在简单桥模式下写DMA结束

表 A-81 PCI 中断状态寄存器

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
保留								CPU DOORBELL				保留											READ DMA END	保留							WRITE DMA END
r/w								r				r/w											r	r/w							r
x								x				x											x	x							x

位号	助记符	描述
23:20	CPU PC DOORBELL	在简单桥模式下CPU DOORBELL位
8	READ DMA END	在简单桥模式下读DMA结束
0	WRITE DMA END	在简单桥模式下写DMA结束

表 A-82 PCI 中断命令寄存器

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
保留												PCI DOORBELL				保留																
r/w												r				r/w																
x												x				x																

位号	助记符	描述
19 : 16	PCI DOORBELL	在简单桥模式下PCI DOORBEL位

表 A-83 PCI 状态及版本信息寄存器

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
保留																TYPE				VERSION											
r/w																r				r											
x																x				x											

位号	助记符	描述
15:12	TYPE	PCI状态 0 = 未定义 1 = 32位AHB-PCI简单桥模式 2 = 32位AHB-PCI主桥模式 3...F = 保留
11:0	VERSION	PCI版本 由IC厂商定义

DSU 相关寄存器

表 A-84 DSU 串口控制寄存器

Address = 0x8000-00C4

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
保留																														BL	EN
r/w																														r	r/w
x																														0	x

位号	助记符	描述
1	BL	若自动置位，波特率锁定
0	EN	接收/发送方向使能位

表 A-85 DSU 串口状态寄存器

Address = 0x8000-00C8

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
保留																									FE	保留	OV	保留	TH	TS	DR
r/w																									r	r/w	r	r/w	r	r	r
x																									0	x	0	x	0	0	0

位号	助记符	描述
6	FE	数据帧错误 表明检测到帧错误
4	OV	表示由于溢出，一个或多个字符丢失
2	TH	发送保持寄存器空标志位，表示发送保持寄存器为空
1	TS	发送移位寄存器空标志位，表示发送移位寄存器为空
0	DR	表示已收到数据但尚未被AHB主设备读出

表 A-86 DSU 串口波特率重载寄存器

Address = 0x8000-00CC

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
保留														Scaler Reload Value																	
r/w														r/w																	
x														x																	

位号	助记符	描述
17 : 0	Scaler Reload Value	分频器重载值 = （（系统时钟*10）/（波特率*8） - 5）/10

表 A-87 DSU 断点寄存器

Address = 0x9000-0010　0x9000-0018

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																														保留	EN
r/w																														r/w	r/w
x																														x	x

位号	助记符	描述
31:2	BADD	地址位
0	EN	指令执行时比较

表 A-88 DSU 屏蔽地址寄存器

Address = 0x9000-0014　0x9000-001C

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																														x	x

位号	助记符	描述
31:2	BMA	屏蔽地址位
1	LD	load data比较位
0	ST	Store数据比较位

表 A-89 DSU 控制寄存器

Address = 0x9000-0000

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
DCNT												RE	DR	LR	SS	PE	EE	EB	DM	DE	BZ	BX	BB	BN	BS	BW	BE	FT	BT	DM	TE
r/w												r/w	r/w	r/w	r/w	r/w	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
x												x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

位号	助记符	描述
31:20	DCNT	TraceBuffer延迟计数器　占用位数依赖于TraceBuffer大小，目前为（8 down to 0）九位宽
19	RE	复位错误模式 1 = 清除处理器的错误模式
18	DR	调试模式响应　1 = 当处理器进入调试模式，DSU通讯链送出一个响应
17	LR	链接响应　1 = AHB总线传输后，DSU通讯链送出一个响应
16	SS	单步　1 = 处理器执行一条指令后返回调试模式
15	PE	处理器错误模式，当处理器是错误模式时返回1，否则返回0
14	EE	外部DSUEN信号值
13	EB	外部DSUBRE信号值
12	DM	调试模式，表示处理器进入调试模式
11	DE	延迟计数使能，1 = TraceBuffer的延迟计数器在每次存储Trace时递减，当遇到DSU断点并且延迟计数器的值非0，将自动置位
10	BZ	错误陷阱中断，1 = 处理器在遇到除下列陷阱外的所有其他陷阱是进入调试模式： 特权指令、FPU禁止、窗口上溢、异步中断、ticc trap
9	BX	陷阱中断，1 = 处理器在遇到任何陷阱是都将进入调试模式
8	BB	DSU断点中断，1 =遇到DSU断点时强制处理器进入调试模式
7	BN	立即中断，　1 = 强制处理器进入调试模式　　0 = 处理器重新开始执行
6	BS	S/W断点中断，1 = 当断点指令被执行市强制进入调试模式
5	BW	IU观测点中断，1 = 在IU观测点处进入调试模式（0xb陷阱）
4	BE	错误时钟中断，1= 当处理器进入错误处理状态，强制处理器进入调试模式
3	FT	时钟冻结，1 = 在调试模式下，LEON时钟将暂停计时，保护软件应用程序
2	BT	Trace中断　1= 在Trace冻结时产生DSU中断
1	DM	延迟计数模式，1 = 进行AHB总线trace时延迟计数递减　0 = 延迟计数针对指令trace递减
0	TE	Trace使能　1 = 允许使用Trace Buffer

B IU/FPU 寄存器堆地址列表

全局寄存器：

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

窗口寄存器：

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

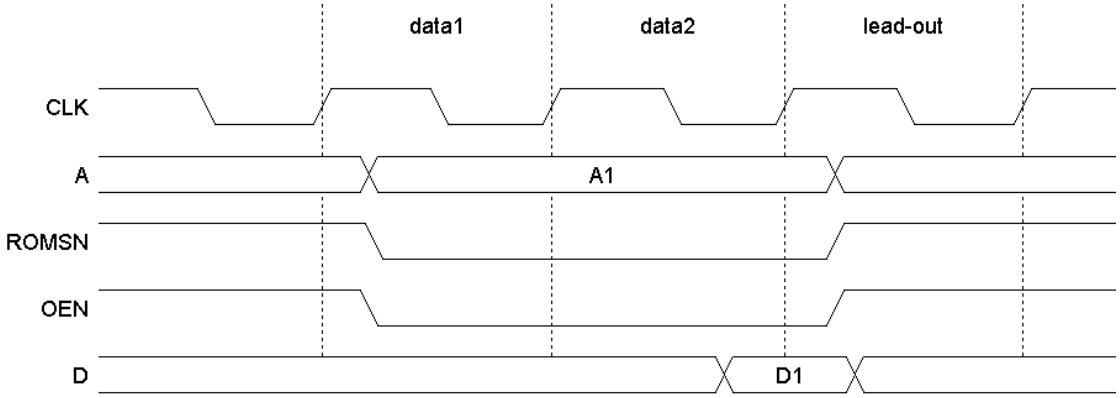
浮点寄存器

%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

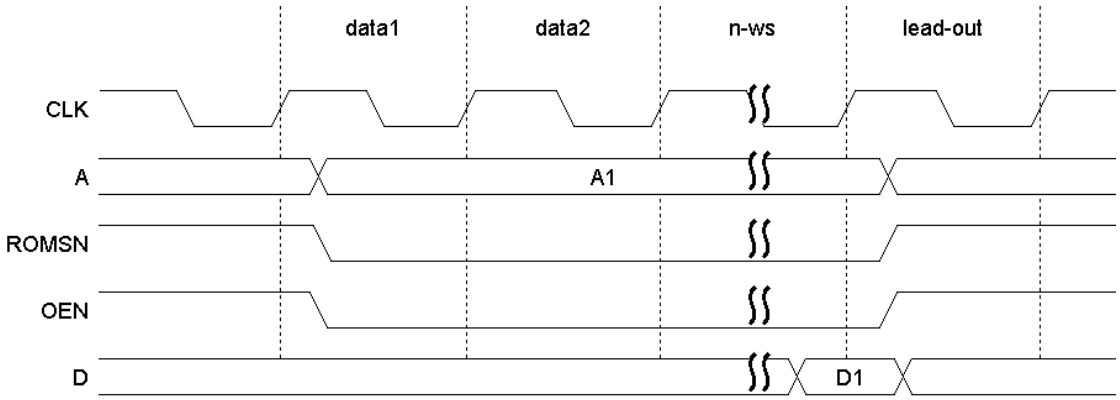
%f16	0x9002-0240	%f24	0x9002-0260
%f17	0x9002-0244	%f25	0x9002-0264
%f18	0x9002-0248	%f26	0x9002-0268
%f19	0x9002-024C	%f27	0x9002-026C
%f20	0x9002-0250	%f28	0x9002-0270
%f21	0x9002-0254	%f29	0x9002-0274
%f22	0x9002-0258	%f30	0x9002-0278
%f23	0x9002-025C	%f31	0x9002-027C

C PROM 和 SDRAM 访问时序

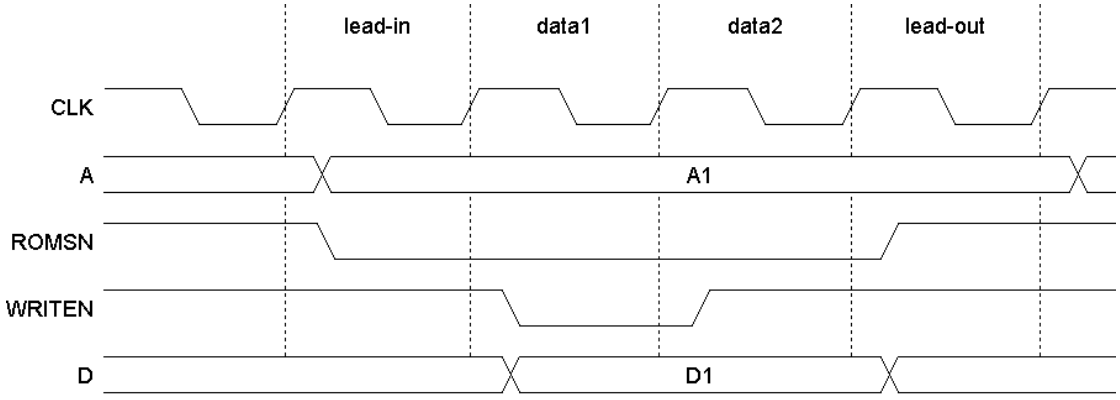
1.PROM 读时序（0 个等待周期）



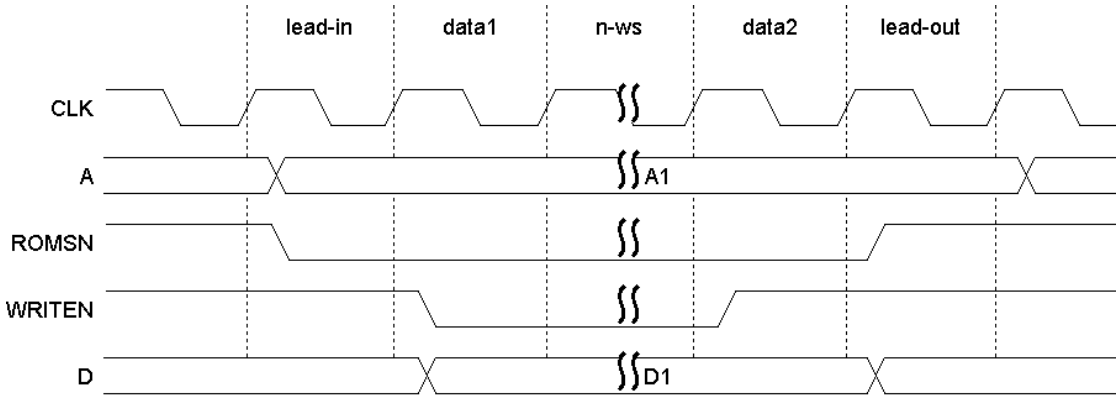
2.PROM 读时序（n 个等待周期）



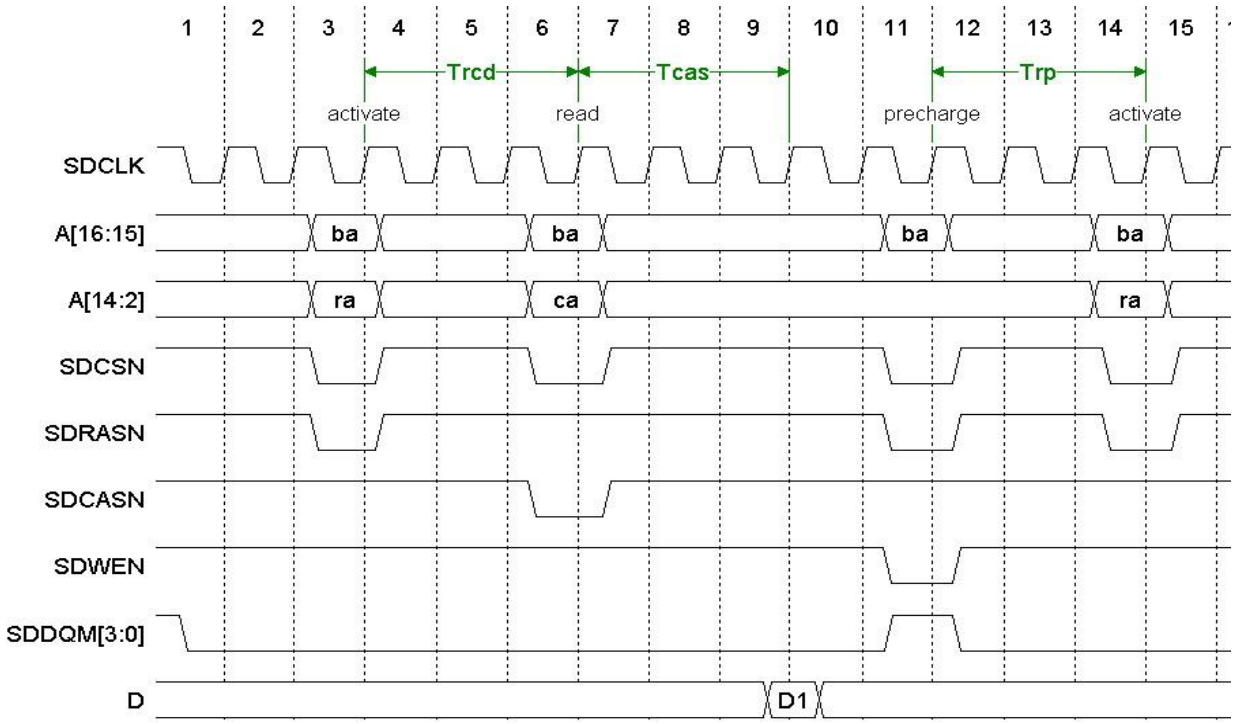
3. PROM 写时序（0 个等待周期）



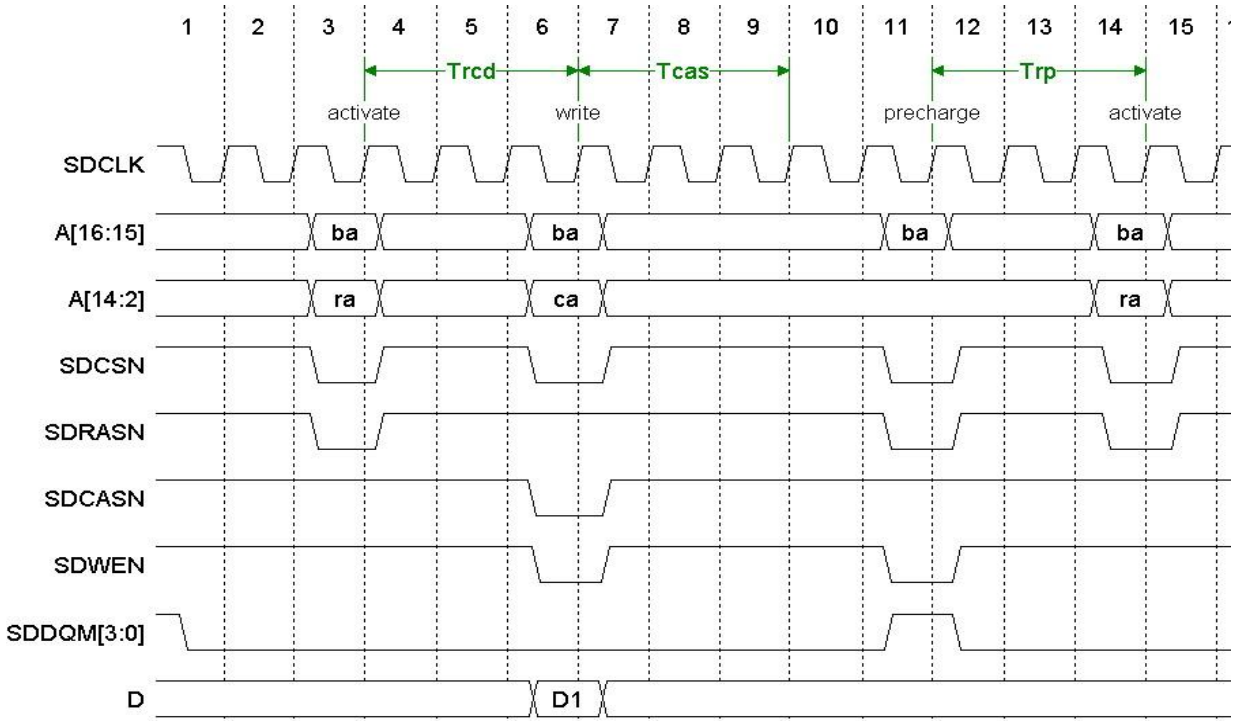
4.PROM 写时序（n 个等待周期）



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



6.SDRAM 写时序（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]
ba
ba
ba
ba
A[14:2]
ra
ca
ra
SDCSN
SDRASN
SDCASN
SDWEN
SDDQM[3:0]
D
D1

D 芯片管脚

全芯片共有 219 个信号功能管脚，具体分配如下：

- IU 和 FPU 管脚（67）：地址总线 28，数据总线 32，校验位 7；
- 存储器接口信号（31）：输出使能，总线准备好信号，读周期指示，写使能，PROM 片选 2，SRAM 输出使能 5，SRAM 片选 5，SRAM 写使能 4，I/O 片选；SDRAM（SDCLK）时钟，SDRAM 列选择（SDCASN），SDRAM 片选 2（SDCSN），SDRAM 行选择（SDRASN），SDRAM 数据字节选择 4（SDDQM）、SDRAM 写使能（SDWEN），PROM bank 大小定义 4（ROMBSD）；
- 系统和时钟信号（4）：CLK 输入，RESET 复位，总线异常 BEXCN，处理器错误 ERRORN；
- 定时器和看门狗输出（3）；
- 并行 I/O 口和串口（20）：并行 I/O 口 16，扩展的串口 4；
- 硬件调试单元信号（5）：DSU（DSUACT）激活信号，DSUBRE 信号，DSU 使能信号（DSUEN），DSU 接受信号，DSU 输出信号；
- PCI 仲裁器（14）：PCI 总线请求 7，PCI 总线响应 7；
- PCI 接口（55）：地址数据线（32）、CBE（4）、奇偶校验、周期结构、初始化准备 IRDY、目标准备信号 TRDY、停止信号、初始化器件选择信号、器件选择、PCI 总线请求、PCI 总线响应、PCI 时钟信号、PCI 复位信号、奇偶校验数据错误，系统错误、PCI 主桥/从桥控制；
- JTAG 信号（5）：TCK（测试时钟控制信号）、TMS（测试模式选择信号）、TDI（测试数据输入）、TDO（测试数据输出）、TRST（测试复位信号）；
- PCI 测试用信号：sim_pci_fast（PCI 快速复位信号）；
- 各种模式信号：scan 模式的选择（scan_mode）；mbist 模式的选择（bist_mode）；scan 模式控制信号（scan_enable）；
- 各个信号和相应的封装管脚对应如表 1-1 所示，输出驱动能力如表 1-2 所示。

表 D-1 信号与管脚对应表

信号名	管脚位置	输入/输出	描述
	G9		
vss_io1	C3	io 地	io 地
vdd_io1	D4	io 电源	io 电源
address[17]	C5	O	处理器地址总线
address[18]	D6	O	处理器地址总线
address[19]	B4	O	处理器地址总线
address[20]	F8	O	处理器地址总线
vss_core1	A3	核地	核地
vdd_core1	E7	核电源	核电源

address[21]	C7	O	处理器地址总线
	F10		
address[22]	B6	O	处理器地址总线
	G11		
address[23]	A5	O	处理器地址总线
address[24]	E9	O	处理器地址总线
vss_io2	A7	io 地	io 地
vdd_io2	D8	io 电源	io 电源
address[25]	C9	O	处理器地址总线
address[26]	D10	O	处理器地址总线
address[27]	B8	O	处理器地址总线
data[0]	E11	I/O	处理器数据总线
data [1]	A9	I/O	处理器数据总线
	F12		
vss_core2	B10	核地	核地
vdd_core2	D12	核电源	核电源
data [2]	C11	I/O	处理器数据总线
	G13		
data [3]	A11	I/O	处理器数据总线
data [4]	E13	I/O	处理器数据总线
data [5]	B12	I/O	处理器数据总线
vss_io3	F14	io 地	io 地
vdd_io3	C13	io 电源	io 电源
data [6]	D14	I/O	处理器数据总线
data [7]	A13	I/O	处理器数据总线
	G15		
data [8]	B14	I/O	处理器数据总线
data [9]	E15	I/O	处理器数据总线
data [10]	C15	I/O	处理器数据总线
vss_core3	F16	核地	核地
vdd_core3	A15	核电源	核电源
data [11]	D16	I/O	处理器数据总线
data [12]	B16	I/O	处理器数据总线
	G17		
data [13]	C17	I/O	处理器数据总线
data [14]	E17	I/O	处理器数据总线
data [15]	A17	I/O	处理器数据总线
vss_io4	F18	io 地	io 地
vdd_io4	B18	io 电源	io 电源
data [16]	D18	I/O	处理器数据总线
data [17]	A19	I/O	处理器数据总线
data [18]	E19	I/O	处理器数据总线
data [19]	C19	I/O	处理器数据总线

	G19		
data [20]	B20	I/O	处理器数据总线
vss_core4	D20	核地	核地
vdd_core4	A21	核电源	核电源
data [21]	F20	I/O	处理器数据总线
data [22]	C21	I/O	处理器数据总线
data [23]	E21	I/O	处理器数据总线
data [24]	B22	I/O	处理器数据总线
	G21		
data [25]	A23	I/O	处理器数据总线
data [26]	D22	I/O	处理器数据总线
vss_io5	C23	io 地	io 地
	F22		
vdd_io5	B24	io 电源	io 电源
data [27]	E23	I/O	处理器数据总线
data [28]	A25	I/O	处理器数据总线
	G23		
data [29]	C25	I/O	处理器数据总线
data [30]	D24	I/O	处理器数据总线
data [31]	B26	I/O	处理器数据总线
vss_core5	F24	核地	核地
vdd_core5	A27	核电源	核电源
pdata [0]	E25	I/O	校验数据总线
pdata [1]	B28	I/O	校验数据总线
pdata [2]	D26	I/O	校验数据总线
vdd_io6	C27	io 电源	io 电源
Clk	D28	I	系统时钟
vss_io6	A29	io 地	io 地
pdata [3]	E27	I/O	校验数据总线
pdata [4]	A31	I/O	校验数据总线
	G25		
pdata [5]	B30	I/O	校验数据总线
pdata [6]	F26	I/O	校验数据总线
pll_rst	C29	I	PLL 复位信号，高电平有效。
pll_min[3]	E29	I	PLL 倍频系数输入
pll_min[2]	A33	I	PLL 倍频系数输入
pll_min[1]	F28	I	PLL 倍频系数输入
pll_pdio2	B32		
pll_iovdd	D30	模拟电源	模拟电源
pll_iovss	C31	模拟地	模拟地
	E31		
	B34		
	G27		

	J29		
	C33		
pll_corevdd	D32	模拟电源	模拟电源
pll_corevss	E33	模拟地	模拟地
pll_pdio1	F32		
vss_io7	D34	io 地	io 地
vdd_io7	H30	io 电源	io 电源
bist_mode	C35	I	测试管脚接低
scan_enable	G31	I	测试管脚接低
pll_min[0]	G33	I	PLL 倍频系数输入
pll_bp	K30	I	PLL 时钟旁路控制信号，高电平有效。 当为低时片内时钟由 PLL 提供。
vdd_core6	F34	核电源	核电源
	L29		
pll_out	E35	O	PLL 测试输出，为片内主时钟。
vss_core6	J31	核地	核地
brdyn	G35	I	外部数据准备好信号，低电平有效。
oen	H32	O	数据总线读使能，低电平有效。
read	J33	O	处理器读/写选择
writen	K32	O	写使能，低电平有效
ramsn[0]	H34	O	RAM 片选，低电平有效。
ramsn[1]	L31	O	RAM 片选，低电平有效。
vdd_io8	J35	io 电源	io 电源
vss_io8	M30	io 地	io 地
ramsn[2]	K34	O	RAM 片选，低电平有效。
ramsn[3]	M32	O	RAM 片选，低电平有效。
ramsn[4]	L33	O	RAM 片选，低电平有效。
	N29		
vdd_core7	L35	核电源	核电源
vss_core7	N31	核地	核地
ramoen[0]	M34	O	RAM 输出使能，低电平有效。
	P30		
ramoen[1]	N33	O	RAM 输出使能，低电平有效。
ramoen[2]	P32	O	RAM 输出使能，低电平有效。
vdd_io9	N35	io 电源	io 电源
	R29		
vss_io9	P34	io 地	io 地
ramoen[3]	R31	O	RAM 输出使能，低电平有效。
ramoen[4]	R33	O	RAM 输出使能，低电平有效。
rwen[0]	T30	O	字节控制的写使能，低电平有效。
rwen[1]	R35	O	字节控制的写使能，低电平有效。
rwen[2]	T32	O	字节控制的写使能，低电平有效。
rwen[3]	T34	O	字节控制的写使能，低电平有效。

	U29		
vss_core8	U33	核地	核地
vdd_core8	U31	核电源	核电源
romsn[0]	U35	O	ROM 片选，低电平有效。
romsn[1]	V30	O	ROM 片选，低电平有效。
Iosn	V34	O	IO 片选，低电平有效。
vdd_io10	V32	io 电源	io 电源
vss_io10	W35	io 地	io 地
sdcsn[0]	W31	O	SDRAM 片选，低电平有效。
sdcsn[1]	W33	O	SDRAM 片选，低电平有效。
	W29		
Sdwen	Y34	O	SDRAM 写使能，低电平有效。
Sdrasn	Y32	O	SDRAM 行地址，低电平有效。
Sdcasn	AA35	O	SDRAM 地址，低电平有效。
vdd_io11	Y30	io 电源	io 电源
vss_io11	AA33	io 地	io 地
	AA31		
sddqm[0]	AB34	O	SDRAM 数据选择，低电平有效。
	AA29		
sddqm[1]	AC35	O	SDRAM 数据选择，低电平有效。
sddqm[2]	AB32	O	SDRAM 数据选择，低电平有效。
sddqm[3]	AC33	O	SDRAM 数据选择，低电平有效。
	AB30		
vss_core9	AD34	核地	核地
Sdclk	AC31	O	SDRAM 时钟输出
vdd_core9	AE35	核电源	核电源
	AC29		
Pio [0]	AE33	I/O	通用 io 与定义 prom 宽度复用
Pio [1]	AD32	I/O	通用 io 与定义 prom 宽度复用
Pio [2]	AF34	I/O	通用 io
Pio [3]	AD30	I/O	通用 io 与串口外部可选时钟复用
Pio [4]	AG35	I/O	通用 io
Pio [5]	AE31	I/O	通用 io
Pio [6]	AH34	I/O	通用 io
vdd_io12	AF32	io 电源	io 电源
vss_io12	AG33	io 地	io 地
Pio [7]	AH32	I/O	通用 io
Pio [8]	AJ35	I/O	通用 io 与 cts2 复用
Pio [9]	AG31	I/O	通用 io 与 rts2 复用
pio [10]	AL35	I/O	通用 io 与 rxd2 复用

