32 位四核处理器 SOC 芯片

S698P4-II 用户手册

（版本：V2.1）

珠海欧比特控制工程股份有限公司

地址:广东省珠海市唐家东岸白沙路 1 号欧比特科技园　邮编:519080

电话:0756-3391979　传真:0756-3391980　网址:www.myorbita.net

http://www.myorbita.net

修改记录

版本变更	日期	修改内容	备注
V2.0	2013-01-17	对文件格式、版式等进行修订后，重新发布。	
V2.1	2014-05-17	对 1.4 节产品信息内容进行了更新,增加了 CQFP256 封装描述。	

版权声明

免责声明

本文档仅提供阶段性信息，所含内容可根据产品的实际情况随时更新，恕不另行通知。如因文档使用不当造成的直接或间接损失，珠海欧比特控制工程股份有限公司不承担任何责任。

珠海欧比特控制工程股份有限公司

ZHUHAI ORBITA CONTROL ENGINEERING CO. , LTD

地址(Addr)：广东省珠海市唐家东岸白沙路 1 号欧比特科技园

Orbita Tech Park, 1 Baisha Road, Tangjia Dong'an, Zhuhai, Guangdong, China

邮编：519080

电话(Tel)：+86 756-3391979

传真(Fax)：+86 756-3391980

网址(web)：www.myorbita.net

目 录

1、概述

S698P4-II 是基于SPARC V8架构的高性能的32位RISC嵌入式4核处理器。采用 SMP “对称多处理”技术，是在一个内核里集成四个功能一样的处理器核心，各 CPU 之间共享内存子系统以及总线结构，总线竞争和仲裁由硬件自动完成，不需要用户设置。它专为嵌入式应用而设计，具有高性能，低复杂度和低功耗的特点。

S698P4-II 支持多核并行处理机制，采用 eCos 实时嵌入式操作系统。eCos 将任务队列对称地分布于多个 CPU 之上，从而极大地提高了整个系统的数据处理能力。所有的处理器都可以平等地访问内存、I/O 和外部中断。系统资源被系统中所有 CPU 共享，工作负载能够均匀地分配到所有可用处理器之上，其运算速度快，数据处理量大、能耗低，性能和可靠性远高于单核处理器。

S698P4-II 处理器可广泛应用于航空航天的高端电子设备、海量数据处理、大规模网络应用、复杂科学计算及大型图形建模为特征的企业或行业等领域。

1.1 S698P4-II 架构和特点

S698P4-II 处理器是基于 SPARC V8 架构的内部集成 4 核的高性能，低功耗 32 位微处理器，处理器内部集成了 IU 单元、浮点单元(FPU)单元以及内存控制器。

针对实时应用的嵌入式领域，S698P4-II 提供了内部看门狗、定时器、中断控制器以及串行通讯接口；针对航空航天领域，S698P4-II 提供了 CAN 总线接口及以太网接口。

同时，为了芯片调试的方便，芯片内部还集成了硬件调试专用接口 DSU，不仅可通过串口进行调试，而且还可以通过以太网进行调试，大大方便了用户的使用。

图 1-1 为 S698P4-II 的架构。

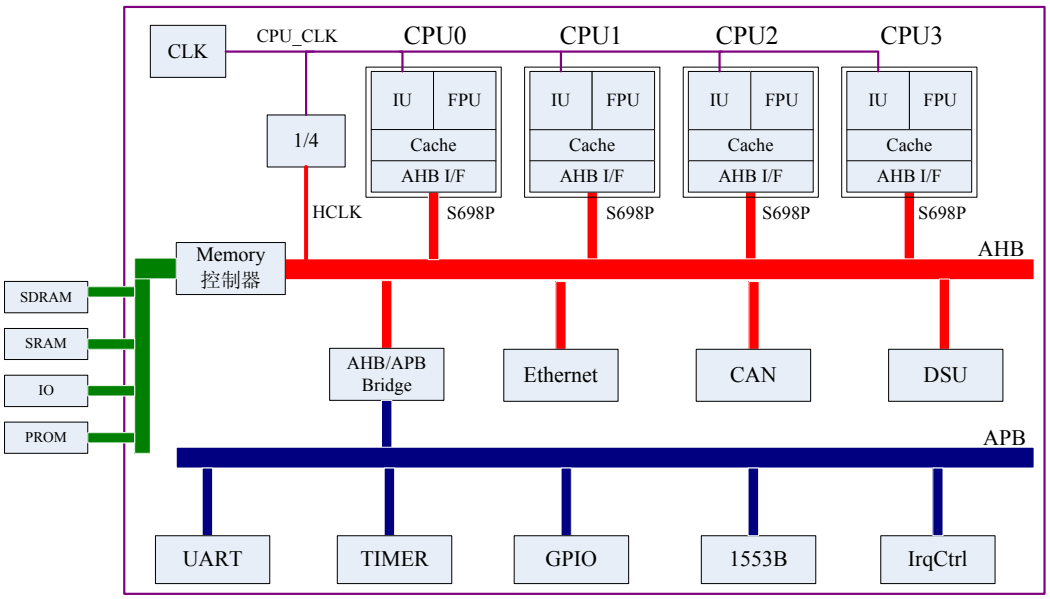


图 1-1 S698P4-II 架构

特点:

- 并行对称多处理架构
- 集成了 4 个高性能 CPU 的处理器内核，每个 CPU 包括：
  - 32位整型数处理单元
  - RISC结构
  - 硬件乘法器和除法器
  - 支持2条DSP指令(MAC & UMAC)
  - 7级流水
  - 优化的32/64位浮点数处理单元，符合IEEE-754标准
  - 8K 数据缓存(data cache)和指令缓存(instration cache)
- CPU 共享外设和存贮空间
- 片上外设
  - 存储器控制器：
    - PROM, SRAM, SDRAM，I/O接口
    - 片选信号生成器

- ◆ 等待周期生成器
- ◆ 存储器写保护

- 定时器
  - ◆ 2 个 32 位定时器
  - ◆ 1 个 32 位看门狗定时器(与定时器 2 复用)
- 中断控制器
- 8位 GPIO
- 2个UART 控制器
- 以太网控制器

- 1553B 总线控制器
- CAN 总线控制器
- 集成调试支持单元 DSU（Debug Support Unit）
- 0.13μm CMOS 工艺生产
- I/O 接口电压：3.3V±0.3V；核心电压：1.2±0.1V
- 工作频率：0MHz～400MHz
- 功耗：1.3W@400Mhz
- 性能:
  - 1000 MIPs @ 400 MHz （Dhrystone 2.1)
  - 400 MFlops@ 400 MHz （Whetstone)
- 工作温度：-55° ～ +125°
- 封装: CQFP256，PQFP208

1.2 多时钟机制

S698P4-II 单核最高运行速度可达 400MHz，如果整个芯片都采用同一个时钟，则所有外设和板级设备都要求运行在 400MHz 的频率上。这会增大系统设计的难度，并增加系统功耗，降低系统的稳定性。

S698P4-II 采用多时钟机制来平衡高速 CPU 内核和低速的外部设备之间的矛盾，在提高芯片性能的同时，避免对板级设备提出过高的要求。

S698P4-II 的时钟电路产生高速的 cpu 时钟 CPU_CLK，CPU_CLK 除了供给 4 个 CPU 内核使用外，还经过 4 分频电路，产生 HCLK 供给 AMBA 总线和外设使用。

S698P4-II 四个内核采用同样的时钟。他们全部从 CPU_CLK 得来。并且跟 CPU_CLK 同频，同相。

CPU_CLK、CPU0_CLK 到 CPU3_CLK 和 HCLK 之间的相位关系如图 1－3 所示。

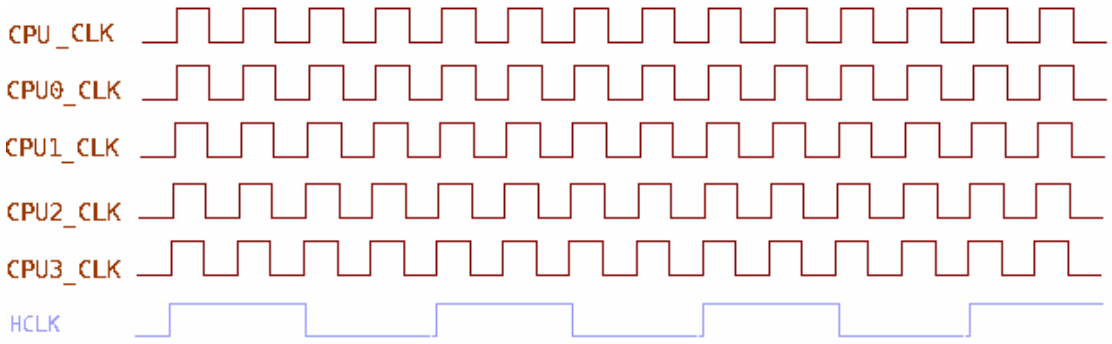


图 1－3 S698P4-II 内部时钟

HCLK 的频率是 CPU_CLK 的四分之一。在 CPU 需要访问 AMBA 总线和外设时，信号的时序需要按照 HCLK 的时序进行。如果 CPU CLK 运行在 400M，则 HCLK 只需要运行在 100M，在板级设备上，普通的 SDRAM 即可满足要求。对于其它慢速设备，则可以通过配置存贮器控制器寄存器的读写等待周期解决。

1.3 S698P4-II 调度机制

S698P4-II 在一个处理器上汇集了 4 个 CPU,各 CPU 之间共享同一个操作系统、内存子系统、总线结构和 I/O 系统等。同时使用多个 CPU 时，从管理的角度来看，它们的表现就像一台单机一样。正常启动后，所有的 CPU 无主从之分，都可以平等地访问内存、I/O 和外部中断。

S698P4-II 各 CPU 之间的通信是通过多核中断控制器(MP IRQCTRL)的中断来实现的，其结构如图 1－4：

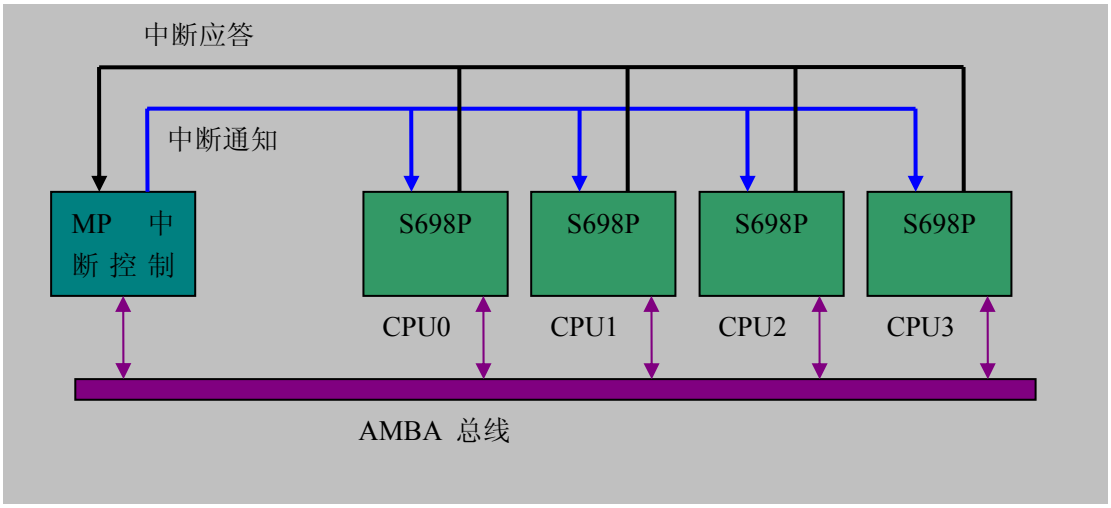
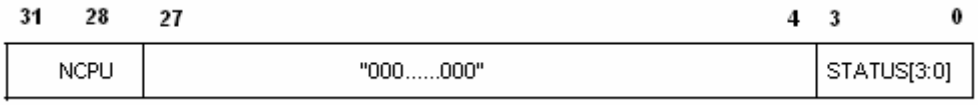


图 1-4 多核中断控制器

S698P4-II 中的每个 CPU 都可以通过多核中断控制器向其它 CPU 发中断请求，每个 CPU 都可以响应其它 CPU 的中断请求。在多核中断控制器中，有一个称为多处理器状态寄存器(Multi-processor status register)，其后 4 位(STATUS[3:0])分别控制 4 个 CPU 的状态,写入 1，其相应的 CPU 就会被激活；写入 0，其相应的 CPU 就会进入休眠。S698P4-II 启动时，CPU 有主 CPU（CPU0）和从 CPU 之分，启动完之后，所有 CPU 不分主从。CPU0 的启动顺序和其他 CPU 的启动顺序是不同的，上电或者软复位后，S698P4-II 先启动 CPU0，其它 CPU 处于 power down 状态，在 CPU0 初始化完成后，通过设置多处理器状态寄存器启动、初始化其他 CPU，之后所有 CPU 无主从之分。

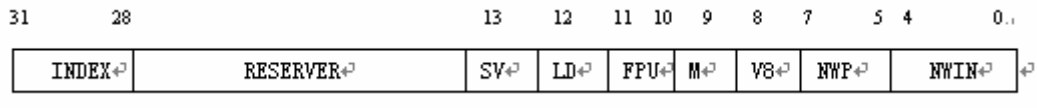
多处理器状态寄存器：



S698P4-II 各个 CPU 均有一个称为%asr17 的寄存器，其 31-28 位(下图 INDEX

部分)，指明当前运行的是哪个 CPU,程序可以从这个寄存器得到当前是在哪个 CPU 上运行，并作相应的处理。

%asr17 寄存器:



在 SMP 系统中，系统资源被系统中所有处理器共享，工作负载能够均匀地分配到所有可用处理器之上。并且因为结构共享存储器、统一地址空间，系统编程比较容易。系统将任务队列对称地分布于多个 CPU 之上，从而极大地提高了整个系统的数据处理能力。

eCos 支持对称多处理器(SMP)系统，多 CPU 之间的任务调度采用多级队列调度，主要调度算法有：时间片轮转调度算法和抢占式优先权调度算法。

多级队列调度的优先级数目在调度器配置的时候给出，最多有 32 个优先级，0 位最高优先级。每个优先级上都有一个队列。每个队列支持多个线程。单个队列中各线程优先级相同，同优先级线程可支持时间片轮转。

在系统调度方法：

任务调度仅在激活的 CPU 上进行。下面的调度在 4 个 CPU 都处于激活状态。当前任务数小于等于 4 个时，系统将每个任务分配一个 CPU 上，之后系统不会进行 CPU 间任务调度，一直运行到结束。当前任务数大于 4 个时，系统才会在 CPU 间进行任务调度。

当任务大于 4 个时，系统将进行 CPU 间的任务调度。调度算法采用时间片轮转调度算法和抢占式优先权调度算法。系统将任务就绪队列中优先级最高的 4 个任务分配到 4 个 CPU 上，每个 CPU 开始执行任务，任务执行时间以时间片为

单位。当时间片时间到达时会产生一个定时器中断，当系统定时器中断产生时，由其中一个 CPU （不确定）接收定时中断，接收定时中断的 CPU 必须为所有的 CPU 的时间片计数器进行操作。当某个 CPU 的时间片计数器到达 0 时，它将给该 CPU 发送一个时间片中断，当其他 CPU 接收到时间片中断时（S698P4-II 每个 CPU 都必须处理时间片），该 CPU 比较当前任务和任务就绪表中最高优先级的任务，如果后者的优先级比前者的优先级高，则系统就会产生调度。CPU 把任务就绪表中最高优先级的任务调到该 CPU 上运行，把先前的任务重新在任务就绪表排队。只要出现了另一个优先权更高的任务，调度程序就在下一个时间片中断暂停原最高优先权任务的执行，而将 CPU 分配给新出现的优先权最高的任务。直到当前任务数小于等于 4 个时，系统才停止 CPU 间调度。

1.4 产品信息

序号	产品型号	产品描述
1	S698P4-PI	S698P4-PI 塑料封装，PQFP208，工业级
2	S698P4-II-CE	S698P4-II-CE 陶瓷封装，CQFP256，工程样片
3	S698P4-II-C	S698P4-II-C 陶瓷封装，CQFP256，军级
4	S698P4-ASIC-IP-F	ASIC 版本固核（ASIC 网表）
5	S698P4-FPGA-IP-V	FPGA 版本固核（FPGA 网表）
6	S698P4-RTL-IP-S	软核（RTL 源码）

2、 S698P4-II CPU 单元

前面提到，S698P4-II 采用 SMP 架构。它的核心由四个 S698P CPU 组成。每个 S698P4-II CPU 都是一个独立的执行单元。S698P4-II CPU 整数单元（IU）采用 7 级流水线处理机制，指令集兼容于 SPARC V8 指令集，所有的指令均为 32 位宽。它的指令结构简单，编码和寄存器地址统一。寄存器窗口为快速传递参数和运行程序提供了保证，快速陷阱切换是实时系统准确运行的基础。