	AE29		
vdd_core10	AK34	核电源	核电源
vss_core10	AF30	核地	核地
pio [11]	AJ33	I/O	通用 io 与 txd2 复用
pio [12]	AJ31	I/O	通用 io 与 cts1 复用
pio [13]	AN35	I/O	通用 io 与 rts1 复用
pio [14]	AH30	I/O	通用 io 与 rxd1 复用
Pio[15]	AM34	I/O	通用 io 与 txd1 复用
	AK32		
vdd_io13	AL33	io 电源	io 电源
vss_ io13	AL31	io 地	io 地
	AP34		
	AG29		
	AJ27		
	AN33		
pci_arb_gnt_n[6]	AM32	O	PCI 仲裁器应答
pci_arb_gnt_n[5]	AN31	O	PCI 仲裁器应答
pci_arb_gnt_n[4]	AM30	O	PCI 仲裁器应答
pci_arb_gnt_n[3]	AP32	O	PCI 仲裁器应答
pci_arb_gnt_n[2]	AK28	O	PCI 仲裁器应答
pci_arb_gnt_n[1]	AR33	O	PCI 仲裁器应答
pci_arb_gnt_n[0]	AL29	O	PCI 仲裁器应答
vdd_io14	AN29	io 电源	io 电源
vss_io14	AK26	io 地	io 地
pci_arb_req[6]	AP30	I	PCI 仲裁器申请
	AJ25		
pci_arb_req[5]	AR31	I	PCI 仲裁器申请
pci_arb_req[4]	AL27	I	PCI 仲裁器申请
pci_arb_req[3]	AR29	I	PCI 仲裁器申请
pci_arb_req[2]	AM28	I	PCI 仲裁器申请
pci_arb_req[1]	AN27	I	PCI 仲裁器申请
pci_arb_req[0]	AM26	I	PCI 仲裁器申请
vdd_core11	AP28	核电源	核电源
vss_core11	AL25	核地	核地
pci_ad [0]	AR27	I/O	PCI 数据/地址总线
Pci_ad[1]	AK24	I/O	PCI 数据/地址总线
pci_ad [2]	AP26	I/O	PCI 数据/地址总线
pci_ad [3]	AM24	I/O	PCI 数据/地址总线
pci_ad[4]	AN25	I/O	PCI 数据/地址总线
	AJ23		
vdd_io15	AR25	io 电源	io 电源
vss_io15	AL23	io 地	io 地
pci_ad[5]	AP24	I/O	PCI 数据/地址总线

pci_ad [6]	AK22	I/O	PCI 数据/地址总线
pci_ad [7]	AN23	I/O	PCI 数据/地址总线
pci_cbe[0]	AM22	I/O	PCI 命令和字节使能
pci_ad [8]	AR23	I/O	PCI 数据/地址总线
	AJ21		
vdd_core12	AP22	核电源	核电源
vss_core12	AL21	核地	核地
pci_ad[9]	AN21	I/O	PCI 数据/地址总线
pci_66	AK20	I	PCI 总线 66MHz 使能
pci_ad [10]	AR21	I/O	PCI 数据/地址总线
pci_ad [11]	AM20	I/O	PCI 数据/地址总线
pci_ad[12]	AP20	I/O	PCI 数据/地址总线
	AJ19		
pci_ad[13]	AN19	I/O	PCI 数据/地址总线
pci_ad[14]	AL19	I/O	PCI 数据/地址总线
pci_ad[15]	AR19	I/O	PCI 数据/地址总线
pci_cbe[1]	AK18	I/O	PCI 命令和字节使能
vdd_io16	AP18	io 电源	io 电源
vss_io16	AM18	io 地	io 地
pci_par	AR17	I/O	PCI 数据奇偶校验错误
pci_serr	AL17	I/O	PCI 系统错误指示
pci_perr	AN17	I/O	PCI 校验错误指示
	AJ17		
pci_stop	AP16	I/O	PCI 停止数据传送信号
pci_devsel	AM16	I	配置器件选择信号
pci_trdy	AR15	I/O	PCI 目标设备准备好
pci_idry	AK16	I/O	初始化准备
pci_frame	AN15	I/O	帧指示信号
vdd_core13	AL15	核电源	核电源
vss_core13	AP14	核地	核地
	AJ15		
pci_cbe[2]	AR13	I/O	PCI 命令和字节使能
pci_ad [16]	AM14	I/O	PCI 数据/地址总线
pci_ad [17]	AN13	I/O	PCI 数据/地址总线
pci_ad [18]	AK14	I/O	PCI 数据/地址总线
pci_ad [19]	AP12	I/O	PCI 数据/地址总线
pci_ad[20]	AL13	I/O	PCI 数据/地址总线
Pci_ad[21]	AR11	I/O	PCI 数据/地址总线
	AJ13		
vdd_io17	AN11	io 电源	io 电源
vss_io17	AM12	io 地	io 地
pci_ad[22]	AP10	I/O	PCI 数据/地址总线
pci_ad[23]	AK12	I/O	PCI 数据/地址总线

pci_idsel	AR9	I	配置器件选择信号
Pci_cbe[3]	AL11	I/O	PCI 命令和字节使能
pci_ad[24]	AP8	I/O	PCI 数据/地址总线
Vdd_core14	AM10	核电源	核电源
Vss_core14	AN9	核地	核地
pci_ad [25]	AM8	I/O	PCI 数据/地址总线
pci_ad [26]	AR7	I/O	PCI 数据/地址总线
pci_ad [27]	AL9	I/O	PCI 数据/地址总线
pci_ad [28]	AR5	I/O	PCI 数据/地址总线
	AJ11		
Pci_ad[29]	AP6	I/O	PCI 数据/地址总线
Pci_ad[30]	AK10	I/O	PCI 数据/地址总线
vdd_io18	AN7	io 电源	io 电源
vss_io18	AL7	io 地	io 地
Pci_ad[31]	AR3	I/O	PCI 数据/地址总线
Pci_req	AK8	O	总线请求
pci_gnt	AP4	I	总线给予
	AM6		
	AN5		
	AL5		
	AP2		
	AJ9		
	AG7		
vss_io19	AN3	io 地	io 地
pci_clk	AM4	I	PCI 时钟输入
vdd_io19	AL3	io 电源	io 电源
pci_rst	AK4	I/O	PCI 复位信号，低电平有效
pci_intd_n	AM2	I	PCI 总线中断 D
pci_intc_n	AH6	I	PCI 总线中断 C
vss_core15	AN1	核地	核地
vdd_ core15	AJ5	核电源	核电源
pci_intb_n	AJ3	I	PCI 总线中断 B
	AF6		
pci_inta_n	AK2	I	PCI 总线中断 A
	AE7		
testout1	AL1	I/O	测试输出，1=高阻状态
vss_core16	AG5	核地	核地
vdd_core16	AJ1	核电源	核电源
testin1	AH4	I	测试管脚使用时接低
rombsd_pad_3	AG3	I	Rom_bank 大小定义
rombsd_pad_2	AF4	I	Rom_bank 大小定义
rombsd_pad_1	AH2	I	Rom_bank 大小定义
rombsd_pad_0	AE5	I	Rom_bank 大小定义

pad_skew_0	AG1	I	时钟树延时输入
pad_skew_1	AD6	I	时钟树延时输入
vss_core17	AF2	核地	核地
vdd_core17	AD4	核电源	核电源
pci_host	AE3	I	Pci host/guest 模式切换 低电平有效
	AC7		
sim_fast_reset	AE1	I	测试管脚接低
testin0	AC5	I	测试管脚使用时接低
testout0	AD2	O	测试输出，1=浮空
wdogn	AB6	OD	看门狗输出，低电平有效。
vss_io20	AC3	io 地	io 地
vdd_io20	AB4	io 电源	io 电源
txd3	AC1	O	串口 3 数据输出
	AA7		
rxd3	AB2	I	串口 3 数据输入
rts3	AA5	O	串口 3 请求发送
cts3	AA3	I	串口 3 清除发送
vss_core18	Y6	核地	核地
vdd_core18	AA1	核电源	核电源
timer0	Y4	O	定时器 1 输出
timer1	Y2	O	定时器 2 输出
	W7		
Dsuen	W3	I	调试使能信号
Dsutx	W5	O	调试数据输出
vss_io21	W1	io 地	io 地
vdd_io21	V6	io 电源	io 电源
Dsurx	V2	I	调试数据输入
Dsubre	V4	I	硬件调试中断输入，高电平有效。
Dsuact	U1	O	调试活动指示，高电平有效。
Vdd_io22	U5	io 电源	io 电源
Vss_io22	U3	io 地	io 地
	U7		
Tclk	T2	I	JTAG 时钟
Tdi	T4	I	测试数据输入
Tdo	R1	Tri state O	测试数据输出
	T6		
vss_core19	R3	核地	核地
vdd_core19	R5	核电源	核电源
Tms	P2	I	测试模式选择信号
	R7		
Trst	N1	I	测试复位信号
Resetn	P4	I	系统复位信号，低电平有效，要求

			保持大于两个时钟周期。
Bexcn	N3	I	总线异常指示低电平有效。
Errorn	P6	OD	处理器错误指示，低电平有效。
address[0]	M2	O	处理器地址总线
address[1]	N5	O	处理器地址总线
vss_io23	L1	io 地	io 地
	N7		
vdd_io23	L3	io 电源	io 电源
address[2]	M4	O	处理器地址总线
address[3]	K2	O	处理器地址总线
address[4]	M6	O	处理器地址总线
address[5]	J1	O	处理器地址总线
address[6]	L5	O	处理器地址总线
Vss_core20	H2	核地	核地
Vdd_core20	K4	核电源	核电源
address[7]	J3	O	处理器地址总线
	H4		
address[8]	G1	O	处理器地址总线
address[9]	J5	O	处理器地址总线
scan_mode	E1	I	scan 模式的选择（高电平有效）
	L7		
address[10]	F2	O	处理器地址总线
address[11]	K6	O	处理器地址总线
address[12]	G3	O	处理器地址总线
vss_io24	G5	io 地	io 地
vdd_io24	C1	io 电源	io 电源
	H6		
address[13]	D2	O	处理器地址总线
address[14]	F4	O	处理器地址总线
address[15]	E3	O	处理器地址总线
	E5		
address[16]	B2	O	处理器地址总线
	J7		

注：

pll_corevdd和pll_corevss: 模拟电源和地pll_corevdd = 1.8±10%

pll_iovdd和pll_iovss : 模拟电源和地pll_iovdd = 3.3±10%

vdd_io和vss_io : I/O电源和地 vdd_io =3.3±10%

vdd_core和vss_core : 核电源和地 vdd_core =1.8±10%

表 D- 2 输出驱动列表

I_{OH}	-0.5mA	-4mA	-8mA	-12mA	-16mA	无 (OD 双向)	无 (OD 输出)	-4mA (三态输出)
I_{OL}	1.5 mA	4mA	8mA	12mA	16mA	1.5mA (OD 双向)	4mA (OD 输出)	4mA (三态输出)
管脚名	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 dsutx dsuact	Sdclk	pci_serr_n pci_inta_n	errorn wdogn	Tdo

E 电气特性

静态参数

表 E-1 电参数

参数	符号	条件 (除另有规定外， -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)		分组	极限值		单位
					最小	最大	
输出高电平电压	V_{OH}	V_{DDD}=1.65V，V_{DDA}=1.65V，V_{DDDIO}=3.0V，V_{DDAIO}=3.0V，测所有适用的输出端。[d]	I_{OH}=-0.5 mA	A1 A2 A3	2.7	—	V
			I_{OH}= -4、-8、-12、-16 mA		2.6	—	
输出低电平电压	V_{OL}		I_{OL}=1.5 mA		—	0.3	V
			I_{OL}= 4、8、12、16 mA		—	0.4	
输入高电平漏电流	I_{IH}	V_{DDD}= 1.95V，V_{DDA}= 1.95V，V_{DDDIO}= 3.6V，V_{DDAIO}= 3.6V，V_I= 3.6V			—	1	μA
上拉输入高电平漏电流	I_{IHPU}				—	5	μA
下拉输入高电平漏电流	I_{IHPD}				40	200	μA
输入低电平漏电流	$\lvert I_{IL} \rvert$	V_{DDD}=1.95V，V_{DDA}=1.95V，V_{DDDIO}=3.6V，V_{DDAIO}= 3.6V，V_I= 0V			—	1	μA
上拉输入低电平漏电流	$\lvert I_{ILPU} \rvert$				40	200	μA
下拉输入低电平漏电流	$\lvert I_{ILPD} \rvert$				—	5	μA
三态输出高电平漏电流	I_{OZH}	V_{DDD}=1.95V，V_{DDA}=1.95V，V_{DDDIO}=3.6V，V_{DDAIO}= 3.6V，V_O=3.6V			—	1	μA
三态输出低电平漏电流	$\lvert I_{OZL} \rvert$	V_{DDD}=1.95V，V_{DDA}=1.95V，V_{DDDIO}=3.6V，V_{DDAIO}= 3.6V，V_O=0V			—	1	μA
待机电源电流	$I_{DDD(SB)}$	V_{DDD}=1.95V V_{DDA}=1.95V V_{DDDIO}=3.6V V_{DDAIO}=3.6V clk 和 pci_clk 时钟信号不工作。	测 V_{DDD} 端		—	10	mA
			测 V_{DDDIO} 端		—	5	
输入电容[a]	C_{IN}	T_A =25℃		A4	—	12	pF
输出电容[a]	C_{IO}	T_A =25℃			—	12	pF

功能测试	V_{DDD}=1.65V、1.8V、1.95V V_{DDA}=1.65V、1.8V、1.95V V_{DDDIO}=3.0V、3.3V、3.6V V_{DDAIO}=3.0V、3.3V、3.6V V_{IH} =V_{DDDIO} V_{IL} = 0V V_{OUT}= V_{DDDIO}/2 [c] f =10MHz 和 100MHz 条件下功能测试。	A7 A8A A8B	

动态特性

表 E-2 动态特性

参数	符号	条件 (除另有规定外， 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)	分组	极限值		单位
				最小	最大	
pll 旁路情况 clk 周期 [e]	t_1	V_{DDD}＝1.65V， V_{DDA}＝1.65V， V_{DDDIO}＝3.0V， V_{DDAIO}＝3.0V， f=10MHz	A9 A10 A11	10	—	ns
Pll 允许下的 clk 周期 [b]	t_{1_p}			40	50	ns
Pll 旁路情况 clk 高低脉宽 [b]	t_2			4.5	—	ns
Pll 允许下的 clk 高低脉宽 [b]	t_{2_p}			18	—	ns
SDCLK 周期 [e]	t_3			10	—	ns
Pll 旁路情况 Sdclk 输出延时	t_4			2	7	ns
PLL 建立时间	t_5			—	10^7	ns
复位脉宽 [e]	t_6			20	—	ns
address[27:0]输出延时	t_{10}			1.5	7	ns
data[31:0] and pardata[6:0] 写输出延时	t_{11}			2	9	ns
data[31:0] and pardata[6:0] 读建立时间 [b]	t_{12}			4.5	—	ns
data[31:0] and pardata[6:0] 读保持时间 [b]	t_{13}			0	—	ns
data[31:0] and pardata[6:0] 写输出保持时间	t_{14}			2	9	ns
Oen 输出延时	t_{15}			2	7	ns
writen 输出延时	t_{17}			1.5	7	ns
Romsn[1:0]输出延时	t_{18}			2	9.5	ns
Ramsn[4:0]输出延时	t_{19}			2	9.5	ns
Ramoen[4:0]输出延时	t_{20}			2	9.5	ns
rwen[3:0] 输出延时	t_{21}			2	9.5	ns
iosn 输出延时	t_{22}			2	9.5	ns
Brdyn 建立时间 [b]	t_{23}			4	—	ns
Brdyn 保持时间 [b]	t_{24}			0	—	ns

sdcasn 输出延时	t_{25}			3	9	ns
sdcsn[1:0] 输出延时	t_{26}			2	8.5	ns
sdrasn 输出延时	t_{27}			2	8.5	ns
sdwen 输出延时	t_{28}			2	8.5	ns
sddqm[3:0] 输出延时	t_{29}			2	8.5	ns
pio[15:0]输出延时	t_{31}			2.5	10	ns
pio[15:0]建立时间 [b]	t_{32}			4.5	—	ns
pio[15:0]读保持时间 [b]	t_{33}			0	—	ns
pio[15:0]写保持时间	t_{34}			2.5	—	ns
Bexcn 建立时间 [b]	t_{35}			4	—	ns
Bexcn 保持时间 [b]	t_{36}			0	—	ns
Pci_clk 周期 [e]	t_{101}			30	—	ns
PCI_CLK 低脉宽和高脉宽 [b]	t_{102}			14.5	—	ns
Pci_ad_n[31:0] 和 pci_cbe_n[3:0] 输出延时	t_{110}			4	12	ns
Pci_ad_n[31:0] 和 pci_cbe_n[3:0] 建立时间 [b]	t_{111}			6	—	ns
Pci_ad_n[31:0] 和 pci_cbe_n[3:0] 保持时间 [b]	t_{112}			0	—	ns
Pci_frame_n, pci_par_n, pci_perr_n, pci_serr_n, pci_stop_n, 和 pci_devsel_n 输出延时	t_{113}			4	11	ns
pci_irdy_n 和 pci_trdy_n 输出延时	t_{114}			4	11	ns
pci_req_n 输出延时	t_{115}			4	12	ns
Pci_frame_n, pci_par_n, pci_perr_n, pci_serr_n, pci_stop_n, pci_idsel_n,和 pci_devsel_n 建立时间 [b]	t_{116}			7	—	ns
pci_irdy_n 和 pci_trdy_n 建立时间 [b]	t_{117}			7	—	ns
pci_gnt_in_n 建立时间 [b]	t_{118}			6	—	ns
Pci_frame_n, pci_par_n, pci_perr_n, pci_serr_n, pci_stop_n, pci_idsel_n, 和 pci_devsel_n 保持时间 [b]	t_{119}			0	—	ns
pci_irdy_n 和 pci_trdy_n 保持时间 [b]	t_{120}			0	—	ns
pci_gnt_in_n 保持时间 [b]	t_{121}			0	—	ns
功能测试	*FT*	V_{DDD}=1.65V、1.8V、1.95V V_{DDA}=1.65V、1.8V、1.95V V_{DDDIO}=3.0V、3.3V、3.6V V_{DDAIO}=3.0V、3.3V、3.6V V_{IH} =V_{DDDIO} V_{IL} = 0V V_{OUT}= V_{DDDIO}/2 [C] *f* =10MHz 和 100MHz 条件下功能测试。	A7 A8A A8B	—	—	—