每个 S698P CPU 内部模块之间的关系如图 2-1 所示。

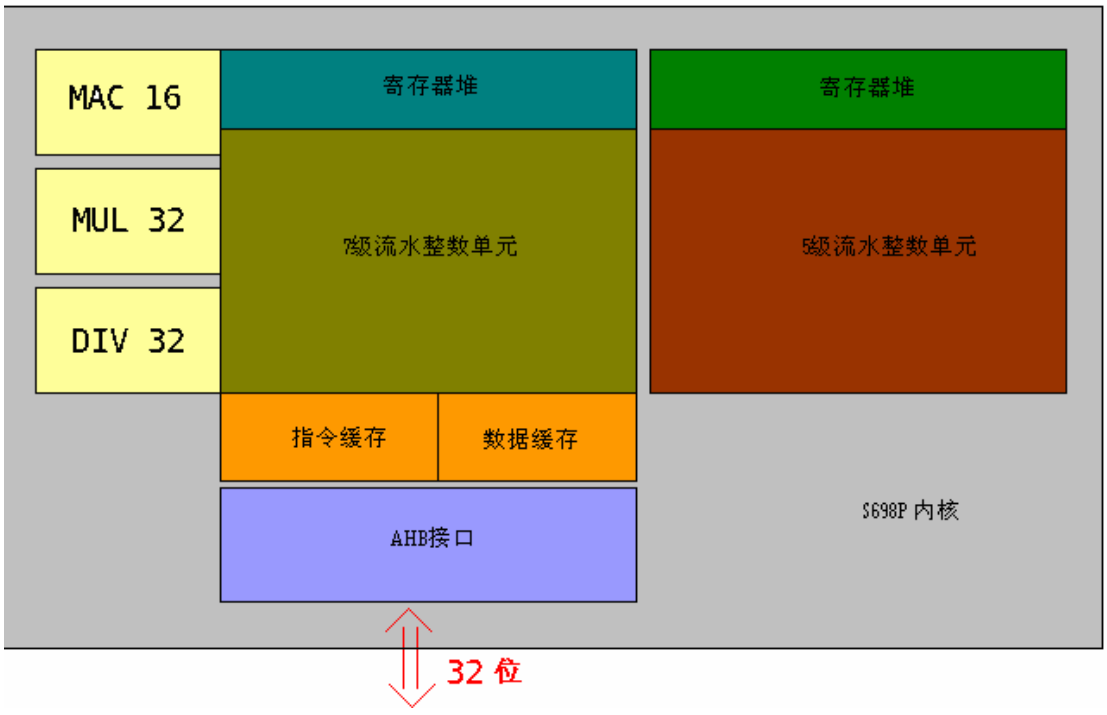


图 2-1 S698P CPU 架构图

整个 S698P4-II CPU 的核心是整数单元（IU）。它主要负责：

- 执行整数指令
- 提交浮点指令给浮点单元(FPU)，并在适当时间取浮点单元(FPU)的运算结果
- 异常和中断处理

- 和缓存交换数据

MAC 16、MUL 32 和 DIV 32 属于整数单元的一部分，分别负责完成 MAC、MUL 和 DIV 指令的执行。

浮点单元(FPU)负责执行浮点指令。它执行的指令由整数单元(IU)负责提供。整数单元在把当前 PC 对应的浮点指令提供给浮点单元(FPU)后，马上开始执行下一条指令。浮点指令由浮点单元(FPU)单元执行完成。在执行完成后，相应的结果放入到浮点单元(FPU)的寄存器中，可以供以后运算使用，或者由整数单元(IU)在需要的时候读取。如果在运算过程中发生了异常，则异常会提交给 IU 进行处理。

指令缓存(instration cache)负责提供整数单元(IU)运行需要的指令。它从外部存贮器读取需要的指令，并放在内部随机存贮器中，供整数单元(IU)在需要的时候使用。

数据缓存(data cache) 负责提供整数单元(IU)运行需要的数据，并完成数据一致性的功能。

2.1 整数单元(IU)

整数单元(IU)具有下列的主要特性：

- 7 级的指令流水线
- 独立的指令和数据缓存接口
- 支持 8 个寄存器窗口
- 拥有可选择的 16x 16 位 MAC 和 40 位累加器的硬件乘法器
- 指数为 2 的除法器
- 为减少的代码尺寸的单一矢量陷阱

在实现上，整数单元(IU)使用一个 7 级的指令流水线，它们是：

1) 预取指令(FE): 如果指令缓存使能，指令预取到指令缓存中。否则，预取被转送给存储器控制器。指令在这一个阶段结束的时候有效和被锁进整数单元(IU)。
2) 译码(DE): 指令被解码，产生调用和分支对象地址。
3) 寄存器访问(RA)： 操作数从寄存器中或者内在的数据旁路被读出。
4) 运行(EX)：运行累加器,逻辑和移位操作。
5) 存储器(ME): 数据缓存被访问。在执行阶段中读出要存储的数据，同时写到数据缓存中。
6) 例外 (XC)： 陷阱和中断。
7) 写 (WR)： 任何 累加器,逻辑的, 移位 , 或缓存操作的结果被回写到寄存器。

整数单元(IU)包括一些通用目的寄存器，控制处理器的全部工作。整数单元(IU)的主要功能是执行整数运算、计算要访问的存储器的地址，和控制浮点单元(FPU)指令的执行。

当 IU 访问存储器时，会在访问地址后面加一个“地址空间标志符”（ASI），ASI 表示处理器是处于管理模式还是用户模式；访问操作是访问指令存储器还是数据存储器。

S698P 的指令按照功能分为 6 类：存储器存取指令、算术运算/逻辑运算/移位指令、跳转控制指令、读/写寄存器指令、浮点运算指令和其它指令。

- **存储器存取指令**

存储器存取指令是唯一用来访问存储器的指令。存储器存取指令用2个‘r’寄存器或者1个‘r’寄存器和1个13位的无符号立即数计算出1个32位、按字节排列的存储器地址，IU再在该地址后面加上“地址空间标志符（ASI）”以决定处理器是处于管理模式还是用户模式，是访问指令存储器还是数据存储器。

存储器存取指令的目标域指定是一个*r*寄存器，*f*寄存器，或者是协处理器寄

存器（此寄存器提供存储的数据或取得要载入的数据）。

整数存取指令支持字节方式(8位)、半字方式(16位)、字方式(32位)和双字方式(64位)。 整数存取指令包括一些整数取数指令，用来从内存中取得8位或16位的单精度数并放入目的寄存器中。浮点和协处理器存取指令支持字方式和双字方式。

S698P4-II 对半字的大于一个字节类型的数据采用高地址优先存储的方式。

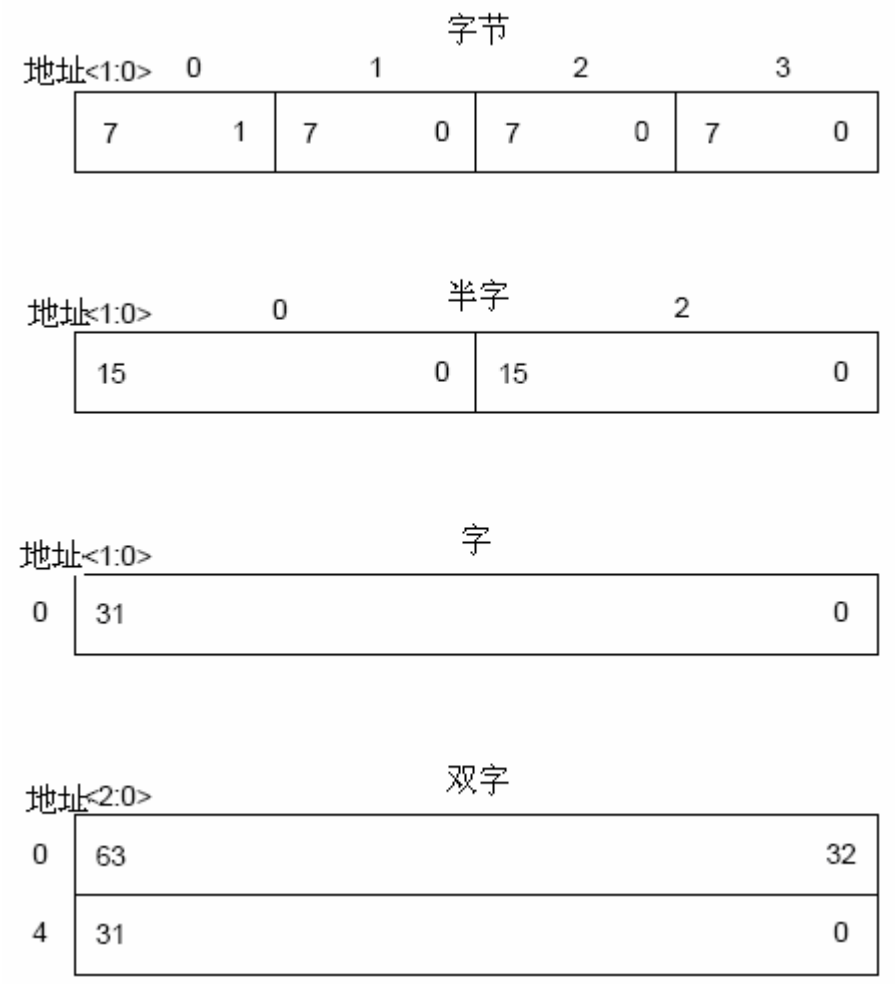


图 2-2 地址转换

- **算术运算/逻辑运算/移位指令**

算术运算/逻辑运算/移位指令提供了算术运算、逻辑运算和移位操作。这些

指令除“SETHI”指令之外，都包括2个操作码，并由这2个操作码运算产生一个结果，这个结果或者放入目的寄存器中，或者丢弃。“SETHI”指令是一条专门的指令，用来和它后面的指令一起创建一个32位的常数并放入r寄存器中。

移位指令用来把‘r’寄存器中的内容右移或左移‘n’位，‘n’的值由指令中的常数或’r’寄存器中的值指定。

整数乘法指令提供有符号或无符号的32 × 32 → 64位操作。整数除法指令提供有符号或无符号的 64 ÷32 → 32位操作。整数乘法指令和整数除法指令的结果会影响“PSR”的相应位。被‘0’整除会产生一个“陷阱”。

- **跳转控制指令**

跳转控制指令（**CTI**s）包括与PC指针相关的分支指令和调用指令、寄存器间接跳转指令和条件陷阱s。大多数跳转控制指令是延时的跳转控制指令（**DCTI**s）。跟在延时跳转控制指令后面的指令会在此跳转完成之前执行。

跟在跳转控制指令后的指令叫延时指令。延时指令总是预取的，即使跳转控制指令是无条件分支。然而，如果跳转控制指令不被执行，跳转控制指令的某一位可以使延时指令也不被执行。

- **读/写状态寄存器指令**

读/写寄存器指令用来读/写用户可见的状态寄存器，也可读/写辅助状态寄存器。

- **浮点运算指令**

浮点运算指令用来完成所有的浮点运算。浮点运算指令是寄存器到寄存器的指令，浮点运算操作利用浮点寄存器进行。象算术运算指令/逻辑运算指令和移位指令一样，浮点运算指令有1或2个源操作数，并得出一个结果。

IU 的大部分指令为单周期，但是， 极少数指令是多周期指令。表 2-2 列出了指令的周期。

表 2-2 指令周期

指令	周期
跳转类	3
双装载	2
单存贮	2
双存储	3
乘法	5
除法	35
陷阱	5
加载/存贮	3
其它	1

2.2 浮点单元(FPU)

浮点单元(FPU)有三个作用：执行浮点指令操作，侦测与操作相关的数据，处理操作异常。浮点单元(FPU)支持单精度和双精度浮点运算，支持 SPARC V8 浮点单元(FPU)指令。通过浮点单元(FPU) 控制器（MFC）连接，执行运算操作与 IU 并行。MFC 执行 SPARC 延迟陷阱模型。

浮点单元(FPU)有32个32位的浮点寄存器。双精度浮点数占用一对偶奇寄存器，四精度浮点数占用四个四字节对齐的寄存器。因此，浮点寄存器能保存32个单精度(32位)、16个双精度(64位)、8个四倍精度寄存器(128位)的数。

浮点 load/store指令用来在浮点单元(FPU)和内存之间交换数据，内存地址由IU计算。浮点数操作指令使用实际的浮点数运算。浮点数数据格式和指令集遵

循IEEE标准。

对于所有的SPARC 浮点单元(FPU)来说，无论它们实现哪种版本的SPARC结构，它们都提供了一个浮点状态寄存器(FSR)，该寄存器中包含与浮点单元(FPU)相关的状态位和控制位。FSR可由用户软件访问，以检测浮点异常、舍入方向和非标准的算法模式。在PSR的EF位为0的情况下,执行一条浮点数指令就会引发一个fp_disabled的陷阱。

浮点单元(FPU)是和整数单元（IU）并立的执行单元，专门负责浮点数处理。它和整数单元（IU）并行执行。

在实现上，浮点单元(FPU)使用了一个 5 级的指令流水线：它们是：

1) 译码 (DE): 指令被解码，产生操作对象地址。
2) 寄存器访问(RA)： 操作数从寄存器中或者内在的数据旁路被读出。
3) 运行 (EX)：执行浮点运算。
4) 例外 (XC)：空操作，例外由 IU 进行处理。
5) 写 (WB)： 缓存操作的结果被回写到寄存器文件。

浮点单元(FPU)支持的浮点操作指令符合 SPARC V8 标准，操作指令主要可以分为六类：

1. 浮点转换类指令

1). 整数转换为浮点数：

操作码	*代码*	*操作*
FiTOs	011000100	整数转换为单精度数
FiTOd	011001000	整数转换为双精度数

2). 浮点数转换为整数：

操作码	代码	操作
FsTOi	011010001	单精度数转换为整数
FdTOi	011010010	双精度数转换为整数

3).浮点格式之间相互转换：

操作码	代码	操作
FsTOd	011001001	单精度数转换为双精度数
FdTOs	011001110	双精度数转换为单精度数

该类指令操作数为单操作数。

2.浮点移动类指令

操作码	*代码*	*操作*
FMOVs	000000001	单精度数移动
FNEGs	000000101	单精度数取相反数
FABSs	000001001	单精度数取绝对值

该类指令操作数为单操作数。

3.浮点加减类指令

操作码	*代码*	*操作*
FADDs	001000001	单精度数加法运算
FADDd	001000010	双精度数加法运算
FSUBs	001000101	单精度数减法运算
FSUBd	001000110	双精度数减法运算

4.浮点乘除类指令

操作码	*代码*	*操作*
FMULs	001001001	单精度数相乘运算
FMULd	001001010	双精度数相乘运算
FDIVs	001001101	单精度数相除运算
FDIVd	001001110	双精度数相除运算

5.浮点开方类指令

操作码	*代码*	*操作*
FSQRTs	000101001	单精度数开方运算
FSQRTd	000101010	双精度数开方运算

该类指令操作数为单操作数。

6.浮点比较类指令

操作码	*代码*	*操作*
FCMPs	001010001	单精度数比较
FCMPd	001010010	双精度数比较

FCMPEs	001010101	单精度数比较
FCMPEd	001010110	双精度数比较

2.3 缓存(cache)

为了解决高速运行的 CPU 和外部缓慢的存贮器之间的矛盾，大多数处理器都加入了缓存系统，可以极大的提高系统的运行速度。在多处理器系统中，还能有效降低总线带宽占用，减少总线冲突，进一步提高系统的整体性能。S698P4-II 采用本地缓存的结构，每个处理器都有自己独立的数据缓存和指令缓存，使用 LRU 算法进行数据调度，采用透写策略（write-through）更新数据。在地址映射时，采用的是全相联映像法。

1.1.1 2.3.1 指令缓存(instration cache)

指令缓存可以配置成一个直接映射缓存或者一个 4 组缓存 ，缓存使用 LRU 算法。每组的大小为 2 千字节, 缓存划分为每行 32 个字节的数据。 每行有一个缓存标签，还结合有标签区域, 每 4 个字节子块的有效区域有一位有效位、和锁位。 当没有命中高速缓冲时，指令预取，对应的标签和数据行更新。 在多组配置中一行依照 LRU 法替代。

如果指令突发预取在缓存控制寄存器 (CCR)中使能，缓存行被从丢失地址开始的主存储器区域填充，直到行结束。 同时，指令被转寄到整数单元(IU) 。如果由于内在的依赖或者多周期指令，IU 没有接收数据流，则整数单元(IU) 被停止，直到行填充被完成。 如果整数单元(IU) 在行填充期间运行一个控制传输指令 (分支/调用/跳转/陷阱)，行填充将会在下次预取时结束。 如果指令突发预取使能,即使缓存被无效，指令流可以工作。在这情况，预取的指令只被转寄到整数单元(IU) ，缓存不更新。 在缓存行再填充期间，在 AHB 总线上产生逐渐增加的突发脉冲。

如果在 IU 停止期间，进行行填充时产生一个存储器访问错误， 在缓存标签

的对应的有效位将不被设定。 如果整数单元(IU) 稍后预取来自无效地址的指令,一个缓存错误将会发生，触发一个对无效地址的访问。 如果错误保持,一个指令错误陷阱 (编号 1) 将会产生。

1.1.2 2.3.2 数据缓存(data cache)

数据缓存由两部分构成。一部分用来存贮缓存的标志（TAG）等信息，另一部分用来保存缓存的数据。

缓存的标志等信息的存贮器称为标志存贮器。每个标志的结构定义如下:

31	10	9	8	7	0
ATAG	LRR	LOCK		VALID	

各个区域的含义：

[31:11]: 地址标志(ATAG) - 主存贮器的地址信息

[10]: LRR - 使用 LRR 算法更新数据。“0”表示不使用

[9:8]: LOCK - 当置位时，这条缓存线被锁定

[7:0]: Valid (V) - 当置位时，表示当前缓存线的数据无效

主存贮器地址的高 21 位保存在 ATAG 中，低 11 位则决定了数据在缓存中的位置。一条 TAG 对应了一块缓存区域，这块区域称为一条缓存线,目前 1 个缓存线大小为 32 字节。

如果 CPU 需要数据，它自己的缓存首先检查数据是否已经在自己的缓存中，如果是(称为缓存命中)，则直接返回数据。如果没有，则从主存中调入这个数据到自己的缓存中，同时传输给 CPU。

S698P4-II 中的数据缓存采用透写策略（write-through）更新数据。当 CPU 把修改后的数据写入到缓存中时，缓存在更新自己缓存的同时，也通过 AHB 总线把数据写到主存贮器中。

S698P4-II 使用基于“写无效”的数据一致性策略。所有的缓存都会监听地址和数据总线，当总线上存在一个写操作时，缓存控制器会检查这个地址是否命中自己的缓存，如果命中，则无效自己相应的缓存区域。当 CPU 需要这个数据时，缓存就会从主存中读取，从而可以使自己的数据和主存中的数据保持一致。

如果在缓存无效这个缓存区域时，CPU 正好需要这个数据，则缓存会把当前数据总线上的数据返回给 CPU，并更新自己的缓存。

为了提高效率，缓存使用双口存贮器保存地址标志，CPU 访问缓存和数据侦听是从两个不同的端口读取标志数据的。

2.4 寄存器

2.4.1 整数单元(IU)寄存器

2.4.1.1 整数单元(IU)通用目的寄存器(r register)

S698P 的整型数处理单元包含 136 个 32 位的通用目的寄存器（***r*** 寄存器）。这些通用目的寄存器被分成 8 个***全局***寄存器、8 个***输入***寄存器、8 个输出寄存器和 8 个***本地***寄存器。见表 2-3。

表 2-3 窗口地址

窗口寄存器	寄存器(r)地址
输入[0] - 输入[7]	r[24] - r[31]
本地[0] - 本地[7]	r[16] - r[23]
输出[0] - 输出[7]	r[8] - r[15]
全局[0] - 全局[7]	r[0] - r[7]

寄存器窗口

在特定的时间内，一条指令可以访问136个寄存器中的8个*全局*寄存器和一个寄存器窗口。其中，寄存器窗口是一个24寄存器组，它包括一个由8个**“输入”** 寄存器和8个**“本地”**寄存器组成的16寄存器组，和与它相邻的16寄存器组中的8个**“输入”** 寄存器(被当前寄存器窗口设定为自己的**“输出”** 寄存器)。