典型特性曲线

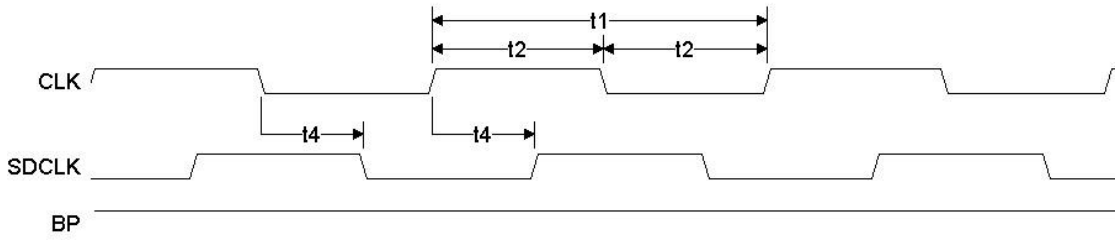


图 E.1　PLL 旁路情况下的时钟信号

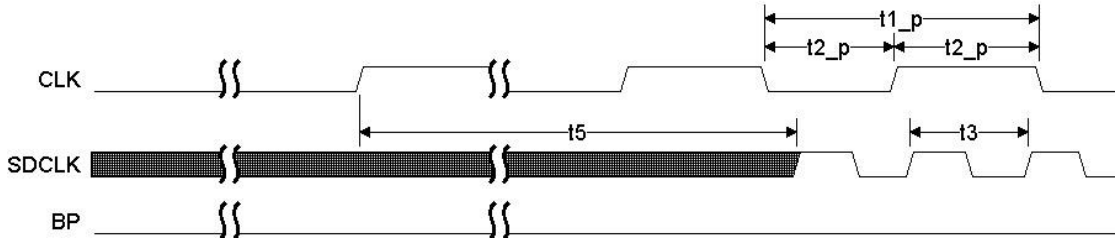


图 E.2　使用 PLL 情况下的时钟信号

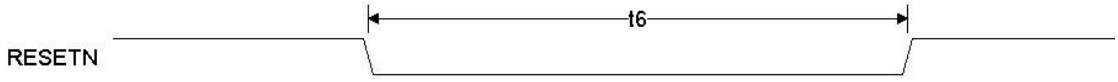


图 E.3 复位信号

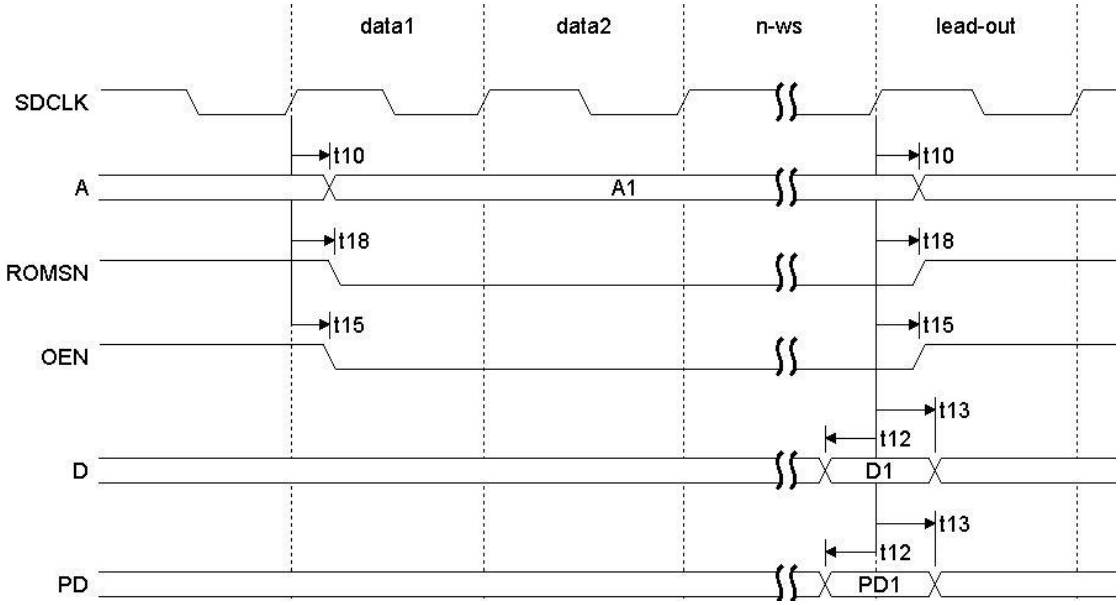


图 E.4　32 位 prom 读时序

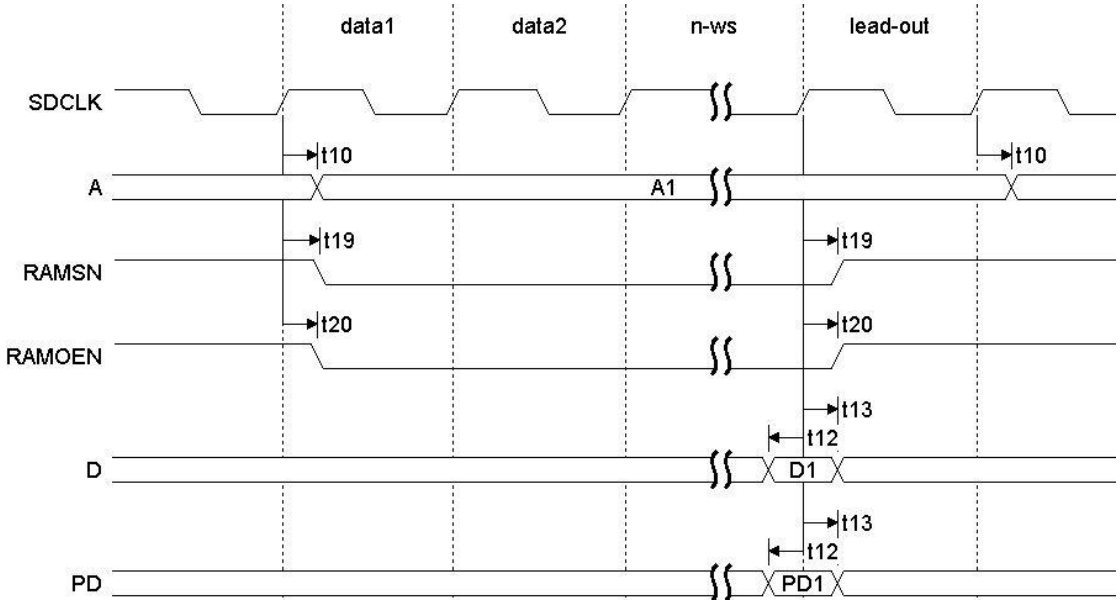


图 E.5　32 位 sram 读时序

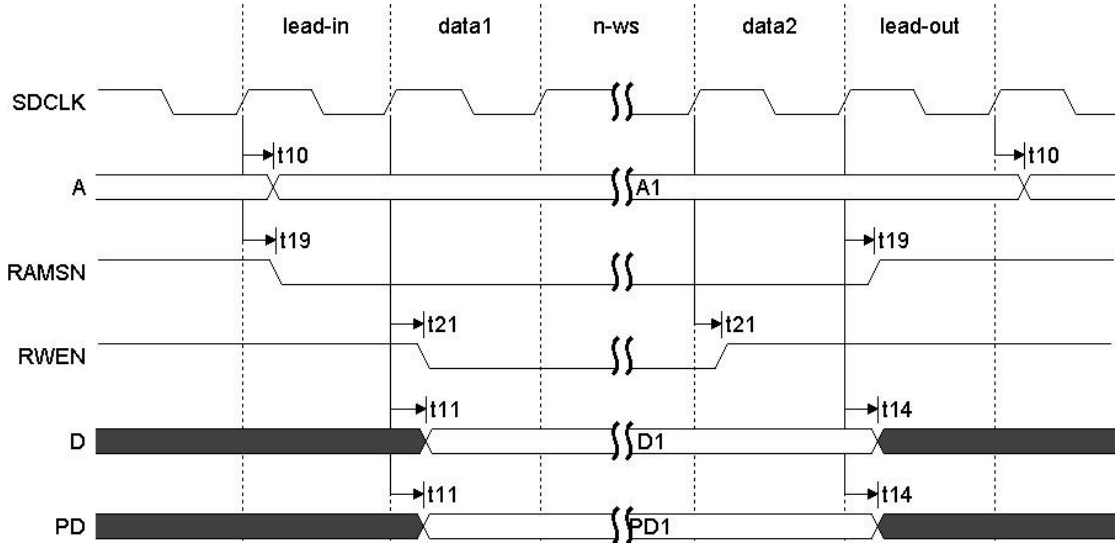


图 E.6　32 位 sram 写时序

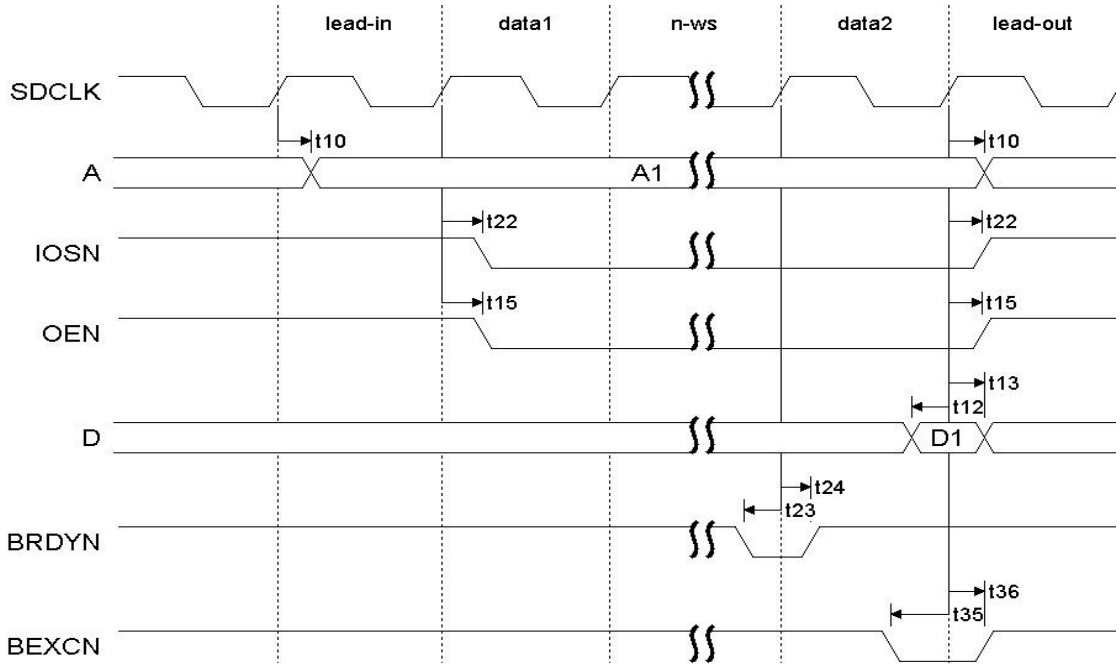


图 E.7　32 位 i/o 读时序

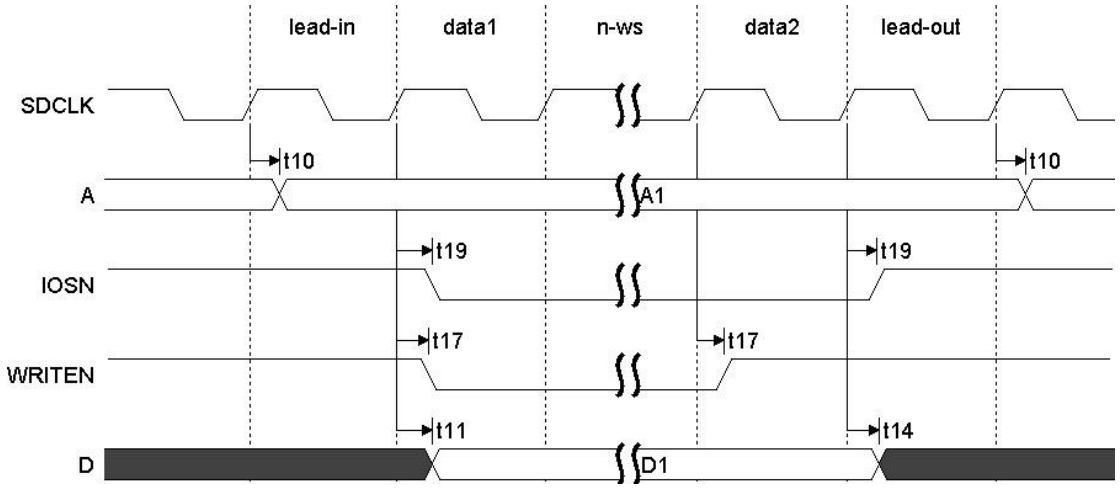


图 E.8　32 位 i/o 写时序

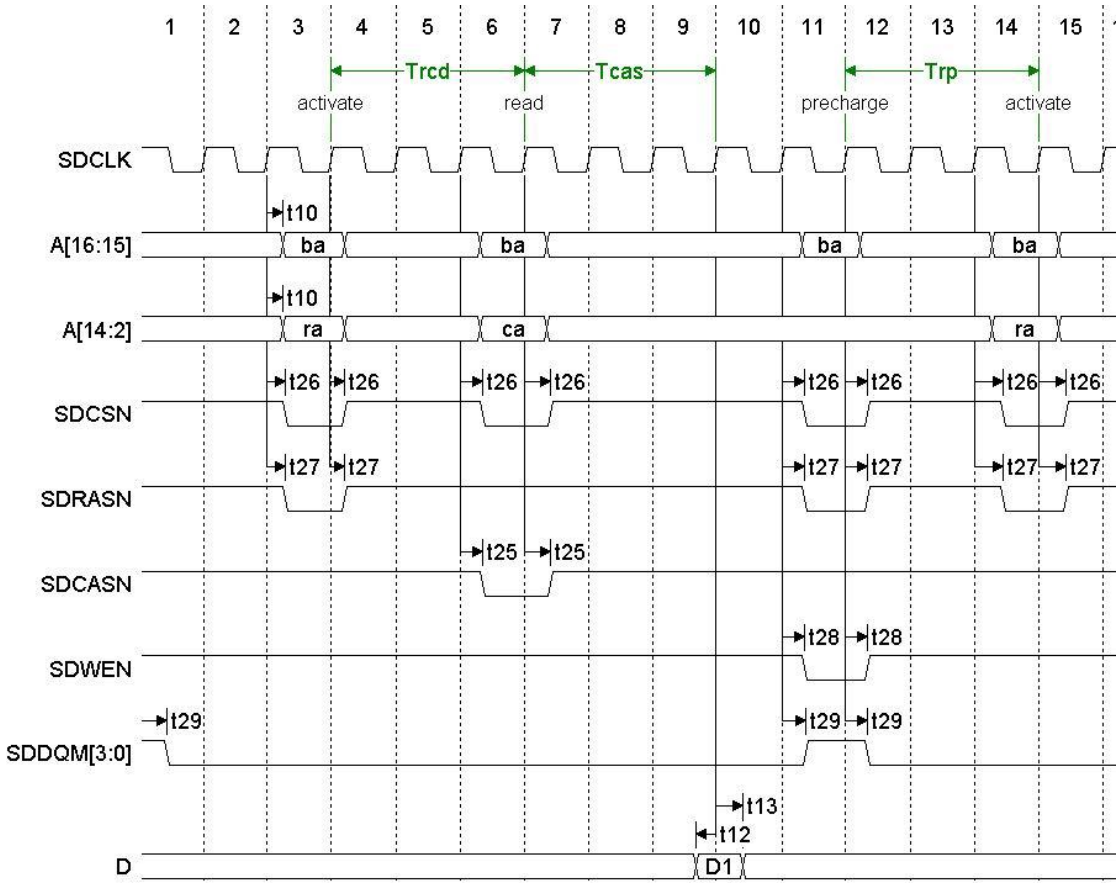


图 E.9　Sdram 读时序

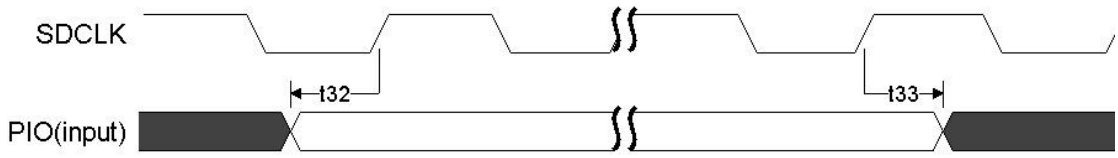


图 E.10　PIO 输入时序

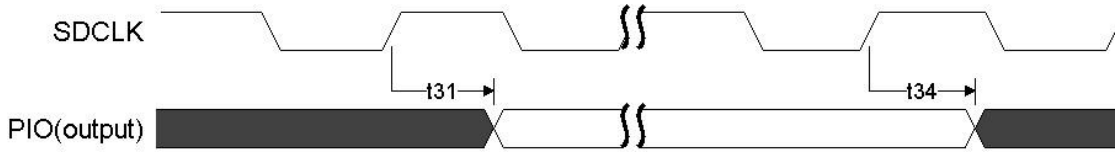


图 E.11　PIO 输出时序

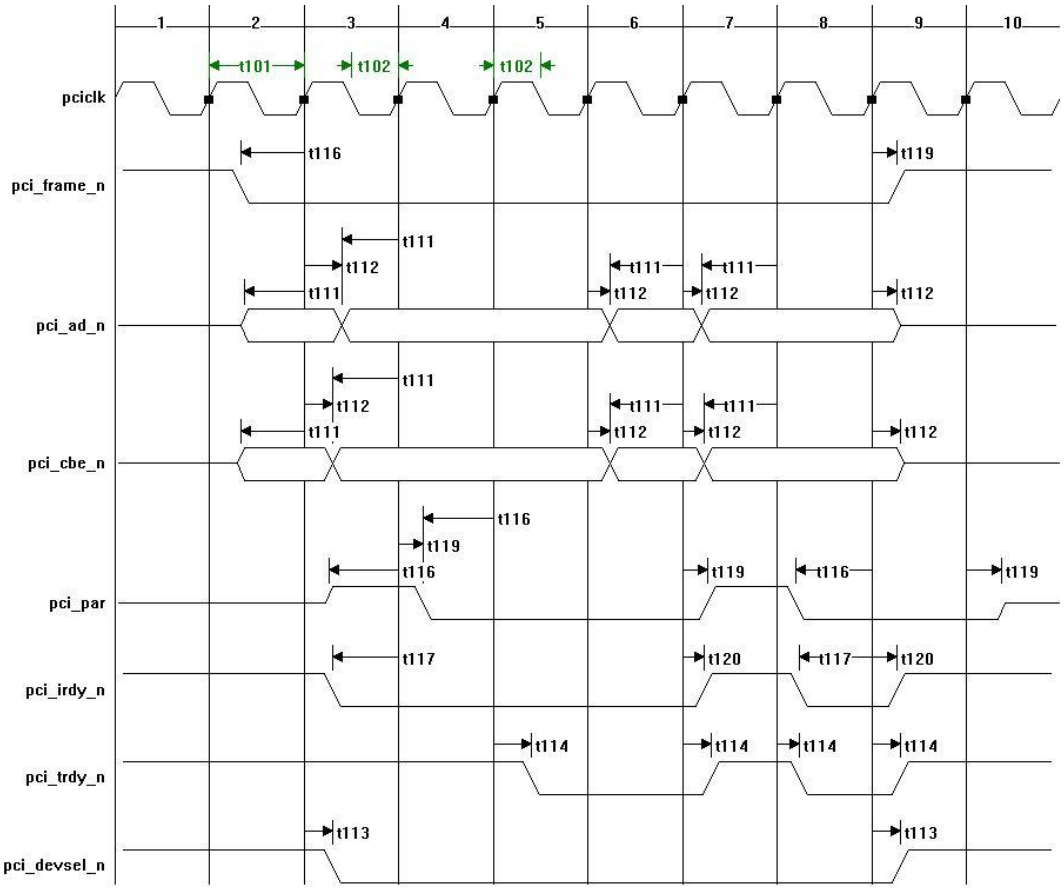

1
2
3
4
5
6
7
8
9
10
t101
t102
t102
pciclk
t116
t119
pci_frame_n
t111
t112
t111
t111
t111
t112
t111
t112
t112
pci_ad_n
t111
t112
t111
t111
t111
t112
t111
t112
t112
pci_cbe_n
t116
t119
t116
t119
t116
t119
pci_par
t117
t120
t117
t120
pci_irdy_n
t114
t114
t114
t114
pci_trdy_n
t113
t113
pci_devsel_n

图 E.12　Pci 写时序

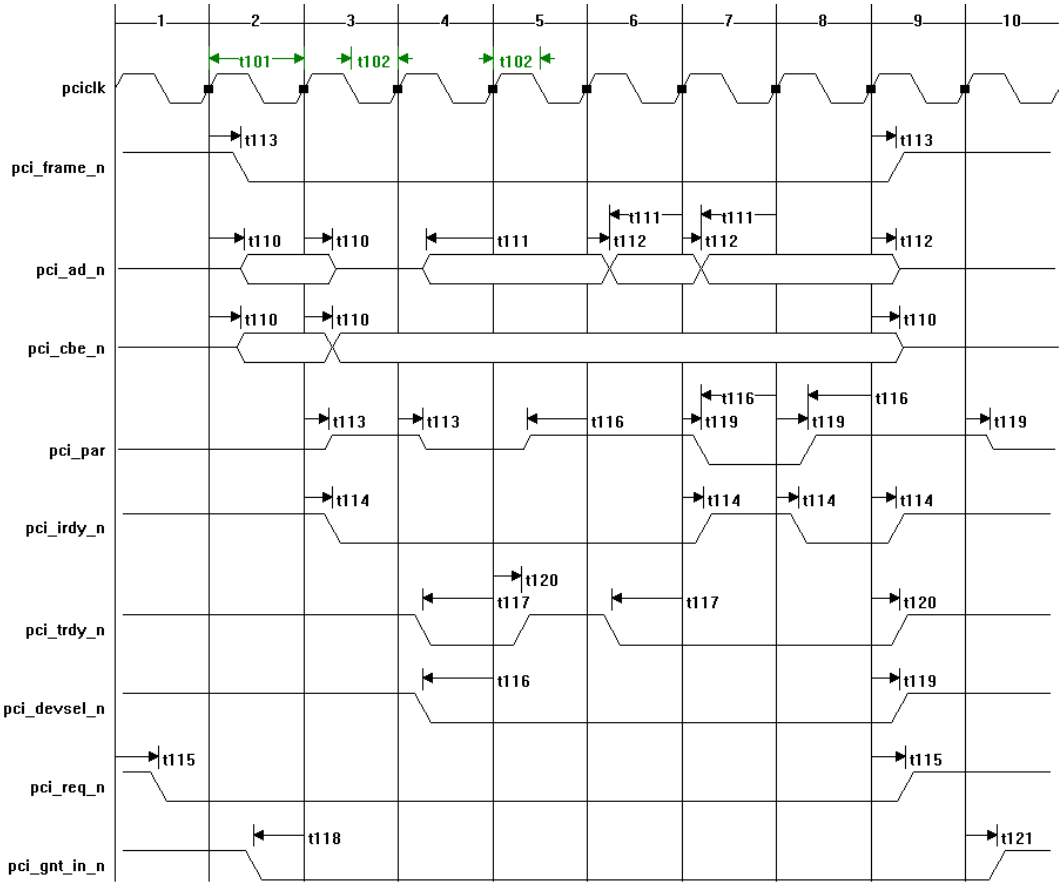

1
2
3
4
5
6
7
8
9
10
t101
t102
t102
pciclk
t113
t113
pci_frame_n
t111
t111
t110
t110
t111
t112
t112
pci_ad_n
t110
t110
t110
pci_cbe_n
t116
t116
t113
t113
t116
t119
t119
t119
pci_par
t114
t114
t114
t114
pci_irdy_n
t120
t117
t117
t120
pci_trdy_n
t116
t119
pci_devsel_n
t115
t115
pci_req_n
t118
t121
pci_gnt_in_n