当前的寄存器窗口在“处理器状态寄存器(PSR)”中一个5位的“当前窗口指针 (CWP)”区设定，当执行“恢复（或返回）”指令时，CWP加1；当执行“保存”指令或产生一个陷阱时，CWP的值减1。寄存器窗口上溢和下溢由“窗口无效寄存器”监控，“窗口无效寄存器”由管理软件控制。

寄存器窗口的重叠

每个寄存器窗口都和与之相邻的2个寄存器窗口共享*输入*寄存器和*输出*寄存器。CWP+1寄存器窗口的*输出*寄存器被设定为当前寄存器窗口的*输入*寄存器；当前寄存器窗口的*输出*寄存器则被设定为CWP-1寄存器窗口的*输入*寄存器。*本地*寄存器对于每个寄存器窗口来说都是唯一的。

一个地址设定为“*o*”的“*r*”寄存器（其中$8 \le o \le 15$），和（CWP-1）下地址为“(*o* + 16)”的寄存器，被认为是同一个寄存器。同样的，一个地址设定为“i”的“*r*”寄存器（ 24 ≤ I ≤ 31），和在（CWP+1）下地址为“(i－ 16)”的寄存器，被认为是同一个寄存器。

S698P包含8个寄存器窗口。编号最高的寄存器窗口（编号为7）与编号最低的寄存器窗口（编号为0）交迭在一起，即寄存器窗口7的*输入*寄存器与寄存器窗口0的*输出*寄存器是同一个寄存器。

编程注意事项：

- 由于过程调用指令“调用”和“跳转”都不改变CWP的值，所以过程调用不改变寄存器窗口。
- 当通过保存指令重新进入窗口时就不能确定“本地”、“输出”寄存器中

的值。由于在恢复和保存之间可能会发生陷阱，执行的程序中若有恢复指令，且其后跟有保存指令，则结果窗口中的“本地”和“输出”寄存器中的数值就不能确定。然而如果陷阱被屏蔽，“本地”和“输出”寄存器中的值就有效了。

- 因为寄存器窗口有交迭，所以，软件可以利用的寄存器窗口比IU实际实现的寄存器窗口少1，即NWINDOWS–1。当寄存器集合被填满时，最新寄存器窗口的“输出”寄存器和仍然保留着有效数据的最旧的寄存器窗口的“输入寄存器”是同一个寄存器。

2.4.1.2 整数单元(IU)控制状态寄存器

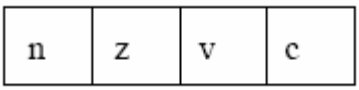
32位的IU控制/状态寄存器包括：处理器状态寄存器(PSR)、窗口无效掩码寄存器(WIM)、陷阱基础寄存器(TBR)、乘法/除法 (Y)寄存器、程序指针寄存器 (PC和 nPC)、4个辅助状态寄存器(ASR)。

处理器状态寄存器(PSR)

32 位的 PSR 包含若干个域，这些域控制处理器的操作或保存状态信息。他们可以被保存,恢复,陷阱,或返回指令修改，特权指令 RDPSR 和 WRPSR 直接读写 PSR 寄存器。

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

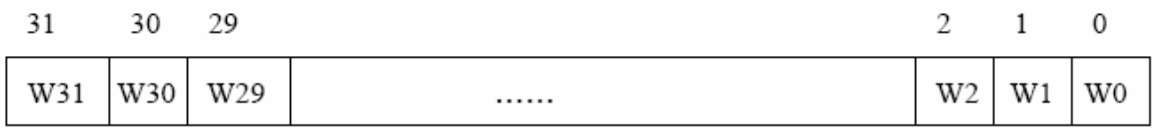
- impl 域：这四位为只读位，用来在硬件上确定，分类硬件体系架构的实现。
- ver 域：这四位也为只读位，标记 S698P 的版本号。
- icc 域：这四位标记 IU 的状态，他们可被以字母串 cc 结尾的算术逻辑指令或是 WRPSR 指令修改。Bicc 和陷阱指令可以引起基于此域的控制转移。此域的格式定义如下：



- n 域：此位指示累加器的运算结果是否要被忽略。1＝忽略，0＝不忽略。
- PSR_zero（z）域：对于累加器的最后一条修改了 icc 域的指令，此位指示是否累加器的运算结果为 0。1＝为零，0＝不为零。
- v 域：对于累加器的最后一条修改了 icc 域指令，此位指示累加器的结果是否在 32 位能表示的范围内。1＝溢出；0＝无溢出。
- c 域：对于累加器的最后一条修改了 icc 域的指令，此位指示是否指令的运算的最后一位有借位或是进位。1＝有，0＝没有。
- r 域：此六位是保留位。如果被 RDPSR 指令读，则返回 0。为了以后扩展兼容性，管理软件可以通过 WRPSR 向此域写入 0。
- EC 域：此位决定协处理器是否被激活。如果没被激活则会有一个与协处理器相关的陷阱。1＝支持，0＝不支持。如果硬件没实现协处理器，则此位始终为 0，这时对此位的写被忽略。
- EF 域：此位决定浮点单元(FPU)是否被激活。如果没被激活则会有一个与浮点单元(FPU)相关的陷阱。1＝支持，0＝不支持。如果硬件没实现浮点单元(FPU)，则此位始终为 0，这时对此位的写被忽略。
- PIL 域：此域指示 CPU 要接收中断的级别。
- S 域：此为指示 CPU 是处于管理状态还是用户状态。1＝管理状态，0＝用户状态。
- PS 域：此位保存当前陷阱所处的状态值。1＝管理状态，0＝用户状态。
- ET 域：此位指示是否允许有陷阱，如果当前有陷阱，则此陷阱会自动将此位置为 0。0＝忽略陷阱，这时若有中断请求则会被忽略，而且一个异常的陷阱会使 IU 执行挂起（halt）操作（此操作又触发重启陷阱）。1＝允许陷阱。
- CWP 域：此五位为当前寄存器窗口指针。当发生陷阱时或保存指令硬件会使 CWP 减少；恢复或返回指令会 CWP 增加。

窗口掩码寄存器（WIM）

WIM 由管理软件控制，被硬件使用来查看是否保存，恢复或返回指令会引起窗口上溢（overflow）或下溢（underflow）陷阱。



31	30	29		2	1	0
W31	W30	W29	……	W2	W1	W0

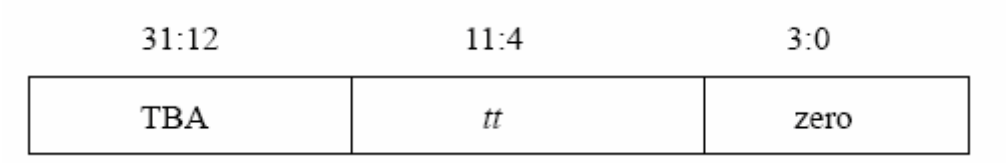
WIM[n]关联 CWP 为 n 时对应的窗口。

当保存，恢复或返回指令执行，当前 CWP 会和 WIM 比较，如果要转到的是一个非法窗口，（即相应的 WIM 位被置为 1），则会转入相应的窗口上溢（overflow）或下溢（underflow）陷阱。

WIM 可被特权指令 RDWIM 读取，被 WRWIM 指令写入。读那些硬件没实现的位会返回 0，对他们写无效。

陷阱基础寄存器（TBR）寄存器

此寄存器有三个域，每个域中都存放陷阱发生时要转向的地址。



31:12	11:4	3:0
TBA	*tt*	zero

- TBA 域：这 20 位存放陷阱地址表的高 20 位地址。这些位被管理软件维护，可以被 WRTBR 指令写入。
- tt 域：这 8 位存放陷阱的类型编码。当发生陷阱时，这 8 位被硬件写入，且一直保持此值到下一个陷阱到来。WRTBR 指令对此域没有影响。
- 0 域：这四位全为 0，不受 WRTBR 指令的影响。这四位留作以后扩展软件使用 WRTBR 时应向这四位写 0。

乘法/除法寄存器（Y）

这 32 位保存整数乘法的结果，相应的乘法指令包括 SMUL,SMULcc,UMUL,UMULcc 乘法指令或是含有 MULScc 指令的例程。这 32 位也可保存 SDIV,SDIVcc,UDIV,UDIVcc 等除法指令的双精度数。

此寄存器可以被 RDY，WRY 指令读或写。

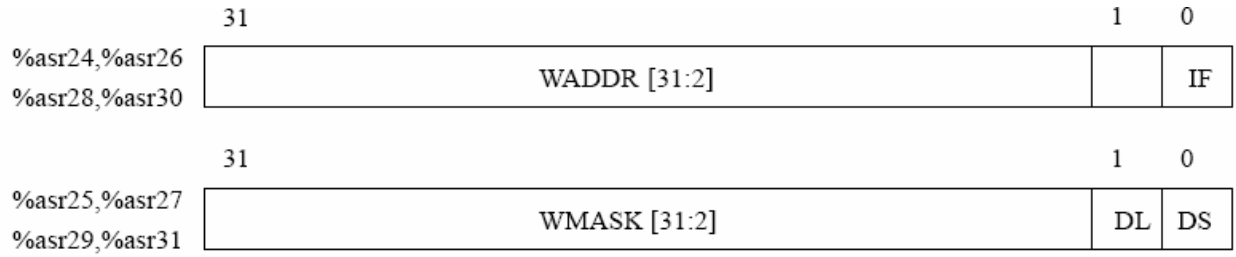
程序指针（PC，nPC）

32 位 PC 寄存器存放当前 IU 运行的指令的地址，nPC 寄存器存放下一个要运行的指令的地址（特指没发生的陷阱）。

PC 被调用和跳转指令读取。在一个陷阱中，PC 和 nPC 被写入到两个本地寄存器中。

监视点寄存器 ASR）

S698P4-II 使用四对 ASR 寄存器（% asr24/25, %asr26/27,%asr28/30,%asr30/31）实现了四个监视点（watch point）：



由 WADDR 域定义的地址范围都可以被监视，这个地址范围也可被 WMASK 域掩码（WMASK[x]=1）激活匹配。在一个监视点上，陷阱 0x0B 会被生成。通过设定 IF，DL 和 DS 位监视 hit 可以在取指，数据存储读写的时候发生。将此三位清 0 会有效的禁止此项功能。

2.4.2 浮点单元(FPU)寄存器

2.4.2.1 浮点单元 f 寄存器

在 S698P4-II 中，浮点单元有 32 个 32 位的 f 寄存器且被 f[0]－f[31]来标记。不像窗口化的 r 寄存器，在给定的时间内指令可以访问任意的 f 寄存器。F 存器可以被处理整数的指令或浮点数的指令（Fpop1/Fpop2 格式）访问。

注意：存储访问单精度/双精度浮点数的指令可以用来载入浮点操作数到 f 寄存器，如下例：

```
x: .float 20.000
y: .float 5.000
z: .float 0
.align 8
temp: .quad 0
.text
start:
set x,%r1
ld [%r1],%f2  ！ x->%f2
set y,%r4
ldd [%r4],%f3
ld [%r4],%fsr
fadds %f2,%f3,%f6
set temp,%r1
std %f6,[%r1]
set z,%r1
st %fsr,[%r1]
nop
```

nop

end: ta

双精度和四精度操作数

一个单精度的 f 寄存器内可存储一个单精度的操作数；双精度操作数需要一个编号分别位奇数和偶数的 f 寄存器对；四精度操作数需要一个含 4 个 f 寄存器的寄存器组。这样，在某一给定时间这些 f 寄存器可以保存最多 32 个单精度数，或 16 个双精度数，或 8 个四精度数。

要使用访问双精度的浮点数的指令得保证双精度的对齐规则：双精度寄存器地址的最低位要为 0 或被软件置为 0；相似的，四精度寄存器地址的最低两位要为 0 或被软件置为 0。

2.4.2.2 浮点单元控制状态寄存器

32 位的浮点单元控制/状态寄存器组包含即浮点状态寄存器（FSR），其内含有浮点单元的状态信息。

即浮点状态寄存器（FSR）

其内含有状态和模式信息，此寄存器被 STFSR 指令读,LDFSR 指令写。

RD	u	TEM	NS	res	ver	ftt	qne	u	fcc	aexc	cexc
31:30	29:28	27:23	22	21:20	19:17	16:14	13	12	11:10	9:5	4:0

RD 域：根据 ANSI/IEEE 标准（754－1985），此域选择浮点结果的逼近方向。

u 域：

第 29，28，12 位（从 0 数）不被使用而留作将来兼容扩展，软件应该只使用指令 LDFAR 对这些位写入 0 值。

TEM 域：

此域有 5 位，用来允许 5 类浮点处理异常。当发生某类的异常时，此寄存器的 cexc 域会指示发生的异常，如果 TEM 域的相应位又为 1，则会产生浮点异常的陷阱；反之若置为 0 就会屏蔽，使之不会产生陷阱。

NM 域：此位为 S698P4-II 保留位。

ver 域：这三位保存 S698P4-II 实现版本信息。

ftt 域：

此域保存浮点处理异常的类型。当一个浮点处理的异常产生之后，ftt 域保存异常的类型直到 STFSR 或 Fpop 被执行,ftt 域可以被 STFSR 指令读，LDFSR 指令对此域没有影响。若由管理模式的软件处理浮点陷阱，则其必须执行一个 STFSR 去确定浮点陷阱的类型。STFSR 是否会将 ftt 域清 0，由芯片设计时决定。如果 STFSR 不会将此域清 0，则处理陷阱的管理软件要保证在用户态时，后来执行 STFSR 域会返回 ftt 域的值为 0。

注意：尽管在返回用户态前执行了一个不会产生陷阱的 Fpop（此指令能将 ftt 清 0 如 fmovs %f0，%f0），因为 LDFSR 指令不会改变 ftt 域则其不可用来处理 ftt 域。Ftt 域在下一条 Fpop 指令执行前都是有效的。

在正常的运算过程中不希望出现硬件错误，它们在用户应用程序中都是不可恢复的。

相反，IEEE_745 异常、不完整的 Fpop 和未实现的 Fpop 错误偶尔会希望出现在正常的运算处理中。且可用管理软件恢复。当浮点类型的陷阱发生时，会有：

1） aexc 域的值不被改变。

2） cexc 域也不会改变，除了发生 IEEE_745 异常时的相应位被置位。不完整的 Fpop、未被实现的 Fpop 或是 sequence_error 不会影响此域。

3） 源 f 寄存器不会改变。（一般通过不改变目标 f 寄存器来实现）

4） fcc 域不会改变。

如果不完整的 Fpop 或未实现的 Fpop 不产生 IEEE 的陷阱，则处理恢复的软件得定义 cexc，aexc 或目标 r 寄存器，fcc。

Ftt＝IEEE745 异常：IEEE745 浮点异常由 ANSI/IEEE 标准定义。此类型在 cexc 域标记。注意：aexc，fcc，和目标 f 寄存器不受 IEEE745 异常陷阱的影响。

Ftt＝没有被实现 Fpop：这时说明浮点单元(FPU)的一个 Fpop 没有被实现。

编程提示：在上两种陷阱下，软件应该进行仿真和重新执行产生异常的指令，更新 FSR、目标寄存器和 fcc。

Ftt＝非正常错误：非正常错误指示三种浮点单元(FPU)非正常错误条件中的一种，它们由管理软件的错误操作引起：

1）执行一个没有“浮点延迟陷阱队列”（FQ）的 STDFQ 指令。

2）执行一个浮点单元(FPU)不能接受的指令。这种类型的队列错误源于管理软件的逻辑错误（这种逻辑错误已经产生了一个浮点陷阱），如前一个浮点陷阱执行完后浮点队列没有被清空。

3）当 FSR.qne＝0，即 FQ 为空时，执行 STDFQ 指令。（推荐产生队列错误，但这里不需要）

编程提示：如果在执行用户程序时，队列错误浮点异常发生，则恢复到正确完整的状态去继续执行用户程序往往是不可能的。

Ftt＝硬件错误：此错误表明浮点单元(FPU)监测到一个灾难性的内部错误（如非法状态或 f 寄存器的奇偶错误）。如果执行用户程序时发生这样的错误，就不可能恢复到正确的状态继续执行。

qne 域：

延迟的浮点异常 类的陷阱之后或执行完双精度浮点队列存储指令（STDFQ）之后，此位指示是否 FQ 为空。Qne＝0 时 FQ 为空；qne＝1 时 FQ 为非空。此位可被 STFSR 指令读，且不受 LDFSR 指令的影响。然而，执行一段 STDFQ 指令会使 FQ 变空（qne＝0）。如果设计芯片时没有实现 FQ，读此位会返回 0。管理软件必须要维护此位使得在用户态读此位时总返回 0。

fcc 域：

这两位包含浮点单元(FPU)的环境信息。这两位被浮点比较指令（FCMP 和 FCMPE）更新；被 STFSR 指令读；被 LDFSR 指令写。FBfcc 根据此域跳转。下表中 f rs1 和 f rs2 与指令的 rs1 和 rs2 域指定的 f 寄存器中的值有关。表中的问号指示无序的关系。注意：当 FCMP 或 FCMPE 产生 IEEE754 异常的陷阱时，fcc 就不会被改变。

aexc 域：

当浮点异常的陷阱被屏蔽时（利用 TEM 域），这 5 位累积 IEEE_745 定义的浮点异常。Fpop 指令执行完后，TEM 域和 cexc 域要被逻辑与。如果结果为非 0，则产生浮点异常的陷阱；否则 cexc 与 aexc 要被逻辑或且将其结果放入 aexc 域。因此，当陷阱被屏蔽，异常就会被积累在 aexc 域。

cexc 域：

这 5 位指示一个或更多的 IEEE_745 浮点异常（由最近执行过的 Fpop 指令产生）。对于没被产生的异常，相应位被清 0。

推荐：IEEE_745 浮点异常产生陷阱时，FSR.cexc 的某一位被置位。如果 Fpop 指令的异常产生了陷阱，而不是异常，那么 FSR.cexc 就不会被改变。

nvc，nva 域：

此位指示是否操作数非法。如 0÷0 等。1＝无效的操作数；0＝有效的操作

数。

ofc，ofa 域：

当结果比正常能表示的数大时，会发生向上溢出。1＝上溢，0＝无上溢。

ufc，ufa 域：

当结果比正常能表示的最小的数还小时会发生向下溢出。1＝下溢，0＝无下溢。

当结果是 0 时，就不能判断是否发生下溢。然而：

如果 UFM＝0：当运算结果要比寄存器能正常表示的数还要小时，ufc 和 ufa 位会被置为 1。Nxc 和 nxa 总是被置位。

如果 UFM＝1：当运算结果要比寄存器能正常表示的数还要小时，会产生一个陷阱。

dzc，dza 域：

X÷0，无论 X 位正常或低于正常的数，都不是合法的。此位指示是否发生 0 为除数的情况。1＝（0 为除数）；0＝（0 不为除数）。但是 0÷0 时 dzc 为不会被置位。

nxc，nxa 域：

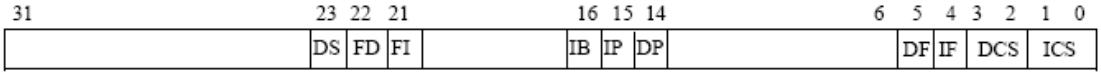
此域指示逼近的结果和精确的结果是否一致：1＝不一致，0＝一致。

2.4.3 缓存(cache)寄存器

2.4.3.1 缓存控制寄存器

指令和数据缓存的操作均通过一个公共的缓存控制寄存器(CCR)来控制。每

个缓存有三种模式：无效、使能和冻结。如果无效，没有缓存操作被执行，且装载和存储要求被直接送到存储器控制器。如果使能，缓存操作将按上述的描述操作。在冻结状态，就象其使能那样缓存读取被保持与主存同步但是读未命中时没有新块被分配。