图 E.13　Pci 读时序

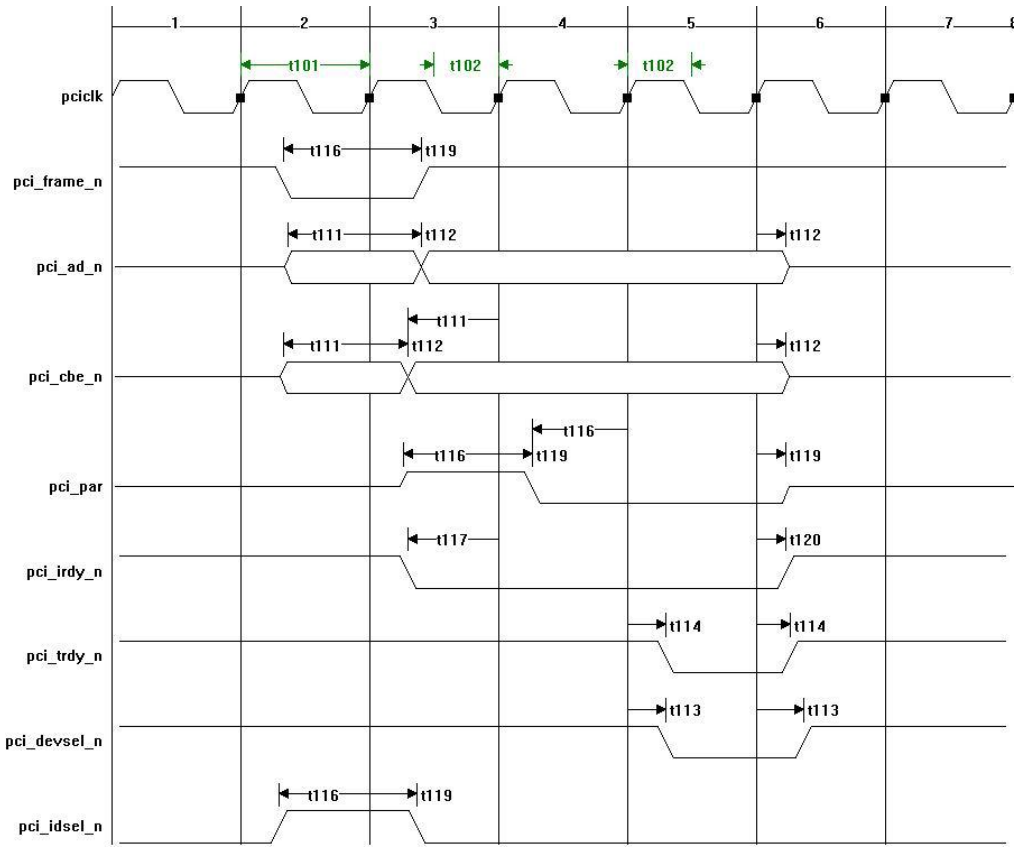


图 E.14　Pci 配置写时序

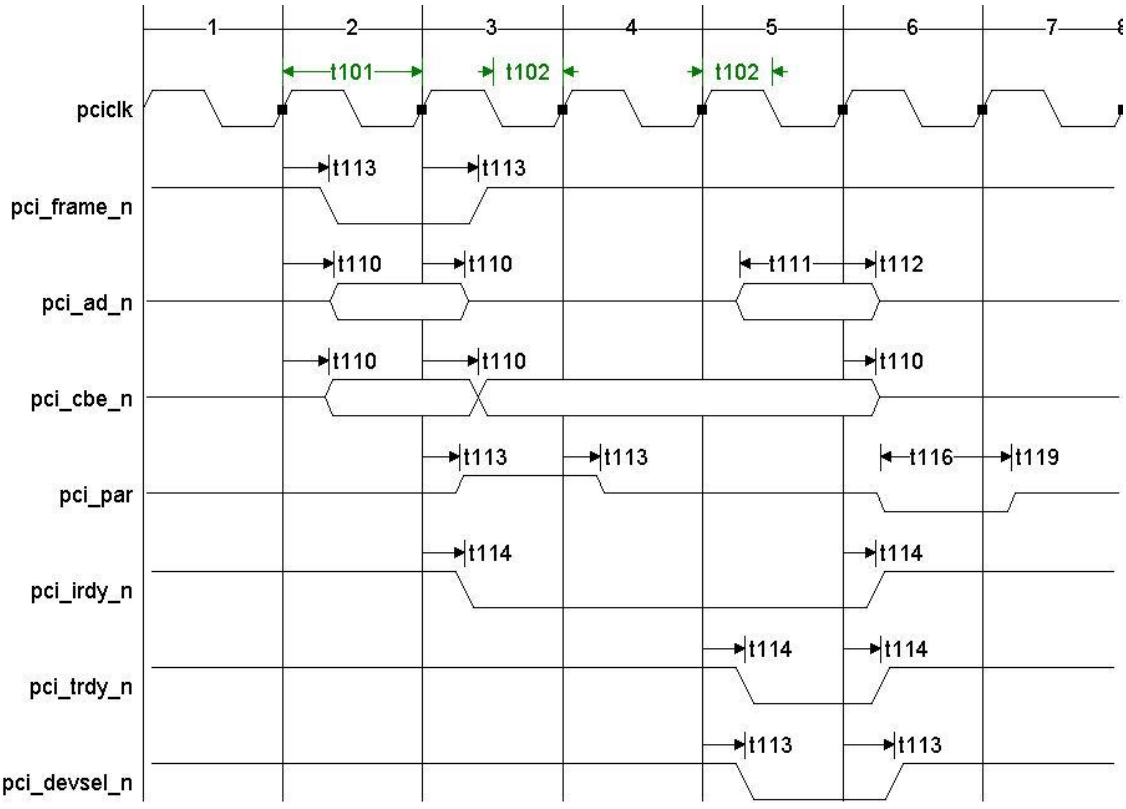


图 E.15　Pci 配置读时序

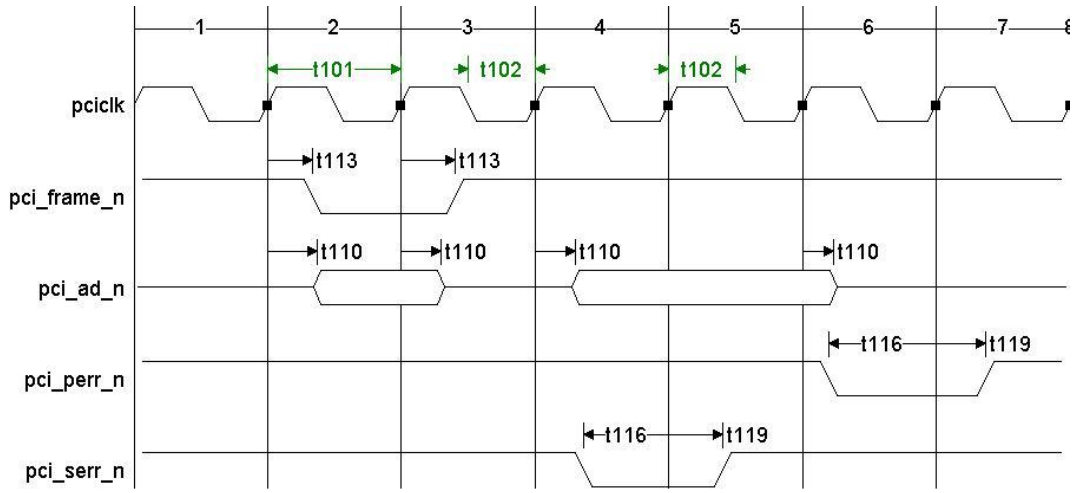


图 E.16　Pci_perr_n 和 pci_serr_n 输入时序

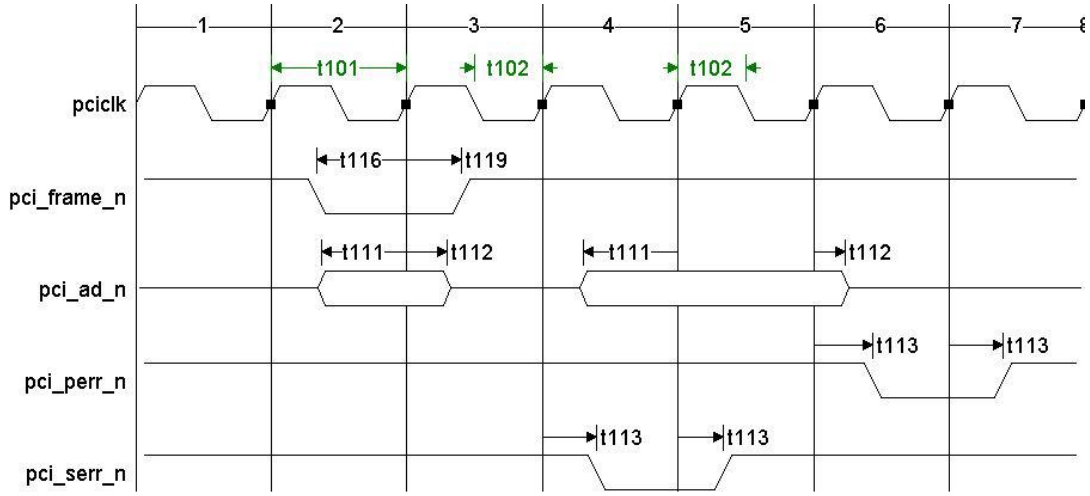


图 E.17　Pci_perr_n 和 pci_serr_n 输出时序

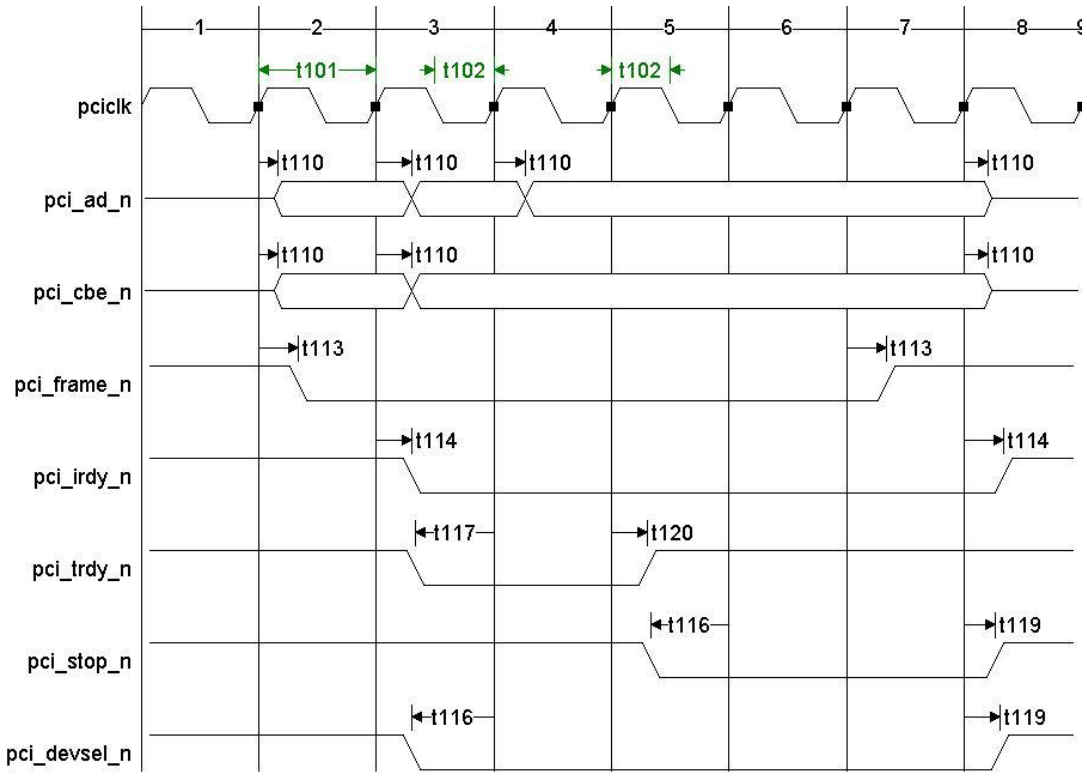


图 E.18　Pci_stop_n 输入时序

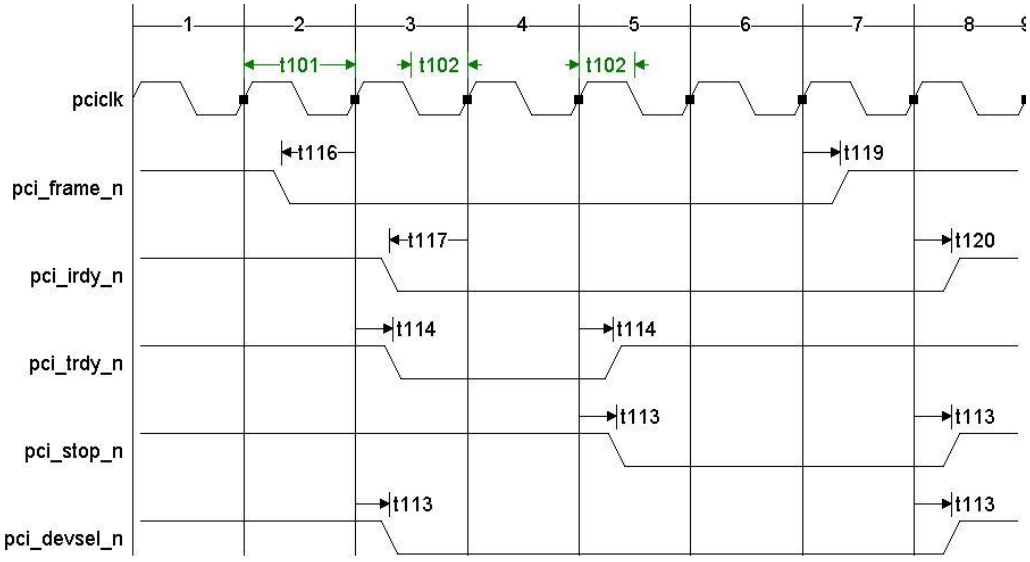


图 E.19　Pci_stop_n 输出时序

图 E　交流参数波形图

F 封装特性

芯片的信号管脚确定为 229 个，加上电源和地共有 317 个可连接管脚。封装采用 PGA391 封装。

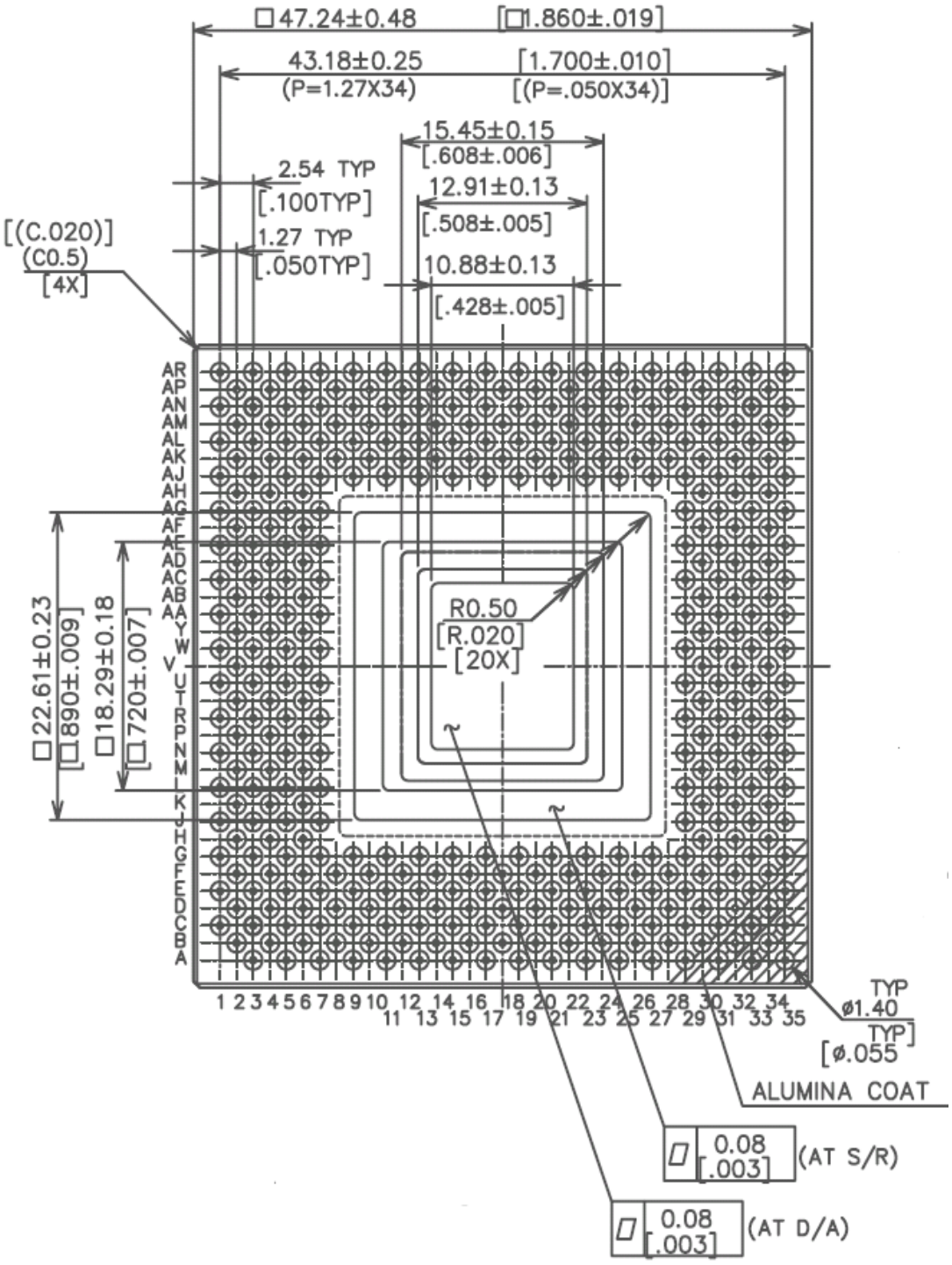


图 F-1 封装外壳底视图

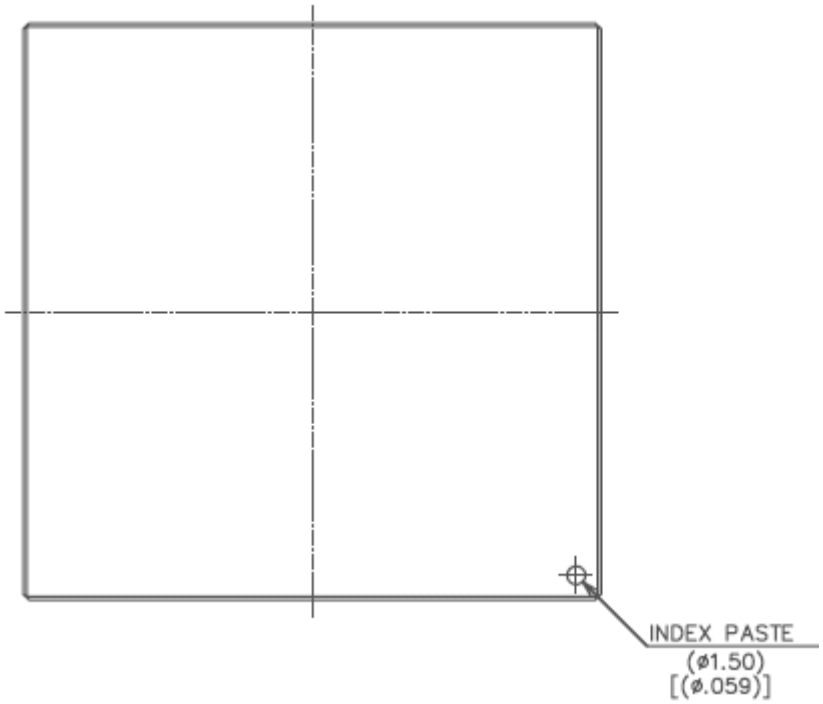


图 F-2 封装外壳俯视图

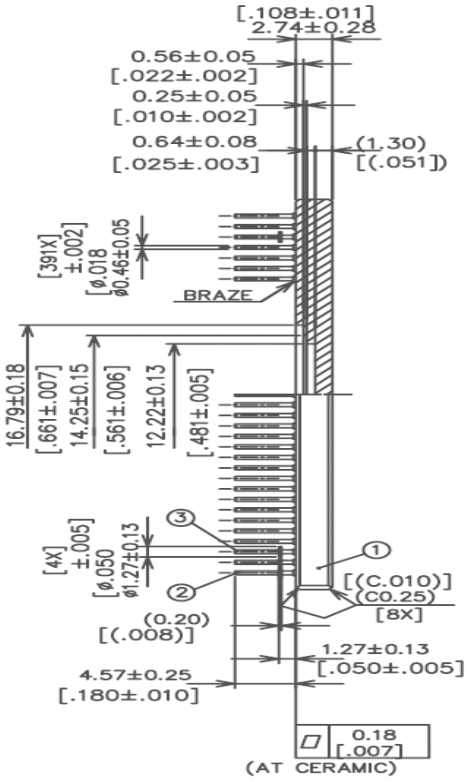


图 F-3 封装外壳侧视图

G 硬件开发规程

BM3803 硬件开发应主要包括如下功能：

1. 外部提供电压输入;
2. BM3803处理器,设计主频100MHz；
3. 板载FLASH；
4. 板载SRAM;
5. 板载SDRAM;
6. 至少提供1个通用异步串行接口,1个DSU调试端口；
7. 提供符合PCIMG 2.0 R3.0规范的CPCI接口；
8. 提供IO总线扩展接口、GPIO扩展接口；

外围硬件结构框图如下：

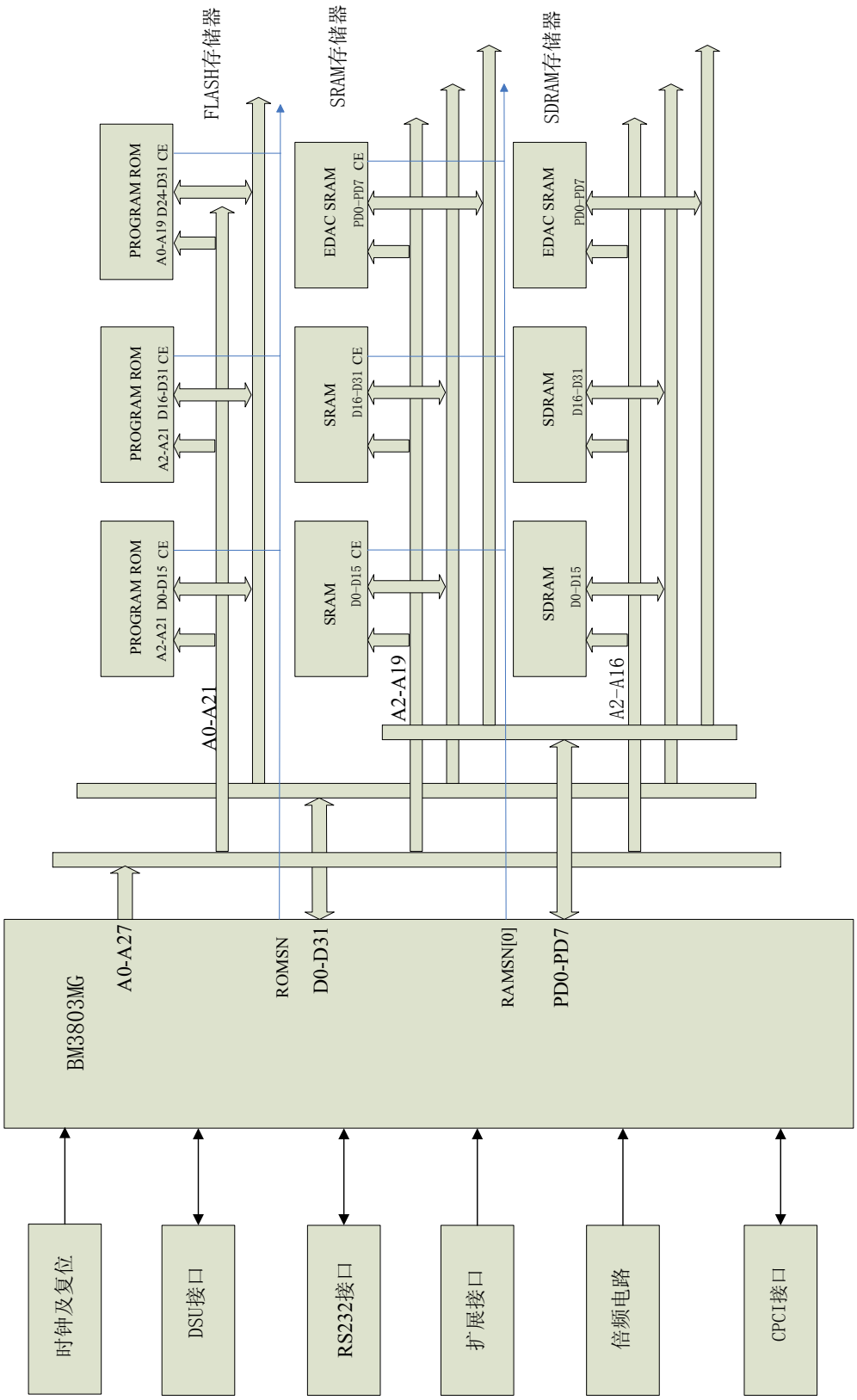


图 G-1 BM3803 结构框图

分别描述如下：

1、3803 硬件使用了 3.3V 和 1.8V 两种电源，提供给 CPU 和存储器。

2、推荐采用了25MHz晶振作为系统时钟， 由于BM3803处理器内部集成了倍频电路,处理器的主频由外频和倍频系M共同决定(其中M 的范围在1－15之间,使用拨码开关可以进行倍频选择),但该电路是否进行时钟倍频操作，完全取决于时钟倍频电路的工作模式,具体模式如下：

- 在BYPASS=1 工作状态下（即PLLBP=H的条件下）,倍频系数M无效：

  $f_{cpuclk} = f_{clk}$ =25MHz

- 在BYPASS=0 工作状态下（即PLLBP=L的条件下）,倍频系数M有效:

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

其中，拨码开关关用于倍频系数 M 的选择,列表如下：

倍频系数 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

注：“1”代表连通 H，“0”代表连通 L；倍频系数 M 推荐为 4，（额定频率 100MHz，不建议超频使用）

3、DSU 及其复位电路部分

此部分电路由拨码开关、RESET 复位开关和ＤＳＵ串口共同组成。

这些开关的组态决定了 CPU 的两种工作状态， 分别是：

- 运行模式

将拨码开关的 DSUEN 和 DSUBRE 都置于低电平后，按下 RESET 按钮，此时处理器运行 FLASH 中的程序，即进入程序运行模式 ；

- 调试模式

将拨码开关的 DSUEN 置于高电平、DSUBRE 置于低电平后，如下图位置所示，按下 RESET 按钮，此时 DSUACT 指示灯变亮，处理器进入调试模式；进入调试模式后可通过ＤＳＵ接口连接芯片进行调试。

4、程序存储器

PROM的接口分为两个Bank，每个Bank最大空间为256M。支持通用Flash接口芯片，如：AM29LV160D。

5、SRAM

SRAM 的接口分为五个 Bank，每个 Bank 最大空间为 256M。支持通用 SRAM 接口芯片，如：CY7C1041。

6、SDRAM

SDRAM 的接口分为两个 Bank，每个 Bank 最大空间为 256M。支持通用 SDRAM 接口芯片，如：HY57V561620。

7、CPCI 接口

BM3803的PCI具有HOST模式和GUEST模式，开发板提供符合PCIMG 2.0 R3.0规范的CPCI接口， 此接口可以根据用户要求通过设置跳线或配置电阻分别实现这两种模式。

8、IO 总线

I/O 空间具有 256M 字节，是单独的一块，I/O 访问控制应用了标准的管脚配置，包括片选、输出使能、读、写块。I/O 块大小是不可编程的，全部 I/O 空间控制信号为：iosn，oen，read，write。

9、GPIO 输出

通用接口GPIO，为32位宽的双向I/O端口，分为高16位和低16位，其中高16位与

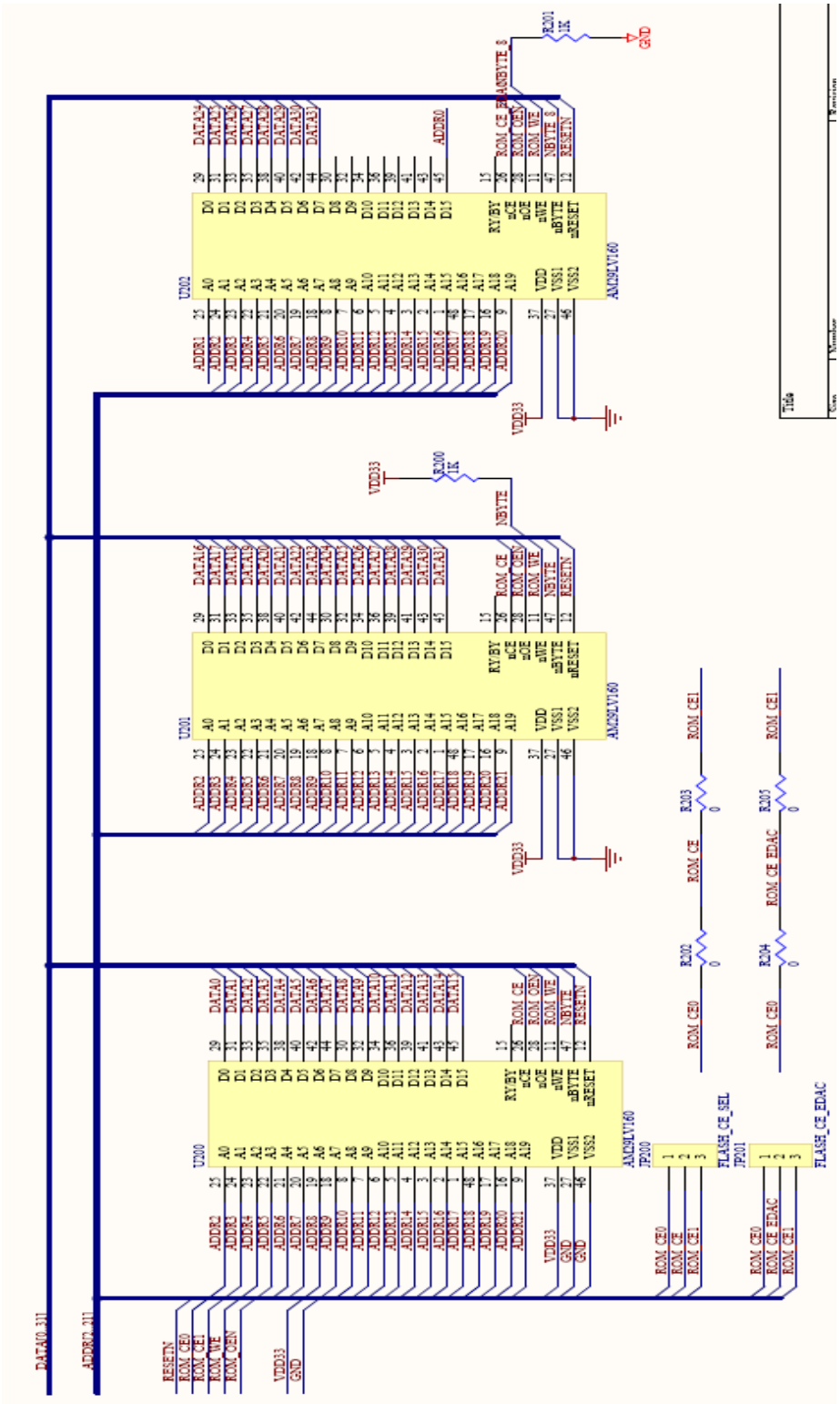
数据总线复用，低16位由并行I/O端口访问

10、UART 输出

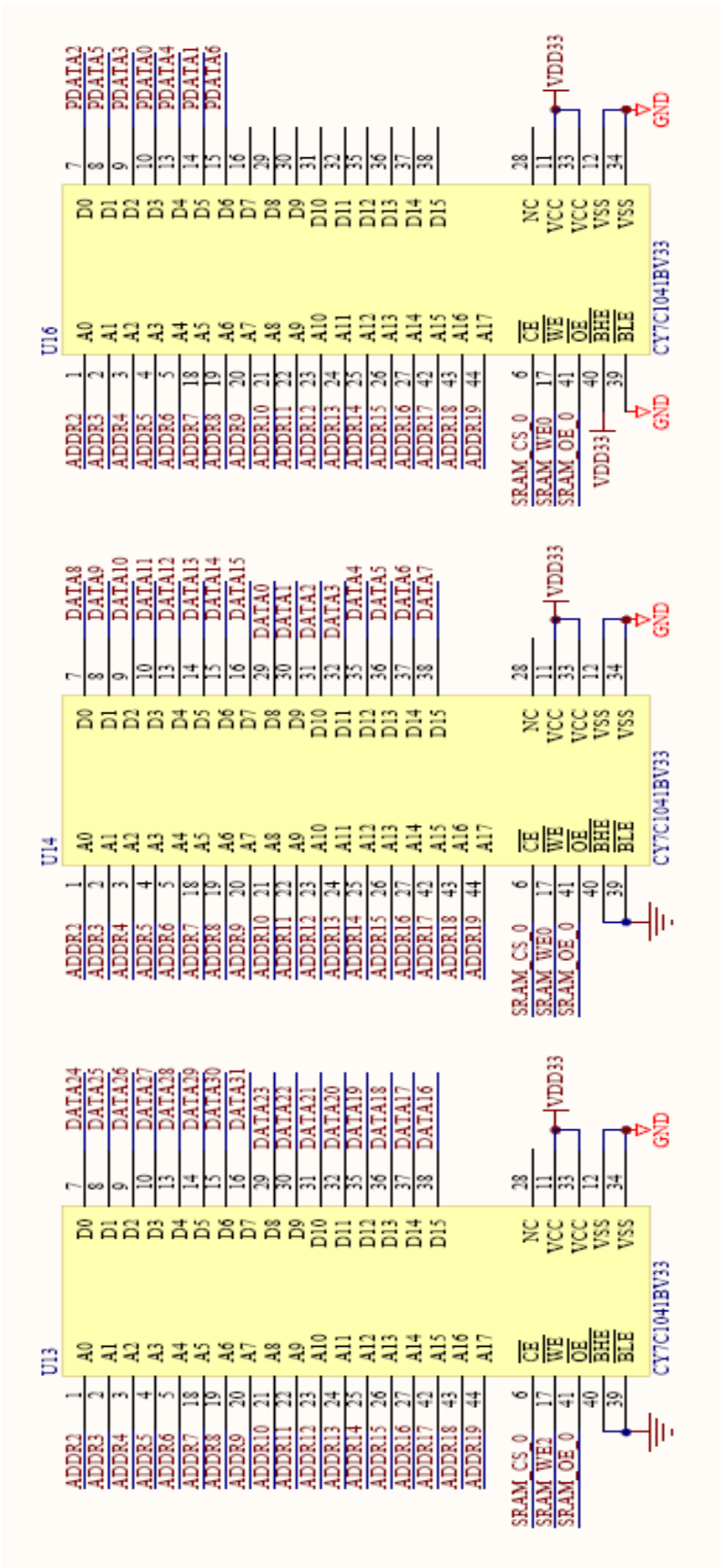
BM3803MG 实现了三个通用异步收发器。其中两个的管脚是与 GPIO 复用。建议用户使用时至少引出一个串口以利调试。

存储器扩展参考电路原理图

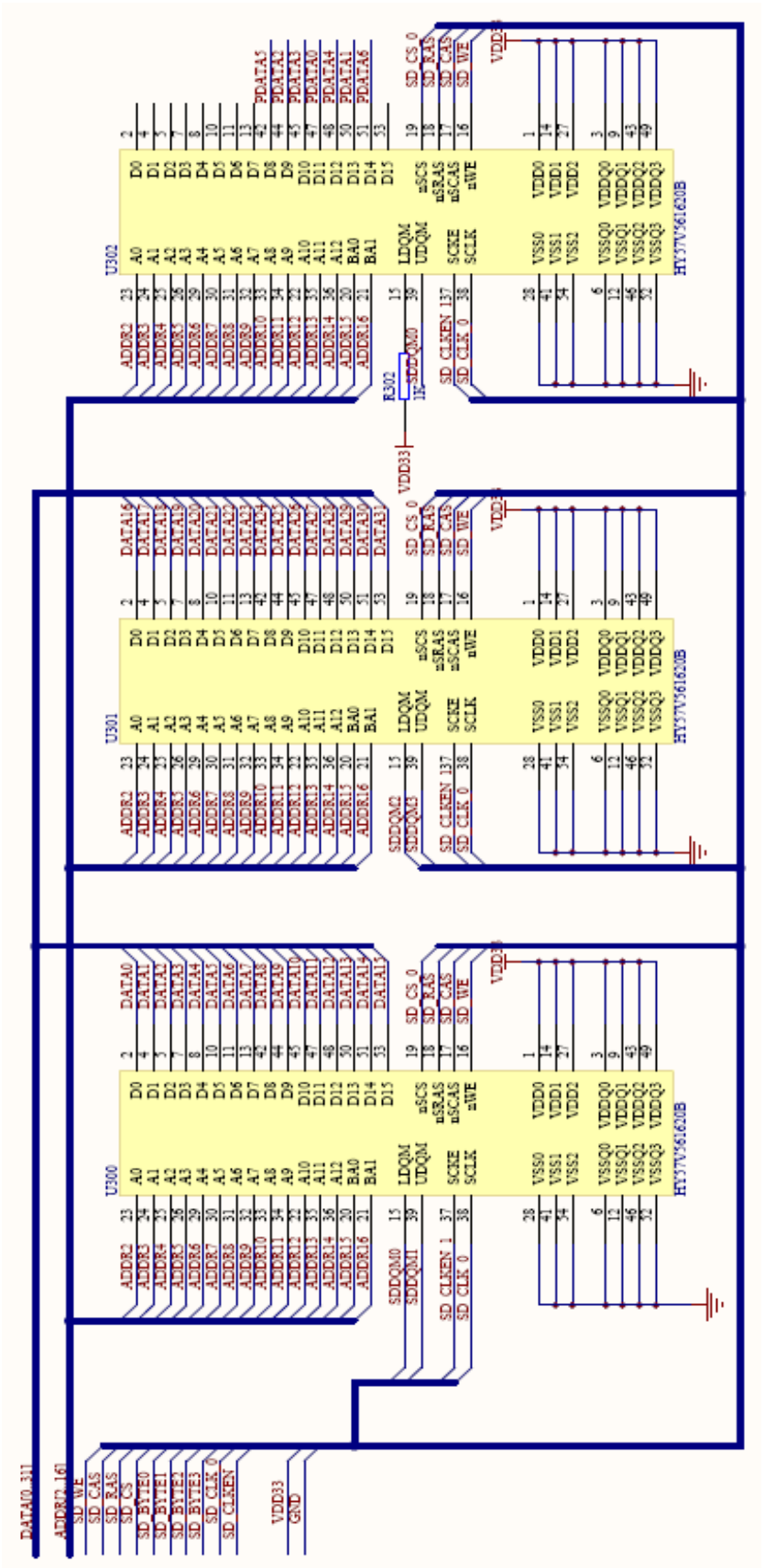
PROM 原理图



SRAM 原理图


U16
U14
U13
CY7C1041BV33
ADDR2
ADDR3
ADDR4
ADDR5
ADDR6
ADDR7
ADDR8
ADDR9
ADDR10
ADDR11
ADDR12
ADDR13
ADDR14
ADDR15
ADDR16
ADDR17
ADDR18
ADDR19
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
A16
A17
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
CE
WE
OE
BHE
BLE
NC
VCC
VSS
SRAM_CS_0
SRAM_WE0
SRAM_WE2
SRAM_OE_0
VDD33
GND
PDATA2
PDATA5
PDATA3
PDATA0
PDATA4
PDATA1
PDATA6
DATA8
DATA9
DATA10
DATA11
DATA12
DATA13
DATA14
DATA15
DATA0
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
DATA24
DATA25
DATA26
DATA27
DATA28
DATA29
DATA30
DATA31
DATA23
DATA22
DATA21
DATA20
DATA19
DATA18
DATA17
DATA16

SDRAM 原理图


U302
U301
U300
HY57V561620B
R302
VDD33
GND
SD_CS_0
SD_RAS
SD_CAS
SD_WE
SD_CLKEN
SD_CLK_0
LDQM
UDQM
SCKE
SCLK
VDD0
VDD1
VDD2
VDDQ0
VDDQ1
VDDQ2
VDDQ3
VSS0
VSS1
VSS2
VSSQ0
VSSQ1
VSSQ2
VSSQ3
SD_BYTE0
SD_BYTE1
SD_BYTE2
SD_BYTE3
SD_CS

H 软件开发规范

H.1 BM3803 寄存器介绍

H.1.1 寄存器堆介绍

对程序员和编译器来说，如何高效使用寄存器是软件设计中的关键问题。BM3803 处理器向软件提供窗口寄存器，全局整数寄存器和浮点寄存器。

窗口寄存器（%i0~%i7、%l0~%l7、%o0~%o7）

窗口寄存器由输入寄存器（in register）、本地寄存器（local register）和输出寄存器（out register）组成。

输入、输出寄存器主要用于向子程序传递参数、接收子程序运行结果和跟踪存储器堆栈。当程序调用时，调用程序（caller）的输出寄存器成为被调程序（callee）的输入寄存器。

第六个输出寄存器（%o6）保存栈指针（%sp），此指针指向当前堆栈的栈顶。此时，%o6 不再作为保存程序传递参数使用。因此，输出寄存器最多可以传递 6 个参数（使用%o0 ~ %o5），更多参数通过堆栈传递。

第六个输入寄存器（%i6）保存堆指针（%fp），此指针指向当前堆栈的栈底。此时，%i6 不再作为保存程序传递参数使用。

本地寄存器用于保存自动类型的变量（如 C 语言中的 auto 型变量）和大部分临时值。

全局寄存器（%g0~%g7）

与输入、本地和输出寄存器不同，全局寄存器不属于任何寄存器窗口。全局寄存器由 8 个全局范围的寄存器组成，类似传统处理器结构中的寄存器。全局寄存器%g0 写无效，读出恒为零。其余全局寄存器（%g1 ~ %g7）可以用于保存临时变量、全局变量或全局指针（用户变量或作为程序执行环境一部分保存的值）。

浮点寄存器（%f0~%f31）

BM3803 有 32 个 32 位浮点寄存器。与全局寄存器类似，浮点寄存器必须通过软件管理。编译器使用浮点寄存器保存用户变量、编译器临时变量和运算中的浮点数值。

H.1.2 特殊寄存器介绍

处理器状态寄存器（Processor State Register，PSR）

32 位的 PSR 包含了多种字段来控制处理器和保持状态信息。PSR 可以被 SAVE，RESTORE，Ticc 和 RETT 以及所有可以修改条件码（condition code）的指令所修改。特权指令 RDPSR 和 WRPSR 可以直接对 PSR 进行读和写。

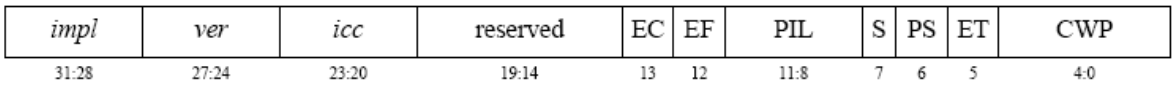


图 H-1-1 PSR

PSR 提供了以下字段：

PSR_ implementation (impl)：处理器状态寄存器的第 28 位到 31 位是只读的，用来标识芯片实现信息。

PSR_ version（ver）：第 24 位到第 27 位是只读的，用来表示芯片版本信息。

PSR_integer_cond_codes(icc)：第 20 到 23 位是整数单元的条件码字段。icc 字段的各位可以被以字母 cc 为结尾命名的算术和逻辑指令以及 WRPSR 指令所修改。Bicc 和 Ticc 指令基于 icc 字段各位的值进行控制转移。icc 各个位定义如下：

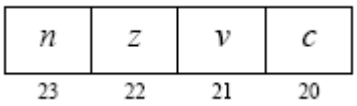


图 H-1-2 PSR 中 icc 字段

PSR_negative(n)：指出了对于最后一条修改 icc 字段的指令，算术逻辑运算单元的 32 位的二进制补码结果是否为负数。1 表示为负数，0 表示为非负数。

PSR_zero(z)：表示对于最后一条修改 icc 字段的指令，算术逻辑运算单元的 32 位结果是否为 0。1 表示为 0，0 表示非 0。

PSR_overflow(v)：表示对于最后一条修改 icc 字段的指令，算术逻辑运算单元的结果是否在 32 位二进制补码表数法的表数范围之内。1 表示溢出，0 表示未溢出。

PSR_carry(c)：表示对于最后一条修改 icc 字段的指令，是否产生了二进制补码的进位（或者借位）。如果位 31 上产生了进位，则在进行加法运算时设置成 carry。如果位 31 产生借位，则在进行减法运算时设置为 carry。1 表示设置成 carry，0 表示未设置成 carry。

PSR_reserved：14 到 19 位为保留位。当一条 RDPSR 指令进行读操作的时候，读这些位得到 0。为了将来的兼容性，管理软件仅仅在执行 WRPSR 指令的时候将这个字段设成值 0。

PSR_ enable_ coprocessor(EC)：位 13 表示是否支持协处理器。BM3803 不包含协处理器，这一位只读，读出为 0。

PSR_ enable_ floating- point(EF)：位 12 决定了是否使能浮点处理单元。1 表示使能，0 表示不使能。

PSR_ proc_ interrupt_level(PIL)：位 8（最低有效位）到位 11（最高有效位）指定处理器将要接收的陷阱的级别。

PSR_ supervisor(S)：确定处理器处于用户态还是管理态。1 表示管理态，0 表示用户态。

PSR_ previous_ supervisor(PS)：保留最近一次中断发生时 S 位中的值。

PSR_ enable_ traps(ET)：表示是否使能陷阱。一个陷阱中断自动的将 ET 复位为 0。当 ET=0 时，忽略中断请求并且异常陷阱会导致整数处理单元停止执行，从而导致一个复位陷阱，这个复位陷阱会在地址 0 处恢复执行。该位为 1 表示使能陷阱，为 0 表示禁止陷阱。

PSR_ current_ window_ pointer(CWP)：位 0（最低有效位）到位 4（最高有效位）组成了当前窗口指针，一个在 r 寄存器中识别当前寄存器窗口的计数器。在陷阱发生或者 SAVE 指令执行时硬件减少 CWP，在 RESTORE 和 RETT 指令执行时增加 CWP（模 NWINDOWS 运算）。

窗口无效掩码寄存器（Window Invalid Mask Register, WIM）

窗口无效掩码寄存器由管理软件控制，硬件利用 WIM 来决定执行一条 SAVE，RESTORE 或者 RETT 指令是否会产生窗口的溢出或下溢陷阱。

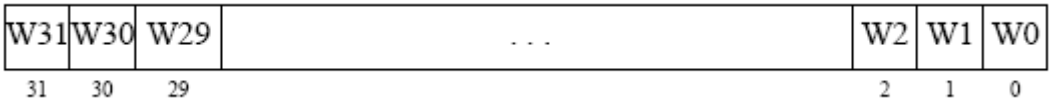


图 H-1-3 WIM

BM3803 有 8 个寄存器窗口，因此该寄存器的低 8 位有效。

当执行 SAVE，RESTORE 或者 RETT 指令时，CWP 的当前值与 WIM 比较。如果 SAVE，RESTORE 或者 RETT 指令使 CWP 指向一个“无效”寄存器组，也就是对应的 WIM 位等于 1（WIM[CWP]=1）的寄存器组的话，就会产生一个窗口溢出或者窗口下溢陷阱。

可以通过特权指令 RDWIM 来读取 WIM 的值，通过 WRWIM 指令对 WIM 进行写操作。对应于未实现的窗口的位为 0，并且对未实现窗口对应位进行写操作没有作用。执行 WRWIM 指令将所有的 WIM 的位置为 1，随后执行 RDWIM 指令，会得到一个位向量，向量中已经实现的窗口（仅仅是实现的窗口）声明为 1。

异常基址标志寄存器（Trap Base Register，TBR）

异常基址标志寄存器包含 3 个字段。当陷阱发生时，这 3 个字段共同构成了控制转移的目的地址。

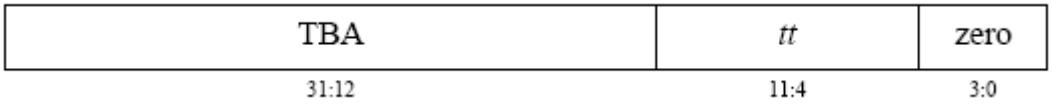


图 H-1-4 TBR

TBR 提供了以下字段：

TBR_ trap_ base_ address (TBA)：位 12 到位 31 位为陷阱的基址，这个基址是由管理软件产生的。它包含了高位的 20 位有效位和陷阱表地址。可以通过 WRTBR 指令来对 TBA 字段进行写操作。

TBR_ trap_ type(tt)：位 4 到位 11 为陷阱类型字段。当陷阱发生时，硬件对这个 8 位字段进行写入，并且保持它的值不变直到下一个陷阱发生。它在陷阱表中提供了一个偏移量。WRTBR 指令的执行不会影响 tt 字段的值。

TBR_ zero(z)：位 0 到位 3 全为 0。WRTBR 指令的执行不会影响到这个字段。为了以后的兼容性，管理软件应该只能在这个字段的值为 0

的时候发出 WRTBR 指令。

乘法/除法寄存器（Multiply/Divide Register, Y）

32 位的 Y 寄存器的内容为整数乘法的双精度结果的最高有效字，作为执行整数乘法指令（SMUL，SMULcc，UMUL，UMULcc）的结果或者用到整型乘法步指令(MULScc)的例行程序的结果。Y 寄存器还保存一个整型除法指令(SDIV, SDIVcc, UDIV, UDIVcc)的双精度被除数的最高字。Y 寄存器由 RDY 和 WRY 指令读写。

程序计数器（Program Counters, PC, nPC）

32 位程序计数器保存着 IU 正在执行的指令的地址。而 nPC 保存着将要被执行的下一条指令的地址（假设不发生陷阱）。

对于延迟控制转移，将紧随在传输指令后面的指令当作延迟指令。在控制转移到目标程序前执行这个延迟指令（除非控制转移指令取消了这条延迟指令）。在延迟指令执行的过程中，nPC 指向控制转移指令的目标程序，而 PC 则指向延迟指令。

可以通过 CALL 或者 JMPL 指令读取 PC 的值。发生陷阱时，将 PC 和 nPC 的值写入两个局部寄存器。

辅助状态寄存器（Ancillary State Registers, ASR）

BM3803 实现了 11 个辅助状态寄存器（ASR's），编号分别是 16、17、18、24~31。

ASR16、17 用于 IU 容错，ASR18 用于乘加运算，ASR24~31 用于硬件观察点。

为了将来的使用，体系结构将编号 1 到 15 的 ASR 保留起来，软件无法访问这些 ASR。

可以通过 RDASR 和 WRASR 指令来对辅助状态寄存器进行读写操作。如果被存取访问的寄存器是特权寄存器，则读/写辅助状态寄存器的指令是特权指令。

浮点状态寄存器（Floating-point State Register FSR）

FSR 寄存器的各字段包含了 FPU 的模式和状态信息。我们可以通过 STFSR 指令和 LDFSR 指令对 FSR 进行读写操作。

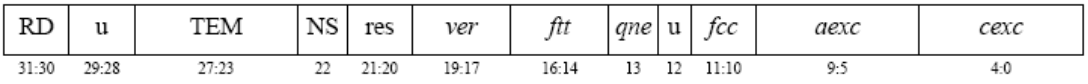


图 H-1-5 FSR

FSR_ rounding_ direction(RD)：位 30 和 31 根据 ANSI/IEEE 的 754-1985 标准选择浮点数结果的舍入方式。

表 H-1-1 FSR 的舍入方式选择（RD）字段

RD	Round Toward:
0	Nearest (even, if tie)
1	0
2	+∞
3	− ∞

FSR_ unused（u）：位 28、29 和位 12 为未使用位。为了以后的兼容性，软件应该仅仅在这几位为 0 的情况下发出 LDFSR 指令。

FSR_ trap_ enable_ mask(TEM)：23 至 27 位为每一个五位的浮点异常的使能位，这些浮点异常是由 current_ exception 字段（cexc）所标识的。如图 4-9 所示。如果一个浮点操作指令产生了一个或多个异常，并且将对应于这一个或多个异常的 TEM 位置 1，就会产生一个浮点异常的陷阱。TEM 置 0 可以防止那种异常类型产生陷阱。

FSR_ nonstandard_ fp (NS)：位 22 为 NS 位。若这位置 1，有可能导致 FPU 产生一个不符合 ANSI/IEEE754-1985 标准的实现相关的结果。例如，在 NS 置位时，为了获得更高的性能，实现可以将一个低于正常值（subnormal）的浮点操作数或结果转换成 0。

FSR_ reserved(res)：20 和 21 位为保留位。当 STFSR 指令读取这些位时，得到 0 值。

FSR_ version(ver)：17 至 19 位识别 FPU 的实现。

FSR_ floating-point_ trap_ type(ftt)：14 至 16 位用来识别浮点异常陷阱的类型。一个浮点异常发生以后，ftt 字段将浮点异常的类型进行编码直到执行了一条 STFSR 或者其他的 FPop 指令。

可以通过 STFSR 指令来读取 ftt 字段的内容。LDFSR 指令的执行不会影响 ftt 字段。

这个字段根据表 1-2 对异常类型编码。

表 H-1-2 FSR 的浮点陷阱类型（ftt）字段

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*

Sequence_ error 和 hardware_ error 需要通过复位恢复。

IEEE_ 754_ exception，unfinished_ FPop 和 unimplemented_ FPop 有时候可以在正常的计算过程中出现而且必须可以被管理软件恢复。当一个浮点陷阱发生的时候（被用户信号处理程序发现）：

1、aexc 的值不会改变；

2、除了发生 IEEE_ 754_ exception 类型的异常时，与要进入陷阱的异常相对应的那一位会被置位外， aexc 的值不会改变。Unfinished_ FPop，unimplemented_ FPop 以及 sequence_ error 几个浮点异常都不会影响 aexc 的值；

3、源 f 寄存器不会被改变；

4、fcc 字段的值不会改变。

在 unfinished_ FPop 和 unimplemented_ FPop 陷阱后不会产生 IEEE 异常的情况中，正常地，我们期望有恢复软件来定义 cexc，aexc，以及目的 f 寄存器或者 fcc 的值。

ftt=IEEE_ 754_ exception： IEEE_ 754_ exception 浮点中断类型意味着产生了一个符合 ANSI/IEEE 754-1985 标准的异常。这个异常类型在 cexc 字段中编码。要注意的是，aexc，fcc 和目标 f 寄存器都不受 IEEE_ 754_ exception 的影响。

ftt=unfinished_ FPop：unfinished_ FPop 表示一个实现的 FPU 不能产生 ANSI/IEEE 754-1985 标准所定义的正确的结果或者异常。在这种情况下，cexc 字段不能被修改。

ftt=unimplemented_ FPop：一个实现的 FPU 中解码到了 FPU 中没有实现的浮点操作(FPop)。这种情况下，cexc 字段不能被修改。

ftt=sequence_ error：表示 FPU 中三种异常错误条件之一，这三种异常错误条件全部由管理软件的错误操作导致的：

— 想要在一个不支持浮点延迟中断队列（FQ）的实现中执行 STDFQ 指令；

— 想要在 FPU 无法接受浮点指令时执行浮点指令。这种类型的 sequence_ error 由于管理软件中的逻辑错误造成，并且这个管理软件先前产生的一个陷阱还没有被完全处理（例如，以前的浮点异常处理完以后，浮点队列依然非空）；

— 想要在浮点延迟中断队列为空的时候也就是当 FSR.qne=0 的时候执行 STDFQ 指令。（注意，这种情况下我们推荐产生一个 sequence_ error 中断，但这种情况下不要求）

ftt=hardware_ error：意味着 FPU 发现发生了严重的内部错误，比如非法状态或者访问 f 寄存器时的奇偶校验错误。

如果在执行用户代码的过程中产生 hardware_ error 陷阱，有可能无法恢复足够的状态，来继续执行用户应用程序。

ftt=invalid_ fp_ register：这种陷阱类型表示 FPop 中的一个（或者多个）操作数未对齐，也就是说，双精度操作数寄存器编号做模 2 运算不是 0，或者四倍精度操作数寄存器的编号做模 4 运算不是 0。在这种情况下，建议实现通过令 FSR.ftt=invalid_ fp_ register 来产生一个 fp_ exception 类型的陷阱。但是实现也可以选择不产生这个陷阱。

FSR_ FQ_ not_ empty(qne)：这一位表示当一个延迟了的 fp_ exception 陷阱发生或者执行了一条存储双精度浮点数队列（STDFQ）指令后，可选的浮点延迟陷阱队列（FQ）是否为空。如果 qne=0，这个队列为空；如果 qne=1，这个队列非空。

可以通过 STFSR 指令来读取 qne 位。LDFSR 指令的执行不会影响 qne 位。然而，执行连续的 STDFQ 指令会（最终）导致 FQ 变为空（qne=0）。如果一个实现不提供 FQ，读这一位得到 0。管理软件必须调整这一位，

使得对于用户软件，读取该位永远得到 0。

FSR_ fq_ condition_ codes(fcc)：第 10 和 11 位是 FPU 的条件码。执行浮点数比较指令（FCMP 和 FCMPE 指令）会修改这两位。我们分别通过 STFSR 和 LDFSR 指令来对这两位进行读和取。FB fcc 的控制转移就是基于这个字段的。

在下表中，f_{rs1}和 f_{rs2}对应着 f 寄存器中通过指令的 rs1 和 rs2 字段指示的单精度，双精度或者四倍精度浮点数的值。问号（？）指的是一种无法比较的关系，当 f_{rs1}或者 f_{rs2}为 signaling 非数或者 quiet 非数。注意：如果 FCMP 或者 FCMPE 执行时产生一个 IEEE_ 754_exception 陷阱，fcc 的值不会改变。

表 H-1-3 FSR 的浮点数条件码字段（fcc）

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)：当禁止 fp_ exception 类型的陷阱利用 TEM 字段时，FSR 的第 5 至第 9 位既 aexc 字段用来累计 IEEE_ 754 类型的浮点异常。如图 4-10 所示，一条浮点操作指令完成之后，将 TEM 和 cexc 两个字段进行逻辑与操作。如果结果非零，将会产生一个 fp_ exception 陷阱；否则，新的 cexc 字段与 aexc 字段进行或操作后将结果保留到 aexc 字段中。这样，当中断被屏蔽的时候，异常就会累计在 aexc 中。

FSR_ current_ exception(cexc)：这个字段表示由于最后执行的 FPop 指令的执行，产生了一个或者多个 IEEE_ 754 浮点异常。如果没有异常，对应的位会被清 0。如图 4-11 所示。

如在 4.4.2 节描述的那样，可以通过执行 FPop 指令来设置 cexc 中的各位，而且 FPop 指令的执行既不会产生陷阱也不会使 FSR.ftt=IEEE_ 754_ exception 而导致产生 fp_ exception 类型的陷阱。若发现 IEEE 754 异常，建议通过触发 IEEE_ 754_ exception 陷阱来对 FSR.cexc 中的对应

该异常的位进行置位。如果是执行 FPop 指令导致中断的产生，而不是因为 IEEE 754 异常产生的 fp_ exception 类型的陷阱的话，FSR.cexc 的内容不会被修改。

浮点数异常字段（Floating-Point Exception Fields）

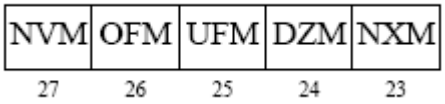


图 H-1-6 FSR 中的陷阱允许屏蔽字段

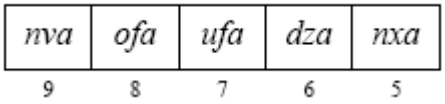


图 H-1-7 FSR 已发生的异常位字段

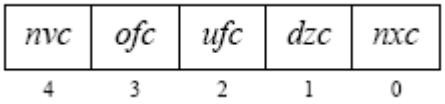


图 H-1-8 FSR 当前异常位字段

FSR_ invalid(nvc,nva)：对于将要执行的操作，操作数为非法操作数。比如，$0\div0$和$\infty-\infty$运算都是无效运算。该位为 1 表示无效操作数，为 0 表示合法操作数。

FSR_overflow(ofc,ofa)：在某种数据格式中，舍入后的结果可能会在数量级上大于正常数值的上限。1 表示上溢，0 表示未上溢。

FSR_underflow(ufc,ufa)：在指定的数据格式中，舍入后的结果可能会不精确而且会在数量级上小于正常数值得下限。1 表示下溢出，0 表示未下溢出。

如果未进行舍入的结果是正确的，并且为 0，则下溢出永远不会发生，除非以下两种情况：

如果 UFM=0：如果一个操作执行后产生的未进行舍入操作的结果是正确的，并且在数量级要小于正常数值的下限，并且进行正确舍入后结果不够精确，这样就会对 ufc 和 ufa 两位进行置位。在正确结果未进行舍入的时候小于该数据类型的表数范围的下界，而进行正确舍入后的结果就是该数据类型的表数范围的下界，这时这两位也会被置位。nxc 和 nxa

往往也会被置位。

如果 UFM=1：如果正确的结果未进行舍入时小于该数据类型的表数范围的下界，则会触发一个 IEEE-754_ exception 类型的陷阱。正确结果未进行舍入的时候小于该数据类型的表数范围的下界，而进行舍入后正确结果就是该数据类型的表数范围的下界时会发生陷阱。

FSR_ division-by-zero(dzc,dza)：$X \div 0$，X 为非正常操作数或者正常操作数时，对该位置位。注意$0 \div 0$运算不会将 dzc 置位。置 1 时除数为 0，置 0 表示除数不为 0。

FSR_ inexact(nxc,nxa)：舍入后的结果与正确的无限精确结果不同时进行置位。置 1 表示不精确结果，置 0 表示精确结果。

H.1.3 寄存器推荐使用方法汇总

		(%r31)	返回地址 －8
	%fp，%i6	(%r30)	堆指针
	%i5	(%r29)	输入参数 6
in	%i4	(%r28)	输入参数 5
	%i3	(%r27)	输入参数 4
	%i2	(%r26)	输入参数 3
	%i1	(%r25)	输入参数 2
	%i0	(%r24)	输入参数 1 / 向调用函数返回的值
	%i7	(%r23)	本地变量 7
	%l6	(%r22)	本地变量 6
	%l5	(%r21)	本地变量 5
local	%l4	(%r20)	本地变量 4
	%l3	(%r19)	本地变量 3
	%l2	(%r18)	本地变量 2
	%l1	(%r17)	本地变量 1
	%l0	(%r16)	本地变量 0
	%o7	(%r15)	临时值 / CALL 指令的地址
	%sp，%o6	(%r14)	栈指针
	%o5	(%r13)	输出参数 6
out	%o4	(%r12)	输出参数 5
	%o3	(%r11)	输出参数 4
	%o2	(%r10)	输出参数 3
	%o1	(%r9)	输出参数 2
	%o0	(%r8)	输出参数 1 / 从被调函数返回的值
	%g7	(%r7)	全局变量 7 (SPARC ABI：保留)

	%g6	(%r6)	全局变量 6 (SPARC ABI：保留)
	%g5	(%r5)	全局变量 5 (SPARC ABI：保留)
global	%g4	(%r4)	全局变量 4 (SPARC ABI：全局reg型变量)
	%g3	(%r3)	全局变量 3 (SPARC ABI：全局reg型变量)
	%g2	(%r2)	全局变量 2 (SPARC ABI：全局reg型变量)
	%g1	(%r1)	临时值
	%g0	(%r0)	0
state	%y		Y寄存器 (用于乘/除法)
	(*icc* field of %psr)		整数状态位
	(*fcc* field of %fsr)		浮点数状态位
	%f31		浮点值
floating	:		:
point	:		:
	%f0		浮点值

H.2 BM3803 指令介绍

H.2.1 加载整数指令

操作码	操作
LDSB	加载有符号字节
LDSH	加载有符号半字
LDUB	加载无符号字节
LDUH	加载无符号半字
LD	加载字
LDD	加载双字
LDA†	从辅助交换空间加载字

特权指令

汇编语言语法	
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* , *reg*$_{rd}$

例如：

从内存地址 0x80000000 处加载数据到%o1：

```
set    0x80000000,%o0
ld [%o0],%o1
```

或者，直接给出地址 200：

```
ld  [200],%o1
```

对于 ASI 指令：

```
lda [%o0]0xc,%o1
```

说明：

加载整型数据指令从存储器中复制一个字节、半字或者字到 r[rd]中。读取的字节或半字在目标寄存器 r[rd]中是右对齐的；左侧根据不同指令，分别对有符号操作和无符号操作进行符号扩展和补零。其中，addrsss 可以为寄存器或 13 位立即数。

加载整型双字指令(LDD，LDDA)把一个双字从存储器复制一个 r 寄

存器对中。有效内存地址的高有效字被移入偶 r 寄存器中，而（在有效内存地址加 4 的地方的）低有效字被移入接下来的奇 r 寄存器。(注意：当 regrd 为%g0 时加载双字指令只改变%g1)并且，加载双字指令的 regrd 只能是偶数号寄存器，当 r[rd]为奇数号寄存器（比如，%o1）时，将引起一个 illegal_instruction 陷阱。

对于加载指令的有效地址，如果 i 字段为 0，是“r[rs1] + r[rs2]”，如果 i 字段是 1，则是“r[rs1] + sign_ext(simm13)”。对于带 ASI 参数的加载指令，在 asi 字段包含着加载所用的地址空间标识符，而且 i 字段必须为 0，否则会出现一个 illegal_instruction 陷阱。不带 ASI 参数的加载指令根据 PSR 的 S 位的值访问用户数据空间或是系统数据空间。

当有效地址不是字对齐时执行 LD、LDA；当有效地址不是半字对齐时执行 LDUH、LDSH；当有效地址不是双字对齐时执行 LDD，均会引起一个 mem_address_not_aligned 陷阱。

当指令带 ASI 参数时，指令会根据不同的 ASI 值影响 cache。具体规则如下表所示：

ASI	用法	备注
0x0,0x1 0x2,0x3	强制cache不命中 (如果数据已缓存时进行替换)	被加载数据按照指令给出的长度装入r[rd]
0x4,0x7	强制cache不命中 (如果命中也更新)	被加载数据按照指令给出的长度装入r[rd]
0x5	刷新指令 cache	对r[rd]无影响
0x6	刷新数据 cache	对r[rd]无影响
0x8, 0x9 0xA, 0xB	正常 cache 访问(如果数据已缓存时进行替换)	同不带ASI参数的指令功能相同
0xC	指令 cache tags	加载指令cache tags到r[rd]
0xD	指令 cache data	加载指令cache data到r[rd]
0xE	数据 cache tags	加载数据cache tags到r[rd]
0xF	数据 cache data	加载数据cache data到r[rd]

通常使用 LDA 指令对 cache 进行操作。这里，regaddr 为 cache 的绝对地址，且 regaddr 应小于 0x7fff。如果 regaddr 大于该值，则仅取低 15 位数据计算地址。

陷阱：

llegal_instruction　　//非法指令(i = 1 时加载交换空间; rd 为

奇时的 LDD)

privileged_instruction //特权指令(从交换空间中加载)

mem_address_not_aligned //存储器地址未对齐(LDSB，LDUB 不会引起该陷阱)

data_access_exception //数据访问异常

data_access_error //数据访问错误

H.2.2 加载浮点指令

擦作码	*操作*
LDF	加载浮点寄存器
LDDF	加载双精度浮点寄存器
LDFSR	加载浮点状态寄存器

汇编语言语法	
ld	[*address*], *freg*$_{rd}$
ldd	[*address*], *freg*$_{rd}$
ld	[*address*], %fsr

说明：

加载单精度浮点数指令(LDF)把一个字从存储器移入 f[rd]。

加载双字浮点指令(LDDF)把一个双字从存储器移入 f 寄存器对。有效存储器地址处的高有效字被移入偶数 f 寄存器，在有效存储器地址+4 处的低有效字被移入奇数 f 寄存器。rd 字段的最低有效位没有使用，应该软件置 0。如果此位非零， LDDF 会起一个 fp_exception 陷阱而且 FSR.ftt = invalid_fp_register。

加载浮点状态寄存器指令(LDFSR) 等待所有未完全完成的 FPop 指令，然后从存储器加载一个字放入 FSR。如果紧跟着一条 LDFSR 指令的三条指令中任一条是 FBfcc，那么 FBfcc 所见的 FSR 的 fcc 字段是不确定的。

如果 i 字段是 0，加载指令的有效地址是 “r[rs1] + r[rs2]”；如果 i 字段是 1，那么有效地址为 “r[rs1] + sign_ext(simm13)”。

如果有效地址不是字对齐的，LDF 和 LDFSR 引起一个 mem_address_not_aligned 陷阱；如果不是双字对齐的，LDDF 进入陷阱；如果 PSR 的 EF 字段是 0，一个加载浮点数指令触发一个 fp_disabled 陷

阱。

陷阱：

fp_disabled　　//FPU 禁用

fp_exception (sequence_error, invalid_fp_register(LDDF))　//FPU　异常（对列错误；浮点寄存器不可用）

data_access_exception　　//数据访问异常

data_access_error　　//数据访问错误

mem_address_not_aligned　　//存储器地址为对齐

H.2.3 存储整数指令

操作码	*操作*
STB STH ST STD	存储字节 存储半字 存储字 存储双字
STA†	向交换空间里存储字

† 特权指令

汇编语言语法			
stb	*reg*$_{rd}$,	[*address*]	*(等价指令:* stub, stsb*)*
sth	*reg*$_{rd}$,	[*address*]	*(等价指令:* stuh, stsh*)*
st	*reg*$_{rd}$,	[*address*]	
std	*reg*$_{rd}$,	[*address*]	
sta	*reg*$_{rd}$,	[*regaddr*] *asi*	