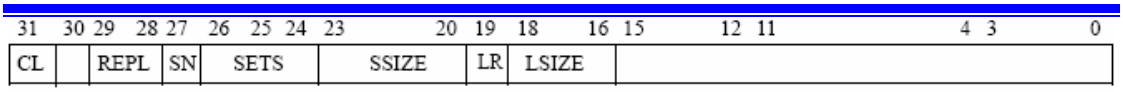


- [23]：数据缓存 侦听允许。如果置位, 将允许数据侦听
- [22]：刷新数据缓存。如果置位，将刷新数据缓存
- [21]：刷新指令缓存。如果置位，将刷新指令缓存
- [16]：指令突发预取。如果置位，将允许突发预取
- [15]：指令缓存刷新保持。当指令缓存刷新时，置位。
- [14]：数据缓存刷新保持。当数据缓存刷新时，置位。
- [5]： 中断时冻结数据缓存。当置位时，如果一个异步中断发生时，数据缓存自动冻结。
- [4]：中断时冻结指令缓存 。当置位时，如果一个异步中断发生时，指令缓存自动冻结。
- [3：2]： 数据缓存状态。表明当前缓存的状态：X0= 禁止，01 =冻结，11 = 允许。
- [1：0]：指令缓存状态。表明当前缓存的状态：X0= 禁止，01 =冻结，11 = 允许。

如果 DF 或 IF 位被设置，当一个异步中断发生时，相应的缓存将被冻结。这对允许更精确计算最坏情况执行时间的实时系统很有好处。

2.4.3.2 缓存配置寄存器

两个缓存的配置寄存器分别定义在两个寄存器中：指令和数据配置寄存器。这些寄存器是只读的。



- [31]：缓存锁
- [29：28]：缓存替换算法
- [27]：缓存侦听，支持缓存侦听
- [26：24]：缓存缓存替换单位
- [23：20]：替换单位大小
- [19]：没有使用
- [18：16]：缓存线大小
- [15：0] ：没有使用

2.5 陷阱(Trap)

表 2-8 陷阱分配及其优先级分配表

陷阱	TT	优先级	描述
复位	0x00	1	复位
写错误	0x2B	2	写缓冲错误
指令读/写错误	0x01	3	取指令错误
无效指令	0x02	5	未实现指令
越权指令	0x03	4	越权指令
FPU禁止	0x04	6	FPU 禁止
指令或数据观察点匹配	0x0B	7	指令或数据观察点匹配
窗口上溢	0x05	8	窗口上溢
窗口下溢	0x06	8	窗口下溢
地址不对齐	0x07	10	地址不对齐

FPU 异常	0x08	11	FPU 异常
指令装载错误	0x09	13	指令装载错误
有符号数算术溢出	0x0A	14	有符号数算术溢出
被零除	0x2A	15	被零除
中断级别 1	0x11	31	保留
中断级别 2	0x12	30	UART 2
中断级别 3	0x13	29	UART 1
中断级别 4	0x14	28	GPI 中断 1
中断级别 5	0x15	27	GPI 中断 2
中断级别 6	0x16	26	GPI 中断 3
中断级别 7	0x17	25	GPI 中断 4
中断级别 8	0x18	24	定时器 1
中断级别 9	0x19	23	定时器 2
中断级别 10	0x1A	22	保留
中断级别 11	0x1B	21	保留
中断级别 12	0x1C	20	以太网
中断级别 13	0x1D	19	CAN
中断级别 14	0x1E	18	保留
中断级别 15	0x1F	17	保留
陷阱指令	0x80-0xFF	16	软件陷阱指令 (TA)

3、 AMBA 总线和片上外设

3.1 AMBA 总线

S698P4-II 内部的AMBA总线包括2种总线：AHB和APB。APB总线用来访问片内外设的寄存器；AHB总线用作高速数据传输。

3.1.1 AHB 总线

S698P4-II 用AMBA-2.0 AHB总线连接处理器缓存控制器和存储器控制器、其它的高速单元，4个IU都是总线上的主控单元。另外，还有其它从属单元连接在总线上：存储器控制器、AHB/APB转换桥，CAN总线控制器。表3-1列出了S698P4-II AHB总线的地址分配。

表3-1 S698P4-II 总线地址分配

地址空间	描述	
0x00000000—0x1FFFFFFF	ROM 区	512M
0x20000000—0x3FFFFFFF	I/O 区	512M
0x40000000—0x5FFFFFFF	存贮器区	512M
0x60000000—0x7FFFFFFF	动态随机存贮器区	512M
0x80000000－0x8FFFFFFF	片上外设	256M
0x90000000 – 0xa0000000	Debug Support Unit	512M
0xb0000000 – 0xfffbffff	保留	1280M
0xFFFC0000－0xFFFC0100	CAN 控制器	256B

3.1.2 APB 总线

AHB/APB转换桥作为一个从属设备连接在AHB总线上，是APB总线唯一的主控单元。处理器通过AHB/APB桥访问大部分片内外设。片内外设包括：

- GPIO（8位）

- 2路UART
- 2个定时器
- 32位看门狗
- 中断控制器
- 以太网
- 1553B控制器
- 存储器控制器
- CAN

S698P4-II　APB 总线地址空间从 0x80000000 到 0x8FFFFFFF。

3.2 片上外设

3.2.1 存储器控制器

3.2.1.1 存储器地址分配

存储器控制器控制一个连有 PROM, 存储器映 I/O 设备, 静态存贮器和动态随机存贮器的存储器总线。控制器作为 AHB 总线上的从设备。存储器控制器的功能通过 APB 总线访问存储器控制寄存器 1,2 和 3(MCFG1, MCFG2&MCFG3)进行配置。存储器总线支持四种类型的设备：prom，静态存贮器, 动态随机存贮器 和 I/O。prom，静态存储器总线只支持 32 位模式，而 I/O 可以配置为 8、16、32 位模式。

表 3-2 存储器控制器地址分配表

地址空间	描述	
0x00000000—0x1FFFFFFF	ROM 区	512M
0x20000000—0x3FFFFFFF	I/O 区	512M
0x40000000—0x5FFFFFFF	存贮器区	512M
0x60000000—0x7FFFFFFF	动态随机存贮器区	512M

3.2.1.2 PROM

处理器提供两个 prom 片选择信号 ROMSN[1：0]。Prom 的访问和存贮器的访问基本相似。

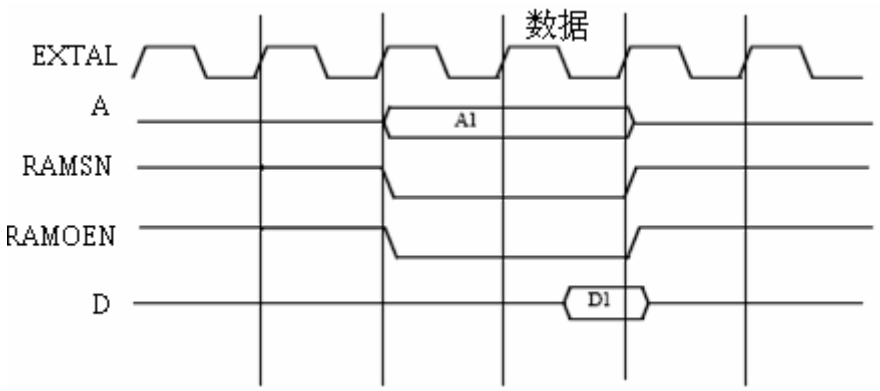


图 3-1 prom 读周期（0 等待）

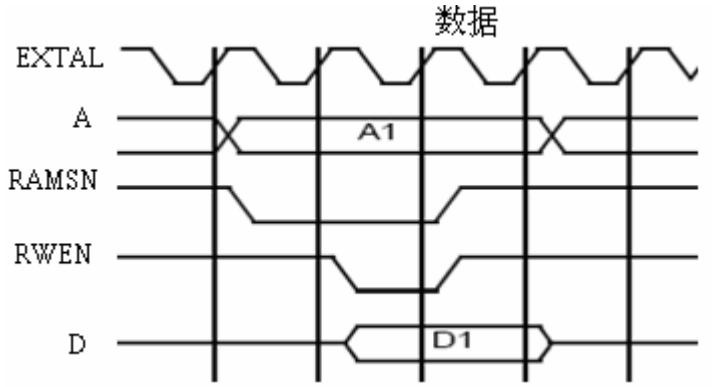


图 3-2 prom 写周期（0 等待）

3.2.1.3 静态存贮器(SRAM)