例如：

将寄存器%o1 中的数据存储到内存地址 0x80000000 处：

```
set    0x80000000,%o0
st %o1,[%o0]
```

或者，直接给出存储地址：

```
st  %o1,[200]
```

对于 ASI 指令：

```
sta %o1,[%o0] 0xd
```

说明：

存储整数指令把字、半字或字节从 r[rd]复制到存储器中。对于半字

和字节，只存储 r[rd]中相应大小的低有效位。

存储整型双字指令(STD，STDA)从一对 r 寄存器内复制一个双字到存储器中。字的高有效位（存储于偶数 r 寄存器）写入指令给出的有效地址所指向的存储器中，字的低有效位（存储于随后的奇数 r 寄存器）写入“有效地址+4”的存储器地址处。存储整型双字指令的 regrd 只能是偶数号寄存器；当 regrd 是奇数号寄存器时，执行存储双字指令引起一个 illegal_instruction 陷阱。

对于存储指令的有效地址，如果 i 字段是 0，是“r[rs1] + r[rs2]”；如果 i 字段为 1，是“r[rs1] + sign_ext(simm13)”。使用带 ASI 参数的存储指令时，i 字段必须为 0，否则将引发 illegal_instruction 陷阱。不带 ASI 参数的存储指令会根据 PSR 的 S 位来访问用户数据空间或者系统数据空间。

当有效地址不是字对齐时执行 ST、STA；当有效地址不是半字对齐时执行 STH、STHA；当有效地址不是双字对齐时执行 STD、STDA，均会引起一个 mem_address_not_aligned 陷阱。

对于带 ASI 参数的存储指令，ASI 参数影响如下表所示：

ASI	用法	备注
0x0,0x1 0x2,0x3	强制cache不命中 (如果数据已缓存时进行替换)	被存储数据按照指令给出的长度送入r[rd]
0x4,0x7	强制cache不命中 (如果命中也更新)	被存储数据按照指令给出的长度送入r[rd]
0x5	刷新指令 cache	对r[rd]无影响
0x6	刷新数据 cache	对r[rd]无影响
0x8, 0x9 0xA, 0xB	正常 cache 访问(如果数据已缓存时进行替换)	同不带ASI参数的指令功能相同
0xC	指令 cache tags	存储r[rd]到cache tags
0xD	指令 cache data	存储r[rd]到cache data
0xE	数据 cache tags	存储r[rd]到cache tags
0xF	数据 cache data	存储r[rd]到cache data

陷阱：

illegal_instruction //非法指令

(i=1 时存储交换空间；使用 STD 和 STDA 时 rd 为奇地址)

privileged_instruction //特权指令（仅针对向交换空间里存储的指令）

mem_address_not_aligned //存储器地址未对齐（STB 和 STBA 不会引起该陷阱）

data_access_exception //数据访问异常

data_access_error //数据访问错误

data_store_error //数据存储错误

H.2.4 存储浮点指令

操作码	*操作*
STF	存储单精度浮点
STDF	存储双精度浮点
STFSR	存储浮点状态寄存器
STDFQ†	在陷阱延迟队列中存储双精度浮点

†特权指令

汇编语言语法
st *freg*$_{rd}$,[*address*]
std *freg*$_{rd}$,[*address*]
st %fsr,[*address*]
std %fq, [*address*]

说明：

存储单精度浮点数指令(STF)把 f[rd]的内容复制到存储器中。

存储双浮点数指令(STDF)把一对相邻的 f 寄存器中的双字复制到存储器中。字的高有效位（存储于偶数 f 寄存器中）写入给定有效地址对应的存储器，低有效位（存储于奇数 f 寄存器中）写入“有效地址 +4”所对应的存储器地址。rd 字段的最低有效位是未使用的，应该总是被软件设置为 0。若此位非 0，STDF 会引起一个 fp_exception 陷阱，同时 FSR.ftt = invalid_fp_register。

存储浮点状态寄存器指令(STFSR)等待同时执行但没有完成的 FPop 指令，然后把 FSR 写入存储器。把 FSR 写入存储器后 STFSR 会把 FSR.ftt 归零。

对于指令给出的地址段，如果 i 字段为 0，存储指令的有效地址为“r[rs1] + r[rs2]”；如果 i 字段为 1，有效地址为 “r[rs1] + sign_ext(simm13)”。

如果地址不是字对齐的，STF 和 STFSR 会引起一个 mem_address_not_aligned 陷阱；如果地址不是双字对齐的，STDF 会进入陷阱。如果 PSR 的 EF 字段为 0，存储浮点指令会引

起一个 fp_disabled 陷阱。.

陷阱：

fp_exception (invalid_fp_register(STDF))　//FPU 异常（FPU 寄存器无效[STDF]）

mem_address_not_aligned　　//存储器地址未对齐

data_access_exception　　//数据访问异常

data_access_error　　//数据访问错误

data_store_error　　//数据存储错误

H.2.5 原子加载-存储无符号字节指令

操作码	*操作*
LDSTUB	原子加载-存储无符号字节

汇编语言语法	
ldstub	[*address*],*reg*$_{rd}$

例如：

从地址 0x40000010 处读取一字节数据到%l0，并在该处存储一个字节的 1：

```
set     0x40000010,%o0
ldstub  [%o0],%l0
```

说明：

原子加载-存储指令从存储器中复制一个字节的数据到 r[rd]中，然后向刚才读出数据的存储器地址位置写全 1。操作自动执行，不允许中断或者延迟陷阱插入干涉。

如果 i 字段为 0，指令有效地址为“r[rs1] + r[rs2]”；如果 i 字段为 1，指令有效地址为“r[rs1] + sign_ext(simm13)”。存储器访问所用的地址空间标识符来自 asi 字段。对 LDSTUB 来说,根据 PSR 的 S 位决定，地址空间是用户数据空间还是系统数据空间。

陷阱：

illegal_instruction //非法指令（i=1 时执行 LDSTUBA）

privileged_instruction //特权指令（只有 LDSTUBA 会引起该陷阱）

data_access_exception //数据访问异常

data_access_error //数据访问错误

data_store_error //数据存储错误

H.2.6 寄存器与存储器交换

操作码	*操作*
SWAP SWAPA†	交换寄存器与内存中的数据 交换寄存器与交换空间内存中的数据

†特权指令

汇编语言语法	
swap	[*address*],*reg*$_{rd}$
swapa	[*regaddr*]*asi* ,*reg*$_{rd}$

例如：

交换地址 0x40001000 处和寄存器%o1 中的数据：

```
set     0x40001000,%o0
swap    [%o0],%o1
```

交换指令 cache data 和寄存器%o1 中的数据：

```
set     0x40001000，%o0
swapa [%o0] 0xd,%o1
```

说明：

SWAP 和 SWAPA 指令交换 r[rd]和指定的存储器单元中一个字的内容。这个操作自动执行，并且不会允许中断或者延迟陷阱插入干预。

如果 i 字段为 0，指令有效地址是“r[rs1] + r[rs2]”；如果 i 字段为 1，指令有效地址是“r[rs1] + sign_ext(simm13)”。对 SWAP 来说，根据 PSR 的 S 位决定地址指向用户数据空间还是系统数据空间。

如果有效地址不是字对齐的，指令引起一个 mem_address_not_aligned 陷阱。

对于 SWAPA 指令，参数 ASI 作用如下表所示：

ASI	用法	备注
0x0,0x1 0x2,0x3	强制cache不命中 (如果数据已缓存时进行替换)	存储器与寄存器内容交换
0x4,0x7	强制cache不命中 (如果命中也更新)	存储器与寄存器内容交换
0x5	刷新指令 cache	只从存储器中读出数据到r[rd]，并不改变存储器中内容
0x6	刷新数据 cache	只从存储器中读出数据到r[rd]，并不改变存储器中内容
0x8, 0x9 0xA, 0xB	正常 cache 访问(如果数据已缓存时进行替换)	同不带ASI参数的指令功能相同
0xC	指令 cache tags	交换r[rd]与cache tags的内容，不影响存储器
0xD	指令 cache data	交换r[rd]与cache data的内容，不影响存储器
0xE	数据 cache tags	交换r[rd]与cache tags的内容，不影响存储器
0xF	数据 cache data	交换r[rd]与cache data的内容，不影响存储器

陷阱：

illegal instruction //非法指令（只有当 i=1 时，SWAPA 才有引起）

privileged_instruction //特权指令（只有 SWAPA 才会引起）

mem_address_not_aligned //存储器地址未对齐

data_access_exception //数据访问异常

data_access_error //数据访问错误

data_store_error //数据存储错误

H.2.7 SETHI 指令

操作码	*操作*
SETHI	设置高22位比特

汇编语言语法
sethi *const22 ,reg*$_{rd}$ sethi %hi(*value*),*reg*$_{rd}$

例如：

将立即数 0x1fffff 送入%o0： sethi 0x1fffff,%o0

数据多于 22 位时： sethi %hi(0xffffffff),%o1

说明：

SETHI 把 r[rd]的最低 10 位清零，把 22 位常数（const22）送入 r[rd]的高 22 位。当源操作数使用%hi（value）格式时，仅取 value 的高 22 位送入 r[rd]，此时，r[rd]的低 10 位也被清零。

SETHI 不影响条件码。

陷阱：（无）

H.2.8 NOP 指令

操作码	*操作*
NOP	空操作

汇编语言语法
nop

例如：

由于跳转指令的延迟特性，在其后面跟一条空指令：

```
1：     cmp     %o0,%o1
        be      1b
        nop
```

说明:

除了程序计数器 PC 和 nPC 各自加一，NOP 指令不改变任何可见的程序状态。

陷阱：（无）

H.2.9 逻辑指令

操作码	*操作*
AND	与
ANDcc	与，改变 *icc*
ANDN	先非后与
ANDNcc	先非后与，改变 *icc*
OR	或
ORcc	或，改变 *icc*
ORN	先非后或
ORNcc	先非后或，改变 icc
XOR	异或
XORcc	异或，改变 icc
XNOR	先非后异或
XNORcc	先非后异或，改变 icc

汇编语言语法

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$_{rd}$*

例如：

%l0 与 %l1 = %o0　　%l0 与 （非 15） = %o0

and %l0,%l1,%o0

addn %l0,15,%o0

说明：

这些指令实现按位逻辑运算。如果 i 字段是 0，则计算“r[rs1] 逻辑运算 r[rs2]”；如果 i 字段是 1，计算“r[rs1] 逻辑运算 sign_ext(simm13)”并把结果写入 r[rd]。

ANDcc, ANDNcc, ORcc, ORNcc, XORcc,和 XNORcc 修改 icc。条件如下：

n 位 根据 r[rd]<31>判断。结果为负，n = 1；非负，n = 0

z 位 结果为 0，z = 1；不为 0，z = 0

v 位 不影响，恒为 0

c 位 不影响，恒为 0

特别注意：ANDN、ANDNcc、ORN、ORNcc、XNOR 和 XNORcc 先对第二个操作数逻辑求非，再进行主操作(AND、OR 或 XOR)。

陷阱：（无）

H.2.10 移位指令

操作码	*操作*
SLL	逻辑左移
SRL	逻辑右移
SRA	算数右移

汇编语言语法
sll reg_{rs1} ,*reg_or_imm* ,reg_{rd}
srl reg_{rs1} ,*reg_or_imm* ,reg_{rd}
sra reg_{rs1} ,*reg_or_imm* ,reg_{rd}

例如：

%l0 中存储的数据逻辑左移 4 位： sll %l0,4,%o0

如果 4 存储在寄存器%l1 中： sll %l0,%l1,%o0

说明：

对于位移的位数，如果 i 字段是 0，是 r[rs2]中数据的最低 5 位的值，高 27 位被忽略；如果 i 字段是 1，则是立即数的低 5 位的值，其余高位被忽略。

SLL 左移 r[rs1]由移位值所给定的位数。SRL 和 SRA 右移 r[rs1]由移位数所指定的位数。SLL 和 SRL 用 0 填充腾出的空位，而 SRA 用 r[rs1]的最高有效位填充腾出的空位。移位数量为 0 时不移位。这些指令均把移位后的结果写入 r[rd]，并且不改变条件码。

陷阱： (无)

H.2.11 加法指令

操作码	*操作*
ADD	加
ADDcc	加，并改变 icc
ADDX	加上进位位的加
ADDXcc	加上进位位的加，并改变 icc

汇编语言语法
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}

例如：

%l0 + %l1 = %o0 %l0 + 15 = %o0

add %l0,%l1,%o0

add %l0,15,%o0

说明：

如果 i 字段是 0，计算“r[rs1] + r[rs2]”；如果 i 字段是 1，则计算“r[rs1] + sign_ext(simm13)”，并把和写入 r[rd]。当 r[rs2]是立即数时，只能是等于或小于 13 位的立即数；若大于 13 位，则需向将两个源操作数均送入不同的寄存器之后，再运算。

在运算的过程中，两个源操作数和结果均作为补码参与运算。（2006-8-8 16:41:45）

ADDX 和 ADDXcc (“扩展加”) 在计算两个源操作数的同时，加上 PSR 的进位位(c)；也就是说，它们计算“r[rs1] + r[rs2] + c”或者“r[rs1] + sign_ext(simm13) + c”并把和写入 r[rd]。

ADDcc 和 ADDXcc 修改整数状态位(icc)，条件如下：

n 位　计算结果为负数时，n = 1；非负时，n = 0

z 位　计算结果为零时，z = 1；非零时，z = 0

v 位　当 r[rs1]和 r[rs2]符号相同而与 r[rd]符号不同时，v = 1；不满足此条件时，v = 0。CPU 仅以三个操作数的最高位（31 位）判断数据的符号。

c 位　r[rd]第 31 位出现进位时，c = 1；不发生进位时，c = 0

陷阱：（无）

H.2.12 带标志的加法指令

操作码	*操作*
TADDcc TADDccTV	带标志的加，并改变 icc 带标志的加，改变 icc，同时引起溢出陷阱

汇编语言语法
taddcc　　reg_{rs1} ,*reg_or_imm* ,reg_{rd} taddcctv　reg_{rs1} ,*reg_or_imm* ,reg_{rd}

例如：

```
%l0 + %l1 = %o0                    %l0 + 15 = %o0
taddcc   %l0,%l1,%o0                   taddcctv   %l0,15,%o0
```

说明:

如果 i 字段是 0，计算“r[rs1] + r[rs2]”；如果 i 字段是 1，则计算“r[rs1] + sign_ext(simm13)”，并把和写入 r[rd]。

TADDcc 和 TADDccTV 指令没有引起陷阱事件时，计算结果同时修改 icc。条件如下：

n 位　计算结果为负数时，n = 1；非负时，n = 0

z 位　计算结果为零时，z = 1；非零时，z = 0

v 位　当 r[rs1]和 r[rs2]符号相同而与 r[rd]符号不同时，v = 1；不满足此条件时，v = 0。CPU 仅以三个操作数的最高位（31 位）判断数据的符号。

c 位　r[rd]第 31 位出现进位时，c = 1；不发生进位时，c = 0

tag_overflow 事件产生条件：任何一个操作数的 tag 段（即第 0 位或第 1 位）非零，或者，该加法产生了算术溢出（两个操作数符号相同而与计算结果符号不同）。

如果 TADDccTV 指令引起了 tag_overflow 事件，就会进入 tag_overflow 陷阱，此时 r[rd]和状态位（icc）保持不变。如果 TADDccTV 指令不引起 tag_overflow 事件，那么 icc 被更新（特别是溢出位，v = 0）并且和被写入 r[rd]。

如果 TADDcc 引起了 tag_overflow 事件，PSR 的溢出位(v)置位；如果没有引起 tag_overflow 事件，溢出位被清空。不管在哪种情况下，其他整数状态位均被更新并且和写入 r[rd]。

陷阱：

tag_overflow (TADDccTV only)　//标志溢出（只有 TADDccTV 才能引起并进入该陷阱；TADDcc 并不能进入该陷阱）

H.2.13 减法指令

操作码	*操作*
SUB	减
SUBcc	减，并改变 icc
SUBX	减去进位位的减
SUBXcc	减去进位位的减，并改变 icc

汇编语言语法	
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}

例如：

%l0 - %l1 = %o0　　%l0 - 15 = %o0

sub　%l0,%l1,%o0　　sub　%l0,15,%o0

说明：

如果 i 字段是 0，计算“r[rs1]－r[rs2]”；如果 i 字段是 1，则计算“r[rs1]-sign_ext(simm13)”并把差写入 r[rd]。当 r[rs2]是立即数时，只能是等于或小于 13 位的立即数；若大于 13 位，则需向将两个源操作数均送入不同的寄存器之后，再运算。

在运算的过程中，两个源操作数和结果均作为补码参与运算。（2006-8-8 16:41:45）

SUBX 和 SUBXcc (“扩展减”)同时减去 PSR 的进位位(c)；也就是说计算“r[rs1]－r[rs2]－c”或者“r[rs1]－sign_ext(simm13)－c”且把差写入 r[rd]。

SUBcc 和 SUBXcc 修改整数状态位(icc)，条件如下：

n 位　计算结果为负数时，n = 1；非负时，n = 0

z 位　计算结果为零时，z = 1；非零时，z = 0

v 位　当 r[rs2]与 r[rd]符号相同而与 r[rs1]符号不同时，v = 1；不满足此条件时，v = 0。CPU 仅以三个操作数的最高位（31 位）判断数据的符号

c 位　运算时，向 r[rs1]第 31 位借位或 r[rs1]第 31 位需要借位，c = 1；不发生借位时，c = 0

陷阱：（无）

H.2.14 带标志的减法

操作码	*操作*
TSUBcc	带标志的减，并改变 icc
TSUBccTV	带标志的减，改变 icc，同时引起溢出陷阱

汇编语言语法
tsubcc reg_{rs1} ,*reg_or_imm* ,reg_{rd} tsubcctv reg_{rs1} ,*reg_or_imm* ,reg_{rd}

例如：

```
%l0 - %l1 = %o0                    %l0 - 15 = %o0
tsubcc   %l0,%l1,%o0               tsubcc   %l0,15,%o0
```

说明：

如果 i 字段是 0，计算“r[rs1]r[rs2]”；如果 i 字段是 1，计算“r[rs1]-sign_ext(simm13)”，差写入 r[rd]。当 r[rs2]是立即数时，只能是等于或小于 13 位的立即数；若大于 13 位，则需先将两个源操作数均送入不同的寄存器之后，再运算。

TSUBcc 和 TSUBccTV 指令没有引起陷阱事件时，计算结果并修改 icc。条件如下：

n 位　计算结果为负数时，n = 1；非负时，n = 0

z 位　计算结果为零时，z = 1；非零时，z = 0

v 位　当 r[rs2]与 r[rd]符号相同而与 r[rs1]符号不同时，v = 1；不满足此条件时，v = 0。CPU 仅以三个操作数的最高位（31 位）判断数据的符号

c 位　运算时，向 r[rs1]第 31 位借位或 r[rs1]第 31 位需要借位，c = 1；不发生借位时，c = 0

引起 tag_overflow 事件的条件：任一操作数的 tag 段（即，第 1 位或第 0 位）非零，或者，如果减法产生了算术溢出（r[rs2]和 r[rd]符号相同而与 r[rs1]符号不同）。

如果 TSUBccTV 引起了 tag_overflow 事件，就产生一个 tag_overflow 陷阱，r[rs1]和 icc 保持不变。如果 TSUBccTV 没有导致 tag_overflow 事件，更新 icc（特别是溢出位，v = 0）同时差被写入 r[rd]。

如果 TSUBcc 引起了 tag_overflow 事件，设置 PSR 的溢出位(v)；如果没有引起 tab_overflow 事件，溢出位被清空。不管上述哪种情况，icc 的其他标志位被更新，同时差写入 r[rd]。

陷阱：

tag_overflow (TSUBccTV only)　//标志溢出（只有 TSUBccTV 会引起并进入该陷阱；TSUBcc 只能引起 tag_overflow 事件，但不能进入该陷阱）

H.2.15 乘法单步指令

操作码	*操作*
MULScc	乘法单步，并修改 *icc*

汇编语言语法
mulscc *reg$_{rs1}$,reg_or_imm ,reg$_{rd}$*

例如：

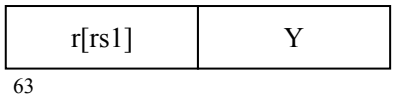
需要单步执行一个 3 位×3 位(二进制)的乘法：

```
set     0x5,%o0
mov     0x5,%y
set     0x5,%o1
mulscc %o0,%o1,%o0
```

说明：

MULScc 执行一位乘法，多次连续使用该指令可以组合出 n×n 位乘法（1 ≤ n ≤ 32）。该指令把 r[rs1]和 Y 寄存器看作一个单独的 64 位可右移双字寄存器。r[rs1]的最低有效位被看作好像和 Y 寄存器的最高有效位相连。MULScc 指令在 Y 的最低有效位的基础上，有条件地加。

r[rs1]和 Y 的组合形式如图所示：



初始时 r[rs1]应置零，r[rs2]存放被乘数，Y 存放乘数；指令执行结束后，乘积的高有效位存放在 r[rs1]中，乘积的低有效位存放在 Y 中。通常，MULScc 指令有 rs1 = rd。

MULScc 运算过程如下：

(1) 如果 i 字段为 0，乘数是 r[rs2]；如果 i 字段为 1，乘数是 sign_ext(simm13)。

(2) 把 r[rs1]右移一位，用 PSR.icc 的 N 和 V 位做异或运算，所得值填充 r[rs1]高位，得到一个 32 位的值。(为先前的部分积作个合适的标记)此时，定义该阶段结果为操作数 1。

(3) 如果 Y 寄存器的最低有效位等于 1，从(2)所移得的位移值和乘

数相加。

如果 Y 寄存器的最低有效位等于 0，从(2)所移得的位移值与 0 相加。此时，定义乘数为操作数 2；相加结果定义为操作结果。

(4) 第(3)步所得的和写入 r[rd]。

(5) 根据第(3)步执行的加法的结果更新 icc。

(6) Y 寄存器右移一位，用未移位时 r[rs1]的最低有效位取代 Y 的最高有效位。

指令对 icc 的影响如下：

N 位　　r[rd]的符号，即 n = r[rd]〈31〉

Z 位　　如果 r[rd]是 0，z = 1；如果 r[rd]非 0，z = 0

V 位　　操作数 1 和操作数 2 符号相同，而与操作结果符号不同时，v = 1；否则，v = 0

C 位　　操作数 1 和操作数 2 符号位（最高位）均为 1，或者，操作数 1 或操作数 2 符号位（最高位）为 1 而操作结果符号位（最高位）为 0 时，c = 1；否则，c = 0

注意：该指令应连续使用，不能在 n 位乘法的中间某一步加插其他指令，否则结果有问题(原因：3802 regfile EDAC 做的不完全)

陷阱：（无）

H.2.16 乘法指令

操作码	*操作*
UMUL	无符号整数乘
SMUL	有符号整数乘
UMULcc	无符号整数乘，改变 *icc*
SMULcc	有符号整数乘，改变 *icc*

汇编语言语法	
umul	*regrs1 ,reg_or_imm ,regrd*
smul	*regrs1 ,reg_or_imm ,regrd*
umulcc	*regrs1 ,reg_or_imm ,regrd*
smulcc	*regrs1 ,reg_or_imm ,regrd*

例如：

%l0 × %l1 = %o0　　%l0 × 15 = %o0

smul %l0,%l1,%o0

umul %l0,15,%o0

说明：

乘法指令执行 32 位与 32 位的乘法，产生 64 位值。如果 i 字段为 0，计算“r[rs1]×r[rs2]”；如果 i 字段为 1，计算“r[rs1]×sign_ext(simm13)”。乘积的高 32 位写入 Y 寄存器，低 32 位写入 r[rd]。

无符号数乘法指令(UMUL, UMULcc)假定操作数为无符号整数，乘积为无符号整数双字。带符号乘法指令(SMUL, SMULcc)假定操作数为带符号数，乘积为带符号整数双字。如果在使用 UMUL 和 UMULcc 指令时，源操作数是负数（如，-125），则认为该操作数的补码（-125 的补码为 0xffffff83）为一个无符号数，并使用其进行计算。

UMUL 和 SMUL 不影响 icc。

UMULcc 和 SMULcc 如下表所示影响 icc。n、z 是根据 r[rd]的最高位变化，并非是整个乘积的最高位。

cc	UMULcc	SMULcc
	如果乘积第[31]位为1，=1 如果乘积第[31:0]位为0，=1 0 0	如果乘积第[31]位为1，=1 如果乘积第[31:0]位为0，=1 0 0

执行 UMUL/UMULcc 之后，目的寄存器（r[rd]）32-bit 溢出通过 Y != 0 来说明。

执行 SMUL/SMULcc 之后，目的寄存器（r[rd]）32-bit 溢出通过 Y != (r[rd] >>31)说明。

陷阱： (无)

H.2.17 除法指令

操作码	*操作*
UDIV	无符号整数除
SDIV	有符号整数除
UDIVcc	无符号整数除，改变*icc*
SDIVcc	有符号整数除，改变*icc*

汇编语言语法	
udiv	*regrs1 ,reg_or_imm ,regrd*
sdiv	*regrs1 ,reg_or_imm ,regrd*
udivcc	*regrs1 ,reg_or_imm ,regrd*
sdivcc	*regrs1 ,reg_or_imm ,regrd*

例如：

%l0 ÷ %l1 = %o0　　%l0 ÷ 15 = %o0

sdiv %l0,%l1,%o0

udiv %l0,15,%o0

说明：

除法指令执行 64 位数除以 32 位数运算，产生 32 位的商。如果 i 字段为 0，计算“(Y □ r[rs1])÷r[rs2]”；如果 i 字段为 1，计算“(Y □ r[rs1]) ÷ sign_ext(simm13)”。整数商写入 r[rd]，余数（如果有）被抛弃。

无符号除法指令(UDIV, UDIVcc)假定被除数(Y□ r[rs1])为无符号整数双字，除数(r[rs2])为无符号整数字，商(r[rd])为无符号整数字。带符号除法指令(SDIV, SDIVcc) 假定被除数(Y□r[rs1])为带符号整数双字，除数(r[rs2]或 sign_ext(simm13))为带符号整数字，商数(r[rd])为带符号整数字。

如果有非零的余数，带符号除法将对不精确的商向 0 舍入；例如，~3÷2 等于~1 余~1 (而不是~2 余 1)。

除法指令的结果在某种情况下可以让 32 位的目的寄存器（r[rd]）溢出。当溢出发生时，在 r[rd]中返回适合的最大整数作为商。发生溢出的条件和对应条件下 r[rd]中的返回值如下表所示：

除法溢出检测及返回值		
指令	溢出发生条件	r[rd]中的返回值
UDIV, UDIVcc	结果 $>2^{32}-1$	$2^{32}-1$ (0xffffffff)
SDIV, SDIVcc (为正数结果时)	结果 $>2^{31}-1$	$2^{31}-1$ (0x7fffffff)
SDIV, SDIVcc (为负数结果时)	结果 $<$ (-2^{31})	-2^{31} (0x80000000)

UDIV 和 SDIV 不影响 icc。

UDIVcc 和 SDIVcc 如下表所示影响 icc。

icc	UDIVcc	SDIVcc
N	如果商（r[rd]）第[31]位为1，= 1	如果商（r[rd]）第[31]位为1，= 1
Z	如果商（r[rd]）第[31:0]位为0，= 1	如果商（r[rd]）第[31:0]位为0，= 1
V	如果溢出，= 1	如果溢出，= 1
C	0	0

陷阱:

division_by_zero　　//被零除

H.2.18 保存和恢复指令

操作码	***op3***	***操作***
SAVE	111100	保存调用程序的窗口
RESTORE	111101	恢复调用程序的窗口

汇编语言语法
save　　reg_{rs1} ,*reg_or_imm* ,reg_{rd}
restore reg_{rs1} ,*reg_or_imm* ,reg_{rd}

例如：

当程序要进入一个新的子程序时，为使子程序使用新的窗口和堆栈，通常会在子程序开头写：

```
save      %sp, -112, %sp
```

当子程序返回到主程序使用的寄存器窗口时：

```
restore
```

说明：

保存指令把 CWP 减 1(模 N 运算)，再把这个新值（new_CWP）和窗口无效标志（WIM）寄存器比较。如果与新 CWP 值相应的 WIM 位是 1，即(WIM ∩ 2new_CWP) = 1，那么产生一个 window_overflow 陷阱。如果与新 CWP 值相应 WIM 位是 0，则不会引起 window_overflow 陷阱，且新 CWP 值写入 CWP 中。这使 CWP̃1 窗口成为当前窗口，因此就保存了调用程序的窗口。

恢复指令把 CWP 加 1(模 N 运算)，再把这个新值(new_CWP)和窗口无效标志(WIM)寄存器比较。如果与新 CWP 值相应的 WIM 位是 1，即(WIM ∩ 2new_CWP) = 1，那么产生一个 window_underflow 陷阱。如果与新 CWP

值相应的 WIM 位是 0，则不会引起 window_underflow 陷阱，且新 CWP 值被写入 CWP 中。这使 CWP-1 窗口成为当前窗口，从而恢复调用程序的窗口。

此外，当且仅当没有产生上溢或者下溢陷阱时，保存和恢复指令的行为类似普通的 ADD 指令，不同之处仅在于源操作数 r[rs1]和/或 r[rs2]是从旧的窗口中读出的(即原来 CWP 标志的窗口)，而和写入新窗口的 r[rd]中（即新 CWP 值标志的窗口）。

陷阱：

window_overflow　　//窗口上溢（只有保存才能引起）

window_underflow　//窗口下溢（只有恢复才能引起）

H.2.19 根据整数状态位跳转的指令

操作码	*cond*	*操作*	*检测 icc*
BA	1000	总是跳转	1
BN	0000	从不跳转	0
BNE	1001	不等则转	**not** Z
BE	0001	等则转	Z
BG	1010	大于则转	**not** (Z or (N **xor** V))
BLE	0010	小于或等于则转	Z **or** (N **xor** V)
BGE	1011	大于或等于则转	**not** (N **xor** V)
BL	0011	小于则转	N **xor** V
BGU	1100	无符号比较，大于则转	**not** (C **or** Z)
BLEU	0100	无符号比较，小于或等于则转	(C **or** Z)
BCC	1101	清进位位则转 (无符号数比较，大于或等于则转)	**not** C
BCS	0101	进位位置位则转 (无符号数比较，小于则转)	C
BPOS	1110	为正则转	**not** N
BNEG	0110	为负则转	N
BVC	1111	清溢出位则转	**not** V
BVS	0111	溢出位置位则转	V

汇编语言语法		
ba{,a}	*label*	
bn{,a}	*label*	
bne{,a}	*label*	*(同义词:* bnz*)*
be{,a}	*label*	*(同义词:* bz*)*
bg{,a}	*label*	
ble{,a}	*label*	
bge{,a}	*label*	
bl{,a}	*label*	
bgu{,a}	*label*	

bleu{,a}	*label*	
bcc{,a}	*label*	*(同义词:* bgeu*)*
bcs{,a}	*label*	*(同义词:* blu*)*
bpos{,a}	*label*	
bneg{,a}	*label*	
bvc{,a}	*label*	
bvs{,a}	*label*	

注：上述指令均设置了“取消”位，即在操作码助记符后添加一个“，a”。

如：“bgu,a　label”。表中括号{}括起来的“,a” 是可选的。

例如：

在寄存器%o0 和%o1 中的两个数相减，差为正则跳转，为负则继续执行下面的指令：

```
1:  subcc  %o0,%o1,%o2
    bpos       1b
    nop
```

两数相减，结果为负，紧接着跳转指令的下一条指令被跳转指令取消：

```
1: subcc  %o0,%o1,%o2
          bpos，a   1b
          set     111，%o2     ！这条语句不执行，被直接跳过
```

说明：

无条件跳转 (BA, BN)

如果不带取消位，BN 指令像“NOP”一样动作。如果带取消位，紧接着的指令被取消（不执行），即，直接执行被取消指令的后续指令。

无论 icc 为何值，BA 指令均引起一个与 PC 相关的，到地址“PC + (4 □. sign_ext(disp22))”的延迟控制跳转。如果带取消位，紧接着的指令被取消（不执行）。如果不带取消位，执行紧接着的指令。

icc-条件跳转

有条件的 Bicc 指令(除了 BA 和 BN 以外其余指令)根据 cond 字段判 icc。这种判断产生一个“真”或“假”的结果。如果为“真”，进行跳转；即，指令产生一个指向地址“PC + (4 □. sign_ext(disp22))”的延迟控制跳

转。如果为“假”，不跳转。

当跳转条件为真时，无论指令是否带取消位，总是执行紧接着跳转指令的指令。如果跳转条件为假且跳转指令带取消位，那么紧接着的指令被取消（不执行）。

注意：无条件跳转指令只要带取消位就会取消执行紧接着的指令，而 icc 条件跳转指令必须同时具备跳转条件为假和带取消位两个条件，才会取消执行紧接着的指令。

陷阱：(无)

H.2.20 根据浮点状态位跳转的指令

操作码	*cond*	*操作*	*检测 fcc*
FBA	1000	总是跳转	1
FBN	0000	从不跳转	0
FBU	0111	无序则转	U
FBG	0110	大于则转	G
FBUG	0101	无序或大于则转	G **or** U
FBL	0100	小于则转	L
FBUL	0011	无序或小于则转	L **or** U
FBLG	0010	小于或大于侧转	L **or** G
FBNE	0001	不等则转	L **or** G **or** U
FBE	1001	相等则转	E
FBUE	1010	无序或相等则转	E **or** U
FBGE	1011	大于或等于侧转	E **or** G
FBUGE	1100	无序或大于或等于则转	E **or** G **or** U
FBLE	1101	小于或等于则转	E **or** L
FBULE	1110	无序或小于或等于则转	E **or** L **or** U
FBO	1111	有序则转	E **or** L **or** G

汇编语言语法		
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*	*(同义词:* fbnz*)*
fbe{,a}	*label*	*(同义词:* fbz*)*
fbue{,a}	*label*	
fbge{,a}	*label*	

fbuge{,a}	*label*
fble{,a}	*label*
fbule{,a}	*label*
fbo{,a}	*label*

注：为根据浮点状态位指令（FBcc）设置了“取消”位，在操作码助记符后添加一个“,a”。例如：“fb1,a　label” 。上述表中括号{}括起来的“,a” 是可选的。

说明：

无条件跳转 (FBA, FBN)

如果它的取消位是 0，一个 FBN 指令像“NOP”一样动作。如果它的取消位为 1，接下来的（延迟）指令不执行。不管哪种情况，控制上的转移都不会发生。

无论浮点状态位的值为何，FBA 引起一个与 PC 相关的，到地址“PC + (4 □. sign_ext(disp22))” 的延迟控制转移。如果跳转指令的取消位是 1，延迟指令不执行。如果取消位是 0，延迟指令执行。

Fcc-条件跳转

条件 FBfcc 指令(除了 FBA 和 FBN 以外其余指令)根据 cond 字段判断浮点数条件码 (fcc)。这种判断产生一个“真”或“假”的结果。如果为“真”，进行跳转，即，指令产生一个与 PC 相对的指向地址“PC + (4 □. sign_ext(disp22))”处的延迟控制转移。如果为“假”，不跳转。

如果一个跳转条件为真，无论取消字段的为何值，延迟指令总是被执行。如果跳转条件为假，且 a（取消位）字段为 1，延迟指令不执行。（注意取消位在条件跳转和无条件跳转下有不同的效果。）

如果 PSR 的 EF 位是 0，一条 FBfcc 指令将不会跳转，也不取消接下来的指令，同时产生一个 fp_disabled 陷阱。

如果 FBfcc 指令前面紧邻执行的是一条 Fpop2 指令，FBfcc 的结果是不明确的。因此，至少一条非 Fpop2 指令应该在 Fpop2 和后来的 FBfcc 指令之间执行。如果紧跟着一条 LDFSR 指令的三条指令中的任一条是 FBfcc，FBfcc 所见的 FSR 的 fcc 字段的值是未定义的。

陷阱：

fp_disabled　　//fp 禁用

fp_exception　　//fp 异常

H.2.21 调用并链接指令

操作码	*操作*
CALL	调用并链接

汇编语言语法
call　*label*

例如：

在主程序中有函数，main（）。在其他文件中调用该函数：

```
call   main
nop   !因为 call 是延迟执行的，所以后面增加一条空操作
```

说明：

CALL 指令产生指向地址“PC + (4 □. disp30)”（即 label）处的控制转移，这一转移是无条件的，延迟的，与 PC 相关的。因为位移字段(disp30)为 30 位宽，所以目标地址可以任意远。CALL 指令把 PC 的值，其中包含着 CALL 指令的地址，写入 r[15] (%o7)。

陷阱：(无)

H.2.22 转移并链接指令

操作码	*操作*
JMPL	转移并链接

汇编语言语法
jmpl　*address* ,reg_{rd}

例如：

跳转到目标地址 0x40000000：

```
        set     0x40000000,%o0
        jmpl    %o0,%o1      ! jmpl 指令的地址写入%o1
```

说明：

JMPL 指令引起一个延迟控制跳转，跳转的目的地址存放在寄存器中。如果 i 字段是 0，转移地址由“r[rs1] + r[rs2]”给定；如果 i 是 1，转移地址由“r[rs1] + sign_ext(simm13)”给定。

JMPL 指令把 PC 值，即 JMPL 指令本身的地址，复制到 r[rd]。

如果跳转地址的最低两位中任一位非 0，将引起一个 mem_address_not_aligned 陷阱。

rd = 15(%o7)时，JMPL 指令像 CALL 指令那样工作。rd=0（%g0）时，JMPL 指令被用来从子程序返回。如果使用 CALL 指令进入非叶子程序(在子程序开始处使用 SAVE 指令)，那么典型的返回地址是“r[31]+8”；如果进入的是叶子程序（在子程序开始处不使用 SAVE 指令），那么典型的返回地址为“r[15]+8”。

陷阱：

mem_address_not_aligned　　//存储器地址未对齐

H.2.23 从陷阱返回指令

操作码	*操作*
RETT†	从陷阱返回

† 特权指令

汇编语言语法
rett　*address*

例如：

从异常处理程序中返回：

```
jmpl    %l1,%g0
rett    %l2
```

说明：

RETT 用于从陷阱句柄中返回。在一些情况下，RETT 自己可能引起一个陷阱。如果一个 RETT 指令没有引起陷阱，它（1）把 CWP 加 1（模 NWINDOWS）（2）引起一个到目标地址的延迟控制转移，（3）从 PS 字段中恢复 PSR 的 S 字段，（4）把 PSR 的 ET 字段设置为 1。如果 i 字段是 0，目标地址为“r[rs1] + r[rs2]”；如果 i 字段为 1，目标地址为“r[rs1] + sign_ext(simm13)”。

当 RETT 被执行时，可能发生下面几个陷阱中的一个。这些陷阱按照优先级由高到低排列如下：

若使能陷阱 (ET=1)且处理机处于用户模式(S=0)，触发

privileged_instruction 陷阱。

若使能陷阱 (ET=1)且处理机处于管理模式(S=1)，触发 illegal_instruction 陷阱。

如果陷阱禁用(ET=0)且（a）处理机处于用户模式(S=0)，或者（b）检测到窗口下溢状态（WIM 与 2new_CWP = 1）或者（c）目标地址的最低两位中任一位非零，那么处理机在 TBR 寄存器的 tt 字段分别指出陷阱（a）privileged_instruction，（b）window_underflow，（c）mem_address_not_aligned 而且进入 error_mode 状态。

在 RETT 之前立即执行的指令必须是一个 JMPL 指令。（如果不是，RETT 后一条或更多的指令可能访问到一个不正确的地址空间。）

当从陷阱句柄返回时，使用下面的程序重新执行进入陷阱的指令：

```
jmpl  %l1,%g0       ! old PC
rett        %l2             ! old nPC
```

在进入陷阱的指令之后，使用下面的程序向指令返回：

```
jmpl        %l2,%r0       ! old nPC
rett        %l2+4             ! old nPC + 4
```

陷阱：

```
illegal_instruction                 //非法指令
privileged_instruction              //特权指令
privileged_instruction      //特权指令（可能导致 CPU 进入错误模式）
mem_address_not_aligned //存储器地址未对齐（可能导致 CPU 进入错误模式）
window_underflow      //窗口下溢（可能导致 CPU 进入错误模式）
```

H.2.24 根据整数状态位进入陷阱的指令

操作码	*操作*	*检测 icc*
TA	总是进入陷阱	1
TN	从不进入陷阱	0
TNE	不等则进入陷阱	**not** Z
TE	等则进入陷阱	Z
TG	大于则进入陷阱	**not** (Z **or** (N **xor** V))
TLE	小于或等于则进入陷阱	Z **or** (N **xor** V)
TGE	大于或等于则进入陷阱	**not** (N **xor** V)

TL	小于或等于则进入陷阱	N **xor** V
TGU	无符号数比较，大于则进入陷阱	**not** (C **or** Z)
TLEU	无符号数比较，小于或等于则进入陷阱	(C **or** Z)
TCC	清进位位则进入陷阱(无符号数比较，大于或等于则转)	**not** C
TCS	进位位置位则进入陷阱（无符号数据比较，小于则转）	C
TPOS	为正则进入陷阱	**not** N
TNEG	为负则进入陷阱	N
TVC	清溢出位则进入陷阱	**not** V
TVS	溢出位置位则进入陷阱	V

汇编语言语法		
ta	*software_trap#*	
tn	*software_trap#*	
tne	*software_trap#*	*(等价指令:* tnz*)*
te	*software_trap#*	*(等价指令:* tz*)*
tg	*software_trap#*	
tle	*software_trap#*	
tge	*software_trap#*	
tl	*software_trap#*	
tgu	*software_trap#*	
tleu	*software_trap#*	
tcc	*software_trap#*	*(等价指令:* tgeu*)*
tcs	*software_trap#*	*(等价指令:* tlu*)*
tpos	*software_trap#*	
tneg	*software_trap#*	
tvc	*software_trap#*	
tvs	*software_trap#*	

例如：

直接给出陷阱号：　　ta　4

如果陷阱号 4 存储在寄存器%l0 中：ta　%l0

说明

软件陷阱指令（Ticc）根据指令 cond 字段不同取值，按照上表中所示规则计算当前 icc，产生一个“真”或“假”的结果。如果结果为“真”且后面没有更高优先级的异常和中断请求待处理，那么产生一个 trap_instruction 陷阱。如果结果为“假”，不触发 trap_instruction 陷阱，指令如同 NOP 指令一样动作。Ticc 指令均为延迟指令，可在该指令的下一条加 nop 指令或向陷阱处理程序传递参数的指令。

当产生 trap_instruction 陷阱时，若 i 字段为 0，陷阱基址寄存器(TBR)

的 tt 字段写入 128 加上“r[rs1] + r[rs2]”最低 7 位的和；若 i 字段为 1，则写入 128 加上“r[rs1] + sign_ext(software_trap#)”最低 7 位的和。因为有 128 个软件陷阱，所以 software_trap#的范围为 0 到 127。

执行 Ticc 之前，应使能陷阱（PSR 中 ET = 1）；若没有使能陷阱，则 CPU 进入错误模式。执行 Ticc 之后，处理机进入管理模式，陷阱禁用，CWP-1（模 NWINDOWS），PC 和 nPC 存入新窗口的 r[17]和 r[18] (%l1 和%l2)。

Ticc 可被用来实现断点，跟踪和调入管理软件。也可以被用来进行运行时检查，比如数组索引越界，整数溢出等。

陷阱：

trap_instruction　　//陷阱指令

H.2.25 读状态寄存器指令

操作码	*操作*
RDY	读 Y 寄存器
RDASR‡	读辅助状态寄存器
RDPSR†	读处理器状态寄存器
RDWIM†	读窗口无效标志寄存器
RDTBR†	读异常基址寄存器

† 特权指令

‡（如果源寄存器是有特权的，则为）特权指令

汇编语言语法
rd　%y, reg_{rd}
rd　asr_reg_{rs1} ,reg_{rd}
rd　%psr,reg_{rd}
rd　%wim,reg_{rd}
rd　%tbr,reg_{rd}

例如：

将处理器状态寄存器中的数据读入到%o0 中

rd　%psr,%o0

说明：

此指令用于读取指定的 IU 状态寄存器，结果存入 r[rd]。

在 BM3803 中，只使用了两个辅助状态寄存器，分别是%asr16 和%asr17。这两个辅助状态寄存器的作用详见《BM3803 用户手册》。

陷阱：

privileged_instruction//特权指令（RDY 不会引起该陷阱）

illegal_instruction　　//非法指令（只有 RDASR 可能引起）

H.2.26 写状态寄存器指令

操作码	*操作*
WRY	写 Y 寄存器
WRASR‡	写辅助状态寄存器
WRPSR†	写处理器状态寄存器
WRWIM†	写窗口无效标志寄存器
WRTBR†	写异常基址寄存器

† 特权指令

‡（如果源寄存器为特权指令，则为）特权指令

汇编语言语法
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

例如：

往 Y 寄存器中写入 0xffffffff：

```
set 0xffffffff,%o0
wr %o0,%g0,%y
```

说明：

对于 WRY, WRPSR, WRWIM 和 WRTBR 指令，如果 i 字段为 0，把“r[rs1] xor r[rs2]”的值写入指定 IU 状态寄存器的可写字段中；如果是 1，则写入“r[rs1] xor sign_ext(simm13)”。（注意：是 xor 操作）

WRASR 向由 rd 指定的辅助状态寄存器(ASR)写入一个值。在 BM3803 中，只有%asr16 和%asr17 两个辅助状态寄存器。

如果一个 WRPSR 指令的结果引起 PSR 的 CWP 字段指向一个不能使用的窗口，它引起一个 illegal_instruction 陷阱且不写 PSR。

陷阱；

privileged_instruction //特权指令（WRY 不会引起）

illegal_instruction　//非法指令（当 CWP □ NWINDOWS 时，WRPSR

引起）

illegal_instruction　　//非法指令（WRASR 引起）

H.2.28 未实现的指令

操作码	*操作*
UNIMP	未实现

汇编语言语法
unimp　*const22*

例如：

当需要主动产生一个 illegal_instruction 陷阱时：

```
unimp          ！此时，const22 默认为 0
```

说明：

UNIMP 指令引起一个 illegal_instruction 陷阱。const22 的值被硬件忽略。

陷阱：

illegal_instruction　　//非法指令

H.2.29 刷新 cache 指令

操作码	*操作*
FLUSH	刷新指令存储器

汇编语言语法
flush　*address*

例如：

刷新 cache:

```
flush
```

说明：

FLUSH 指令刷新 cache。同时，保证在任何先于 FLUSH 指令提交的加载、存储和原子加载—存储指令在 FLUSH 指令刷新 cache 之前完成。

对于 FLUSH 指令有效的虚地址，如果 i 字段是 0，为“r[rs1] + r[rs2]”；如果 i 字段是 1，为“r[rs1] + sign_ext(simm13)”。

注意：在 BM3803 中，flush 指令与 SPARC V8 标准中的 flush 指令功能不完全相同。

陷阱：

unimplemented_FLUSH　　//未实现的 FLUSH

illegal_instruction　　//非法指令

H.2.30 整形转换为浮点型指令

操作码	*操作*
FiTOs	定点数转换为单精度浮点数
FiTOd	定点数转换为双精度浮点数

推荐的汇编语言语法	
fitos	*freg*$_{rs2}$, *freg*$_{rd}$
fitod	*freg*$_{rs2}$, *freg*$_{rd}$

说明：

这些指令把 f[rs2]中的 32-bit 的整型字操作数转换为目标格式的浮点数。把结果写入 rd 指定的 f 寄存器。

根据 FSR 的 RD 字段舍入。

陷阱：

fp_disabled

fp_exception (NX (FiTOs only), invalid_fp_register(FiTOd)

H.2.31 浮点数转换为整形指令

操作码	*操作*
FsTOi	单精度浮点数转为定点数
FdTOi	双精度浮点数转为定点数

推荐的汇编语言语法	
fstoi	*freg*$_{rs2}$, *freg*$_{rd}$
fdtoi	*freg*$_{rs2}$, *freg*$_{rd}$

说明：

这些指令把由 rs2 指定的 f 寄存器中的浮点操作数，转换为 f[rd]中一个 32-bit 的整型数。

结果会向零舍入。(FSR 寄存器的 RD 字段被忽略)。

陷阱：

fp_disabled　　//fp 禁用

fp_exception (NV, NX, invalid_fp_register(FdTOi, FqTOi)) //fp 异常

H.2.32 浮点格式之间转换指令

操作码	*操作*
FsTOd	单精度浮点数转为双精度浮点数
FdTOs	双精度浮点数转为单精度浮点数

推荐的汇编语言语法		
fstod	*freg*$_{rs2}$,	*freg*$_{rd}$
fdtos	*freg*$_{rs2}$,	*freg*$_{rd}$

说明：

这些指令把 rs2 指定的 f 寄存器中的浮点操作数，转换为目标格式的浮点数。把结果写入 rd 指定的 f 寄存器。

根据 FSR 的 RD 字段指定的舍入方式进行舍入。

FdTOs 可以引发 OF, UF 和 NX 异常，FsTOd 不会。

如果源操作数是一个有符号的 NaN，两条指令都可以触发一个 NV 异常。

陷阱：

fp_disabled //fp 禁用

fp_exception (OF, UF, NV, NX, invalid_fp_register) //fp 异常

H.2.33 浮点数基本操作指令

操作码	操作
FMOVs	浮点数寄存器间移动
FNEGs	浮点数求反
FABSs	浮点数求绝对值

推荐的汇编语言语法		
fmovs	*freg*$_{rs2}$,	*freg*$_{rd}$
fnegs	*freg*$_{rs2}$,	*freg*$_{rd}$
fabss	*freg*$_{rs2}$,	*freg*$_{rd}$

说明：

FMOVs 把 f[rs2]的内容拷贝到 f[rd]。

FNEGs 把 f[rs2]的内容拷贝到 f[rd] ，符号位求反。

FABSs 把 f[rs2] 的内容拷贝到 f[rd]中，但清除符号位。

这些指令不舍入。

在 f 寄存器之间转移多精度的值，每个字需要一次 FMOVs 指令的传

输。

如果源寄存器和目的寄存器(fregrs2 和 fregrd)是同一个，对 FNEGs、FABSs，执行一次可以完成单精度和双精度浮点数的取反或取绝对值操作。

如果源寄存器和目的寄存器是不同的，双精度浮点数的取反和取绝对值操作使用一条 FNEGs 或 FABSs 一条 FMOVs 指令实现。

陷阱：

fp_disabled ///fp 禁用

H.2.34 浮点平方根指令

操作码	*操作*
FSQRTs	单精度浮点数开平方
FSQRTd	双精度浮点数开平方

推荐的汇编语言语法		
fsqrts	*fregrs2*，	*fregrd*
fsqrtd	*fregrs2*，	*fregrd*

说明：

这些指令产生由 rs2 字段指定的 f 寄存器中的浮点操作数的平方根。结果放入由 rd 字段指定的目的 f 寄存器。

根据 FSR 的 rd 字段指定的舍入方式进行舍入。

陷阱：

fp_disabled //fp 禁用

fp_exception (NV, NX, invalid_fp_register(FSQRTd, FSQRTq))//fp 异常

H.2.35 浮点加减法指令

操作码	*操作*
FADDs	单精度浮点数加法
FADDd	双精度浮点数加法
FSUBs	单精度浮点数减法
FSUBd	双精度浮点数减法

推荐的汇编语言语法			
fadds	*fregrs1*，	*fregrs2*，	*fregrd*
faddd	*fregrs1*，	*fregrs2*，	*fregrd*
fsubs	*fregrs1*，	*fregrs2*，	*fregrd*

fsubd	$freg_{rs1}$,	$freg_{rs2}$,	$freg_{rd}$

说明：

浮点加指令把 rs1 字段指定的 f 寄存器和 rs2 字段指定的 f 寄存器加起来，把和写入由 rd 字段指定 f 寄存器内。

浮点减指令用从 rs1 字段指定的 f 寄存器减去由 rs2 字段指定的 f 寄存器，把差写入 rd 字段指定的 f 寄存器。

陷阱：

fp_disabled //fp 禁用

fp_exception (OF, UF, NX, NV) //fp 异常

invalid_fp_register(all except FADDs and FSUBs))//无效的 fp 寄存器

H.2.36 浮点乘除法指令

操作码	*操作*
FMULs FMULd	单精度浮点数乘法 双精度浮点数乘法
FsMULd	单精度浮点数乘法，结果为双精度浮点数
FDIVs FDIVd	单精度浮点数除法 双精度浮点数除法

推荐的汇编语言语法			
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	$freg_{rs1}$,	$freg_{rs2}$,	$freg_{rd}$

说明：

浮点乘法指令把由 rs1 字段指定的 f 寄存器乘以由 rs2 字段指定的 f 寄存器，并把乘积放入 rd 字段指定的 f 寄存器。

FsMULd 指令为两个单精度操作数提供精确的双精度乘积，没有下溢，上溢或者舍入错误。

浮点除法指令用 rs1 字段指定的 f 寄存器的值除以 rs2 字段指定的 f 寄存器，并把商数写入 rd 字段指定的 f 寄存器。

陷阱：

fp_disabled //fp 禁用

fp_exception (OF, UF, DZ (FDIV only), NV, NX, //fp 异常

invalid_fp_register(all except FMULs and FDIVs))//无效的 fp 寄存器

H.2.37 浮点数比较指令

操作码	操作
FCMPs FCMPd	单精度浮点数比较 双精度浮点数比较
FCMPEs FCMPEd	单精度浮点数比较，结果为无序时产生异常 双精度浮点数比较，结果为无序时产生异常

推荐的汇编语言语法		
fcmps	fregrs1 ,	fregrs2
fcmpd	fregrs1 ,	fregrs2
fcmpes	fregrs1 ,	fregrs2
fcmped	fregrs1 ,	fregrs2

说明：

这些指令比较由 rs1 字段所指定的 f 寄存器和由 rs2 字段所指定的 f 寄存器，且根据下表设置浮点条件码。

浮点条件码 (fcc)

fcc	*Relation*
0	*fregrs1 = fregrs2*
1	*fregrs1 < fregrs2*
2	*fregrs1 > fregrs2*
3	*fregrs1 ? fregrs2 (无序)*

如果任一操作数是有符号的 NaN 或者是一个正的 NaN，比较结果为无序，引起非法(NV)异常。如果任一操作数是有符号的 NaN，FCMP 引起一个非法（NV）异常。

一条浮点比较指令和一条 FBfcc 指令只见需要插入一条非浮点比较指令，这样 FBfcc 才可以正确分支，否则，FBfcc 的结果是不可预知的。

陷阱：

fp_disabled //fp 禁用

fp_exception (NV, invalid_fp_register(all except FCMPs and FCMPEs)) //fp 异常

H.3 BM3803 编程必选规则

H.3.1 寄存器窗口与函数调用

函数调用

函数调用的一般原则：

1、调用函数通过 CALL 指令使 PC 指针分支到被调函数的入口地址

2、被调函数使用 SAVE 指令获得新的寄存器窗口，并取得新的堆栈空间

3、被调函数使用“RET； RESTORE”指令组合返回调用函数。

例 1：调用函数：main（）；被调函数：printf（）

```
main：
        call printf                ！调用 printf（）
        nop                        ！延迟一条指令
        nop                        ！位置①
            :
            :
printf:
        save %sp, -1024, %sp       ！获得新寄存器窗口
                                   ！和新堆栈空间
            :
            :
        ret
        restore                    ！被调函数返回
                                   ！main（）位置①
```

参数传递

参数传递的一般规则：

1、调用函数将参数写入当前寄存器窗口的输出寄存器中（%o0~%o5）。如果参数多于 6 个，则从第 7 个参数开始写入堆栈。

2、调用函数使用 CALL 指令调用被调函数

3、被调函数使用 SAVE 获得新寄存器窗口和堆栈后，从新寄存器窗

口的输入寄存器（%i0~%i5）取得参数。如果参数多于 6 个，则从调用函数的堆栈取得后面的参数。

4、被调函数将返回值放入输入寄存器。返回调用函数后，调用函数从输出寄存器中取得返回值。

例 2：（例中黑体字标注了参数传递）

```
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;
}
```

以上两函数编译后生成：

```
<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 ]     ! 参数 c7，堆栈传递
ld    [ %fp + -40 ], %o0
st    %o0, [ %sp + 0x60 ]     ! 参数 c8，堆栈传递
ld    [ %fp + -12 ], %o0      ! 参数 c1
ld    [ %fp + -16 ], %o1      ! 参数 c2
ld    [ %fp + -20 ], %o2      ! 参数 c3
ld    [ %fp + -24 ], %o3      ! 参数 c4
ld    [ %fp + -28 ], %o4      ! 参数 c5
ld    [ %fp + -32 ], %o5      ! 参数 c6
call    40001d30 <reg_window>
nop
st    %o0, [ %fp + -44 ]     ! 从输出寄存器
! 取得返回值 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 ]     ! 输入参数，c1→a1
  st    %i1, [ %fp + 0x48 ]     ! 输入参数，c2→a2
  st    %i2, [ %fp + 0x4c ]     ! 输入参数，c3→a3
  st    %i3, [ %fp + 0x50 ]     ! 输入参数，c4→a4
  st    %i4, [ %fp + 0x54 ]     ! 输入参数，c5→a5
  st    %i5, [ %fp + 0x58 ]     ! 输入参数，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      ! 输入参数，c7→a7
  add     %o0, %o1, %o0
  ld    [ %fp + 0x60 ], %o1      ! 输入参数，c8→a8
  add     %o0, %o1, %o0
  st    %o0, [ %fp + -12 ]
  ld    [ %fp + -12 ], %o0
  mov     %o0, %i0      ! 返回值 b
  b     40001d9c <reg_window+0x6c>
  nop
  ret
```

```
  restore
```

C 语言、汇编语言函数混合调用

C 语言与汇编语言函数相互调用时同样遵循上述参数传递原则。不同的是，需要手工确定参数地址和顺序。

例 3，C 程序调用汇编语言函数：

使用汇编语言定义数据向左位移函数 asm_shift_function(p1,p2); //p1=被位置的数据；p2=位移量

例中使用 asm_shift_function()将 1 左移 2 位。

代码如下：

```
asm("
asm_shift_function:
  save %sp, -112, %sp
  sll %i0, %i1, %i0      ！从输入寄存器中取得输入参数
  ret
  restore
  ");

main()
{
      int result = 0;
      result = asm_shift_function(1,2); //  输入参数
}
```

汇编后生成：

```
<asm_shift_function>:  ！编译器不修改汇编语言编写的代码
  save    %sp, -112, %sp
  sll    %i0, %i1, %i0
  ret
  restore
```

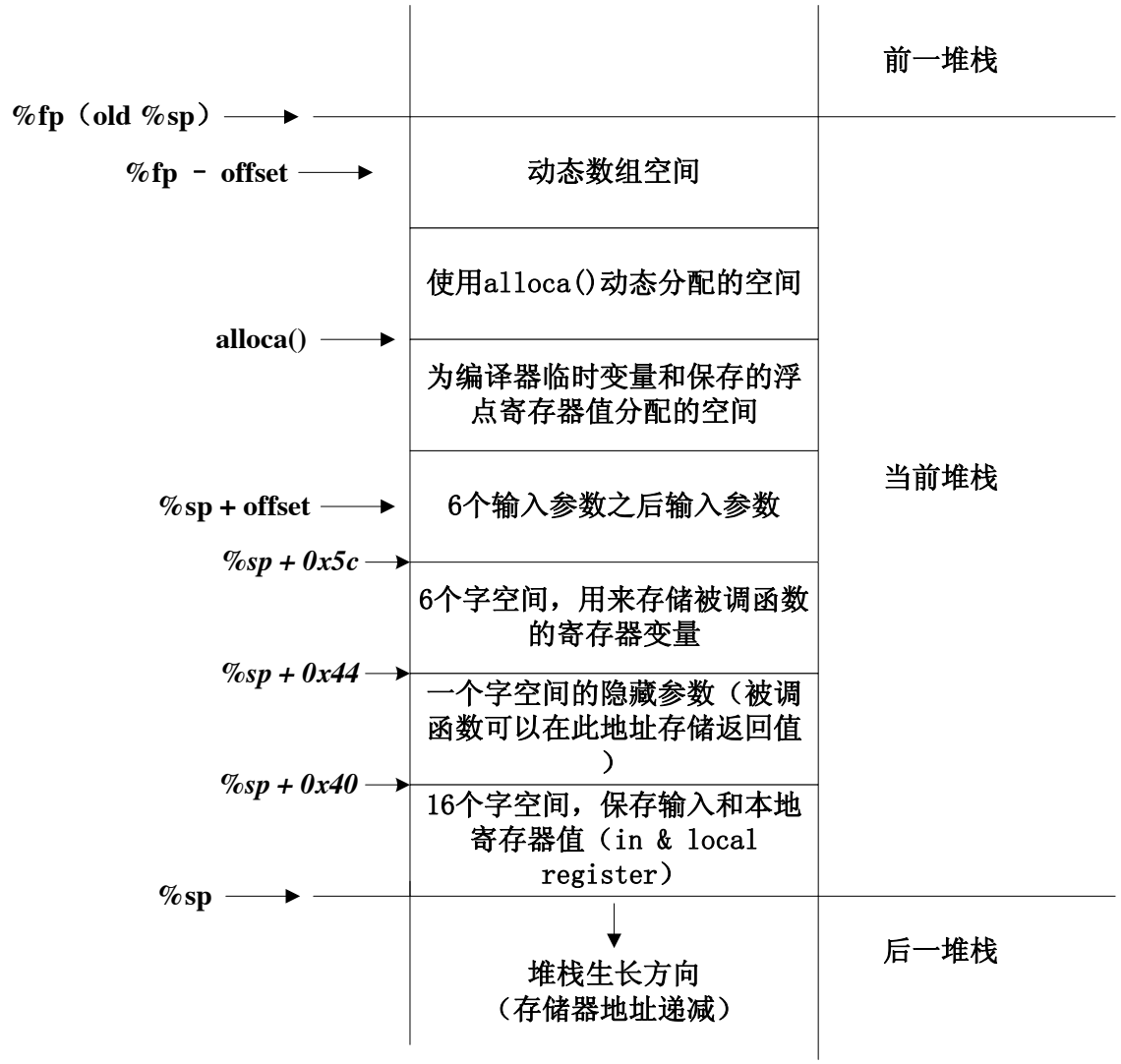
```
<main>:
  save   %sp, -152, %sp
  clr    [ %fp + -48 ]
  mov    1, %o0  ! 按照 C 代码顺序将参数放入输出寄存器
  mov    2, %o1  ! 按照 C 代码顺序将参数放入输出寄存器
  call   40001da4 <asm_shift_function>
  nop
  st     %o0, [ %fp + -48 ]    ! 从输出寄存器中取得函数返回值
  ret
  restore
```

H.3.2 堆栈

堆栈介绍

SPARC V8 标准定义的程序堆栈由堆指针（%sp）和栈指针（%fp）组成，由高地址向低地址生长。

堆栈使用方法如图所示：



每次分配一个寄存器窗口，同时也分配一段存储器空间作为堆栈。由于堆栈向上生长，因此通常程序段首地址在 ROM 或 RAM 的地址低端，堆栈段首地址在 RAM 的地址高端，以达到 RAM 利用率最高。

H.3.3 陷阱与中断

概述

BM3803 支持两种类型的陷阱：

- 精确陷阱（一般称为陷阱）
- 中断陷阱（一般称为中断）

与中断和陷阱相关的配置寄存器及相关设置如下：

1、处理器状态寄存器（%PSR）

ET：陷阱使能控制

PIL：设置处理器响应中断的级别

2、浮点状态寄存器（%FSR）

TEM：浮点陷阱使能掩码

FTT：浮点陷阱类型

AEXC：已产生浮点异常

CEXC：当前浮点异常

3、陷阱基地址寄存器（%TBR）

4、中断控制寄存器组

中断级别/优先级控制寄存器（0x8000 0090）

中断挂起寄存器（0x8000 0094）

中断强制寄存器（0x8000 0098）

中断清除寄存器（0x8000 009C）

5、陷阱/中断向量表（即中断/陷阱程序入口）

陷阱	优先级	中断类型（tt）	描述
reset	1	0x00	上电复位
data_store_error	2	0x2B	数据存储错
instruction_access_exception	3	0x01	指令访问异常
privileged_instruction	4	0x03	用户模式下执行特权指令
illegal_instruction	5	0x02	UNIMP或其他未实现指令
fp_disabled	6	0x04	禁能FPU执行浮点指令
cp_disabled	6	0x24	禁能协处理器执行协处理指令
watchpoint_detected	7	0x0B	硬件观察点命中
window_overflow	8	0x05	SAVE进入无效窗口
window_underflow	8	0x06	RESTORE进入无效窗口
register_access_error	9	0x20	寄存器堆EDAC错误
mem_address_not_aligned	10	0x07	访问存储器地址未对齐
fp_exception	11	0x08	FPU异常
data_access_exception	13	0x09	加载/存储指令访问错误
tag_overflow	14	0x0A	Tagged运算溢出
division_by_zero	15	0x2A	被0除
trap_instruction	16	0x80 − 0xFF	软件陷阱
interrupt_level_15	17	0x1F	保留
interrupt_level_14	18	0x1E	PCI中断
interrupt_level_13	19	0x1D	保留
interrupt_level_12	20	0x1C	UART 3
interrupt_level_11	21	0x1B	DSU跟踪缓冲区
interrupt_level_10	22	0x1A	保留

interrupt_level_9	23	0x19	定时器 2
interrupt_level_8	24	0x18	定时器 1
interrupt_level_7	25	0x17	GPIO中断 3
interrupt_level_6	26	0x16	GPIO中断 2
interrupt_level_5	27	0x15	GPIO中断 1
interrupt_level_4	28	0x14	GPIO中断 0
interrupt_level_3	29	0x13	UART 1
interrupt_level_2	30	0x12	UART 2
interrupt_level_1	31	0x11	AHB总线错误

陷阱控制

1、ET 和 PIL 控制

陷阱要正常工作，PSR 中的 ET 位必须为 1。ET=1 时，IU 根据中断向量表区分陷阱和中断的优先次序，只响应最高优先级的陷阱或中断。当处理最高优先级中断时，低优先级中断请求保持在中断挂起寄存器中。

对中断响应来说，IU 比较中断请求等级与%PSR 中 PIL 字段。如果中断请求等级大于或等于 PIL 字段值，即未屏蔽此中断，则处理器响应该中断请求。

当 ET=0 时，如果发生中断或未屏蔽的中断，处理器进入错误模式。

2、TEM 控制

如果 TEM 字段的特定位置 1，与之对应的浮点异常发生时，允许引起浮点异常陷阱（fp_exception）。

如果 TEM 字段的的特定位置 0，不允许引起浮点异常陷阱。此时，产生的异常会被记录在%FSR 的“已产生浮点异常”字段中（AEXC）。

如果没有屏蔽浮点异常陷阱，则引起浮点陷阱时，被中断执行的指令的目的寄存器、FCC 和 AEXC 字段保持不变。如果浮点异常被屏蔽，引起陷阱的指令的目的寄存器、FCC 和 AEXC 字段被更新。

陷阱识别

陷阱基地址寄存器（%TBR）中的“陷阱基地址”字段（TBA）保存中断向量表基地址的高 20 位。

当一个陷阱发生时，一个与之惟一对应的识别码写入%TBR 的 8 位 tt 字段，程序以%TBR 的值为目的地址进行分支。因为%TBR 的低四位为 0，因此中断向量表的每一入口包含中断处理程序的前 16 字节（4 个字）。

陷阱处理

如果 ET=1，陷阱发生时处理器做如下动作：

- 禁止陷阱：ET←0
- 保存当前用户/管理员模式：PS←S
- 转换为管理员模式：S←1
- 进入一个新的寄存器窗口：
- CWP←((CWP-1) modulo 8)
- [注意：不检查窗口是否溢出]
- 在新窗口的本地寄存器中保存当前 PC 和 nPC 值：%l1←PC，%l2←nPC
- %TBR 的 tt 字段写入陷阱或中断类型
- 若陷阱是复位陷阱，则程序分支到地址 0x0：
- PC←0，nPC←4
- 若陷阱不是复位陷阱，则程序分支到中断向量表：
- PC←%TBR，nPC←%TBR+4

如果 ET=0：

- 发生精确陷阱，则处理器进入错误模式
- 发生中断陷阱，则处理器忽略该中断陷阱

陷阱与中断的区别

- 触发位置

  陷阱发生在符合陷阱触发条件的指令运行期间；

  中断发生在一条指令运行完毕之后。

- 控制方式

  陷阱只受 ET 或 ET+TEM 自动控制

  中断不但受 ET 控制，而且 PIL 和四个中断控制寄存器控制

H.3.4 Cache 刷新

概述

BM3803 可以同过指令刷新 cache，刷新完成后，cache 的 tag 位全部清零。

刷新方法

- 指令 cache 刷新

1、执行 flush 执行

2、cache 控制寄存器的 21 位写 1

3、执行 asi=0x5 的 sta 指令（写任意地址）

- 数据 cache 刷新

1、执行 flush 执行

2、cache 控制寄存器的 22 位写 1

3、执行 asi=0x6 的 sta 指令（写任意地址）

- 刷新状态判断

cache 控制寄存器的 14、15 位置 1 分别表示数据 cache、数据 cache 正在刷新，置 0 表示刷新结束或没有刷新动作。

刷新流程

1、关闭 cache

2、刷新 cache

3、判断 cache 刷新状态，等待刷新结束

4、刷新结束后，用 lda 读取 cache tag

5、打开 cache

H.4 BM3803 软件容错设计

H.4.1 概述

BM3803 片内 regfile、cache 和存储器控制器三部分包含系统级容错能力。本章介绍这三个模块容错机能的使用方法。

H.4.2 Regfile EDAC

工作过程

BM3803 设计中对 regfile 中数据采用 EDAC 技术进行保护，可以纠正一位错，检测两位错及某些多位错。

Regfile 由 32 位数据和 7 位校验码组成。向 regfile 写数据时，根据写入数据计算生成校验位，将数据和校验位一同写入 regfile。从 regfile 读取数据时，如果检测到

可以纠正的错误，流水线断流，纠正错误，将正确的数据写入发生错误的寄存器，之后，重启流水线，从断流位置重新运行程序；如果检测到不可纠正的错误，引起“寄存器 EDAC 错”（0x20）陷阱。

寄存器说明

控制 regfile EDAC 功能的寄存器有两个，分别是%ASR16 和%ASR17。详细说明如下所示：

%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				**R**					**TCB**							**CNT**																
r/w				**r**					**r/w**							**r**															
1111				**0**					**x**							**x**															

位号	名称	描述
31:28	**CB**	**0000 = 错误计数** **0100 = 错误计数清零** **1010 = 造错使能，即对及数据位和校验位造错**
22:16	**TCB**	**对校验位造错**
15:0	**CNT**	**记录发生一位错的次数**

%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
数据位造错																															
r/w																															
x																															

位号	名称	描述
31:0	**校验位造错**	**对数据位造错**

基本操作

- 纠、检错

1、初始化开始处配置%asr16 和%asr17，清“校验位造错”和“数据位造错”字段

2、根据需要使能或禁能“单位错错误计数”。若希望记录一位错发生

次数，则首先写“造错控制”字段 0100，清错误数记录，再使能单位错计数。

- 造错

1、初始化开始处配置%asr16 和%asr17，清“校验位造错”和“数据位造错”字段

2、配置“校验位造错”和“数据位造错”字段，在希望造错的相应位置写“1”。

3、%asr16 的“造错控制”字段中写“1010”使能造错

4、向希望造错的寄存器写入数据正确数据。此时，该寄存器中保存的是已经造好错误的数据。

5、关闭造错功能

例 4：

对%l0 的数据造一位错。正确数据：0x5555 5555；造错位：数据第 0 位；造错后数据：0x5555 5554

```
mov %g0, %asr16 ! %ast16 清零
mov %g0, %asr17 ! %ast17 清零
set 0x55555555, %o0
set 0xa0000000, %o1
set 0x1, %o2
mov %o2, %asr17 ! %asr17 写入 0x1
! 对数据的第 0 位造错
mov %o1, %ast16 ! %asr16 写入 0xa000 0000
! 使能造错
mov %o0, %l1        ! 数据写入%l1，造错
mov %g0, %asr16 ! 关闭造错
mov %g0, %asr17 ! 清造错位
```

例 5：

对%l0 数据的校验位造 2 位错。正确数据：0xaaaa aaaa、校验码：0x0c；造错：校验位 4、5 位；造错后数据：0xaaaa aaaa、校验码：0x3c

mov %g0, %asr16！%ast16 清零

mov %g0, %asr17！%ast17 清零

set 0xaaaaaaaa, %o0

set 0xa0300000, %o1

mov %o1, %ast16！%asr16 写入 0xa030 0000

！校验位造 2 位错，使能造错

mov %o0, %l1　　！数据写入%l1，造错

mov %g0, %asr16！关闭造错

注意事项

1、regfile EDAC 功能不能被关闭

2、引导程序起始处关闭 regfile 造错机制

3、不能对已经造错的程序段在实际运行前进行读取操作。建议：调试时不要进行造错

4、对数据位和校验位最多共造 2 位错，如果造错多于 2 位，则很可能引起程序错误

H.4.3 Cache 奇偶校验

工作过程

BM3803 设计中对 cache 中数据和 TAG 采用奇偶校验技术进行保护，可以检测奇位错。

指令和数据 cache 的数据和 TAG 由数据位和校验码组成。向 cache 写数据时，根据写入数据计算生成校验位，将数据和校验位一同写入 cache 存储体。从 cache 读取数据时，如果检测到错误，则从直接存储器中该地址处读取数据送往 IU，并把正确的数据写入 cache。

寄存器说明

控制 cache 奇偶校验功能的寄存器有三个，详细说明如下所示：

Cache 容错控制寄存器（CCR1）Address = 0x8000_0110

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			保留															CMEEN			CNTRST	CPSEL		保留				EPOS			
r			r															r/w			r/w	r/w		r				r/w			
0	0	1	0															0			0	0		0				0			

位号	助记符	描述
31:9	VER	Cache寄存器版本号，定义为001
13:11	CMEEN	Cache造错使能标志位 ‘011’=造错开启 否则造错无效
10	CNTRST	Cache校验计数器清零标志位 1 =清零所有错误计数器
9:8	CPSEL	Cache造错目标区选择位 ‘00’=指令Cache的data区 ‘01’=指令Cache的tag区 ‘10’=数据Cache的data区 ‘11’=数据Cache的tag区
3:0	EPOS	Cache校验位造错 若选择为data区，对应于数据的4个校验位；若选择为tag区，对应于tag的4个校验位

Cache 造错寄存器（CCR2）Address = 0x8000_0114

当 Cache 造错使能，且目标区域为 Data 区时：

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																															

位号	助记符	描述
31:0	EPOS	造错位置标识低32位

当 Cache 造错使能，且目标区域为 Tag 区时：

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							

位号	助记符	描述
31	LOCK	Cache LOCK位
30:12	TAG	Cache TAG区域数据
11:8	VALID	无效位

Cache 错误计数寄存器（CCR3）Address = 0x8000-0118

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									

位号	助记符	描述
31:26	DTAGCNT	DCache Tag 错误计数器
25:16	DDATACNT	DCache Data错误计数器
15:10	ITAGCNT	ICache Tag 错误计数器
9:0	IDATACNT	ICache Data错误计数器

基本操作

- 检错

1、对“cache 造错寄存器”（CCR2）和“cache 错误计数寄存器”（CCR3）清零。

2、对 cache 容错控制寄存器（CCR1）写“0x0”，关闭造错。

- 造错

1、关闭 cache。

2、根据造错需求配置“cache 造错寄存器”（CCR2）和“cache 容错控制寄存器”（CCR1）的“cache 校验位造错”字段。注意：对 cache 数据和 TAG 部分造错时，CCR2 的定义是不同的。

3、配置“cache 容错寄存器”（CCR1）的“cache 造错目标区选择位”字段，选择需要造错的部分。

4、使能 cache。

5、从存储器中读取数据，造错。

6、关闭 cache 造错功能。

例 6，数据 cache 的数据区造错：

存储器地址 0x4001 0000 处有数据 0x1234 5678，对数据第 31 位造错，得 0x9234 5678。

set 0x12345678, %l0

```
set 0x40010000, %l1
st   %l0, [%l1]        ! 地址 0x40010000 写入
! 数据 0x12345678

set 0x40010000, %l0
set 0x80000000, %l1
set 0x1c00, %l2
set 0xc, %l3

st   %g0, [%l1 + 0x14]      ! CCR 清零，关闭 cache
st   %g0, [%l1 + 0x114] ! CCR2 清零
st   %g0, [%l1 + 0x118] ! CCR3 清零

st   %l1, [%l1 + 0x114]     ! 写 CCR2
! 配置为对数据的第 31 位造错
st   %l2, [%l1 + 0x110]     ! 写 CCR1，配置为对
! 数据 cache 的数据区进行造错
st   %l3, [%l1 + 0x14]      ! 写 CCR，使能数据 cache

ld   [%l0], %o0  ! 从地址 0x4001000 处读取数据
! 此时被造错的数据缓存进数据 cache

st   %g0, [%l1 + 0x14]      ! 关闭 cache
st   %g0, [%l1 + 0x110] ! 关闭 cache 造错
st   %g0, [%l1 + 0x114]
st   %g0, [%l1 + 0x118]

lda [%l0] 0xf, %o1          ! 从数据 cache 的数据区读取数据
```

注意事项

1、cache 奇偶校验功能总是使能

2、造错时请通过一般方式（load 和 store）操作 cache，不要使用 ASI 指令造错

3、造错时请仔细考虑有多少条指令会被缓存进 cache，特别是指令 cache。由于读取指令的连续性，往往难于对单独某一条指令进行造错。

H.4.4 存储器控制器 EDAC

工作过程

BM3803 处理器支持存储器片上纠错检错功能，能够纠正一位错，检测两位错误及某些多位错。支持 EDAC 的存储器类型如表所示：

地址范围	存储器类型		EDAC保护
0x00000000 - 0x1FFFFFFF	PROM	8 bits	支持
		16 bits	不支持
		32 bits	支持
0x20000000 - 0x3FFFFFFF	I/O	All	不支持
0x40000000 - 0x7FFFFFFF	RAM	8 bits	支持
		16 bits	不支持
		32 bits	支持

寄存器说明

控制 cache 奇偶校验功能的寄存器有三个，详细说明如下所示：

存储器容错配置寄存器 1（MECFG 1）Address = 0x8000_0100

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	SRSEEN	ROMEEN	保留			EWB2	ERB	EWB1	SDEEN	EWB0	PRBS				保留				TCBAREA		保留	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						

位号	助记符	描述
31	SR1EEN	SRAM BANK1 EDAC 使能 1 = 使能 0 = 禁能
30	SR2EEN	SRAM BANK2 EDAC 使能 1 = 使能 0 = 禁能
29	SR3EEN	SRAM BANK3 EDAC 使能 1 = 使能 0 = 禁能
28	SR4EEN	SRAM BANK4 EDAC 使能 1 = 使能 0 = 禁能
27	SR5EEN	SRAM BANK5 EDAC 使能 1 = 使能 0 = 禁能
26	ROMEEN	PROM EDAC 使能 1 = 使能 0 = 禁能
22	EWB2	EDAC写旁路 EWB2、1、0 = “011”有效
21	ERB	EDAC读旁路使能， 1 = 使能 0 = 禁能
20	EWB1	EDAC写旁路使能，EWB2、1、0 = “011”有效
19	SDEEN	SDRAM EDAC使能，1 = 使能 0 = 禁能
18	EWB0	EDAC写旁路，EWB2、1、0 = “011”有效
17:14	PRBA	PROM bank size （0000 = 8 KByte, 0001 = 16 KByte … 1111 = 256 Mbyte）
9:8	TCBAREA	将每个bank的SRAM和SDRAM平均分成4份区域。在通过设置此字段而指向的区域中，被访问的数据校验位才可以写入TCB字段，第5个BANK除外，其不受TCBarea字段控制
6:0	TCB	测试校验位，如果MECFG1中ERB使能，且对于32位数据宽度SRAM和SDRAM读数据时根据TCBarea字段，将读到的正常检验位存入此字段

存储器容错配置寄存器 2（MECFG 2）Address = 0x8000_0104

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																															

位号	助记符	描述
31:0	DR	Data Reversal数据位翻转使能位 1 = 使能 0 = 禁能

存储器容错配置寄存器 3（MECFG 3）Address = 0x8000_0108

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
保留																									PR						
r/w																									r/w						
0																									x						

位号	助记符	描述
6:0	PR	Data Reversall数据位翻转使能位 1 = 使能 0 = 禁能

基本操作

- 存储器容错寄存器（mecfg 1）配置方法

如图所示：

EDAC 使能

EWB ERB	0		1	
0	mecfg 1_1		mecfg 1_3	
	检错 纠错	W：数据位、校验位 R：数据位、校验位	检错 不纠错	W：数据位、校验位 R：数据位、校验位
1	mecfg 1_2			
	不检错 不纠错	W：数据位、校验位 R：数据位、校验位	不检错 不纠错	W：数据位、校验位 R：数据位、校验位

EDAC 禁能

EWB ERB	0		1	
0			mecfg 1_7	
	不检错 不纠错	W：数据位、校验位 R：数据位、校验位	不检错 不纠错	W：数据位、校验位 R：数据位、校验位
1	mecfg 1_6			
	不检错 不纠错	W：数据位、校验位 R：数据位、校验位	不检错 不纠错	W：数据位、校验位 R：数据位、校验位

说明：

1、EWB、ERB 为“0”表示禁能，为“1”表示使能（EWB 使能=011）。

2、mecfg1_x 表示对一种“EDAC、EWB、ERB”组合方式的命名。如：mecfg1_1，表示“EDAC 使能、EWB 禁能、ERB 禁能”。

3、“W”表示 write，即向存储器写；“R”表示 read，即从存储器读。

4、框起来的字，写入时表示“对写入的数据造错”，读出时表示“不能读出实际数据”。即；

W：数据位，表示写入时对数据位造错

R：校验位，表示读出时无法得到实际的校验位

5、例如：

mecfg1_7 配置：

EDAC = 0；EWB = 001；ERB = 0

此时，不检错、不纠错

写入时对数据位和校验位造错

读出时只能得到实际的数据位，无法得到实际的校验位

- 纠、检错

  1、mecfg 2、3 清零，禁能存储器造错

  2、根据需要配置 mecfg1 为 mecfg1_1 配置

- 造错

  1、根据造错需要配置 mecfg2 和 mecfg3

  2、根据上表配置 mecfg1 使能造错

  3、向存储器中写入数据，造错成功

  例 7：

向 SDRAM 地址 0x6000_0000 写数据 0x8765_4321，对数据第 0 位造错，造错后数据为 0x8765_4320

```
set 0x60000000, %l0
set 0x87654321, %l1
set 0x01, %l2
set 0x80000100, %l3
set 0x001c0000, %l4 ! mecfg 1_7
st   %g0, [%l3 + 0x08] ! 清 mecfg3
st   %l2, [%l3 + 0x04] ! 配置 mecfg2
! 对数据的第 0 位遭造错
st   %l4, [%l3]        ! 配置 mecfg1，使能造错
st   %l1, [%l0] ! 造错
st   %g0, [%l3]          ! 清 mecfg1
st   %g0, [%l3 + 0x04] ! 清 mecfg2
st   %g0, [%l3 + 0x08] ! 清 mecfg3
```

注意事项

1、BM3803 对 PROM 空间没有写校验码的能力，校验码需要另行写入。

2、只能读出一个 bank 的四分之一空间里的校验位。特别注意：读取校验位的地址和程序运行的地址应处在同一 bank 的不同四分之一中，否则校验码会被连续读出的指令的校验码冲掉。