静态存贮器区域最大可支持 512M 字节空间，具有 5 个存贮器块，每块的大小在 MCFG2[12：9]配置，可以设置从 8 千字节到 256 兆字节。静态存贮器读访问包括两个数据周期和 0-15 个等待周期。在非连贯的访问中，在一个读周期后增加一个前导输出的周期，可以阻止由于存储器或者 I/O 设备的关闭时间引起的总线竞争。图 3-3、3-4 显示了基本的读/写周期波形（0 等待周期）。

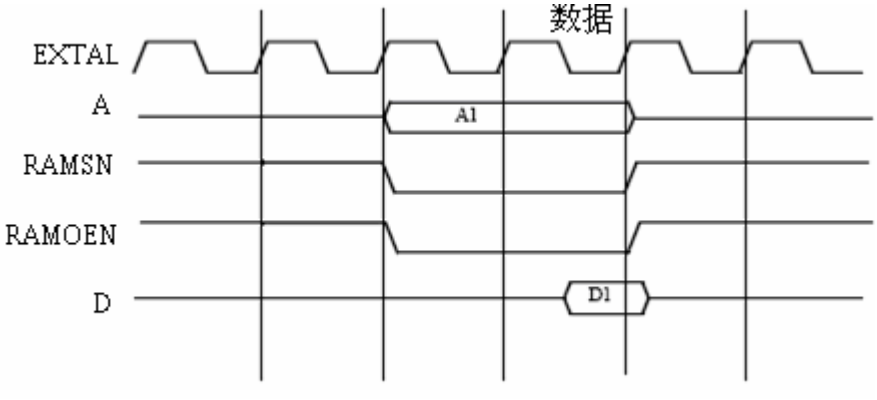


图 3-3 静态存贮器读周期（0 等待）

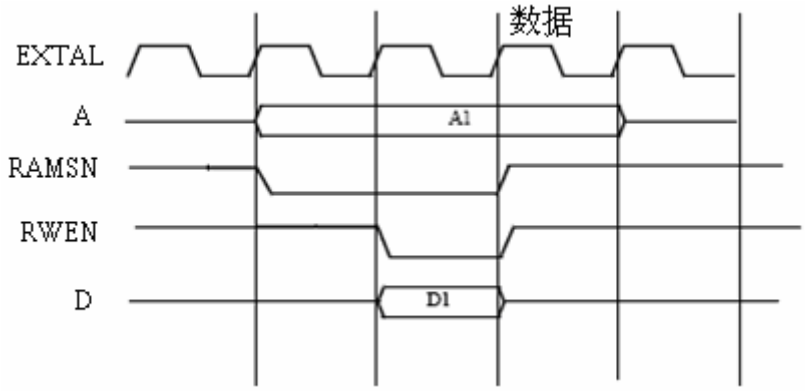


图 3-4 静态存贮器写周期（0 等待）

3.2.1.4 动态随机存贮器(SDRAM)

动态存贮器访问支持两块 PC100/PC133 兼容的设备。动态随机存贮器控制器支持带有 8-12 个列地址位，并且最高 13 行的动态随机存贮器。通过 MCFG2 和 MCFG3（见下文）控制动态随机存贮器的操作。只支持 32 位的数据总线宽度，每个块的大小可以编程在 4 M 字节和 512M 字节之间。为了对不同的动态随机存贮器器件（在不同的频率）提供最优的访问周期，一些动态随机存贮器的参数可以在存储器配置寄存器 2（MCFG2）中设定。可设定的动态随机存贮器参数可以在表 3-3 中看到。

表 3-3 动态随机存贮器可编程的时序参数

功能	参数	范围	单位
CAS 延迟, RAS/CAS 延迟	tCAS, tRCD	2 - 3	时钟

充电活动	tRP	2 - 3	时钟
自刷新周期	tRFC	3 - 11	时钟
自刷新周期参数		10 - 32768	时钟

动态随机存贮器的时序参数设置可参见 PC100/PC133 的动态随机存贮器设置例子。

表 3-4 动态随机存贮器参数设置

动态随机存贮器设置	tCAS	tRC	tRP	tRFC	tRAS
100 MHz, CL=2; TRP=0, TCAS=0, TRFC=4	20	80	20	70	50
100 MHz, CL=3; TRP=0, TCAS=1, TRFC=4	30	80	20	70	50
133 MHz, CL=2; TRP=1, TCAS=0, TRFC=6	15	82	22	67	52
133 MHz, CL=3; TRP=1, TCAS=1, TRFC=6	22	82	22	67	52

3.2.1.5 输入/输出(I/O)

输入输出读写等待周期也可在 MCFG1 中设定访问，可插入一个额外的等待周期（0-15 个周期）。处理器通过 MCFG1 决定 I/O 的数据总线宽度是 8 位、16 位、还是 32 位。

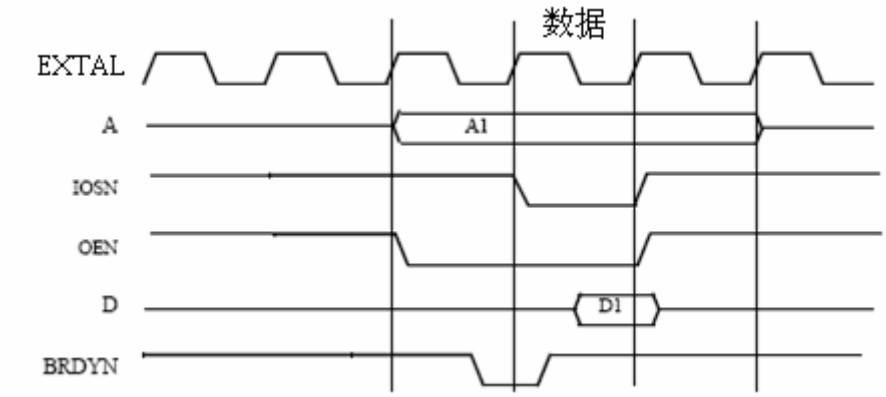


图 3-5 输入/输出读周期（0 等待）

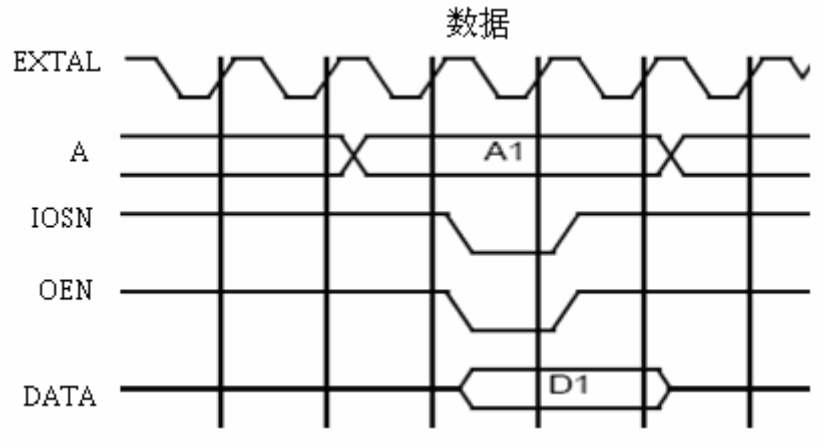


图 3-6 输入/输出写周期（0 等待）

3.2.2 中断控制器

S698P4-II 中的 AMBA 系统提供一个中断方案，中断线排成一行连同剩余的 AHB/APB 总线信号线，形成一个中断总线。来自 AHB 和 APB 单元的中断通过总线连结在一起被发送。多处理器中断控制器附属到 AMBA 总线，作为一个 APB 从设备，而且监视组合的中断信号。

在中断总线上产生的中断全部被转送给中断控制器。 中断控制器通过优先级区分，中断屏蔽选择，把最高优先级的中断送给处理器。

中断监视器监视中断总线中的 1-15 个中断。通过设置中断电平寄存器，每个中断可以被指定两个电平（0 或者 1）。电平 1 中断的优先级比水平 0 中断的优先级高。而每个电平里面的中断也是有优先级区分的，中断 15 有最高的优先级，中断 1 优先级最低。电平 1 的优先级最高中断将会被转送到处理器。 如果电平 1 没有非屏蔽挂起中断存在，来自电平 0 的最高非屏蔽挂起中断将会转送到处理器。

当多个处理器单独屏蔽和转送时候，中断在系统电平上有区分。 多处理器系统的每个处理器有单独的中断屏蔽和强制寄存器。 当一个中断在中断总线上被告知的时候，中断挂起寄存器的相应位置 1，跟着中断信号发送到每个 CPU 屏蔽寄存器，为每个 CPU 进行中断屏蔽，然后对中断进行优先级选择，把优先级高的中断送到 CPU。

当有一个 CPU 应答中断后，对应的中断挂起位将会自动地被清除的。中断也可以通过

设置中断强制寄存器强制产生对应的中断。在这情况，处理器应答后将清除强制位，而并非挂起的位。复位之后，中断屏蔽寄存器全部被设定成零，剩余的控制寄存器是不确定的。注意：中断 15 能被 S698P4-II 处理器屏蔽，使用时要小心—大部分操作系统不能正确地处理这个中断。

S698P4-II 中断控制器把S698P4-II 内部和外部的所有中断按照优先级先后的顺序排列，并传送给IU。S698P4-II 总共有15个中断, 15个中断如下表所示：

表 3-5 系统中断列表

中断	中断源
15	-
14	-
13	CAN 中断
12	以太网中断
11	-
10	-
9	定时器 2 中断
8	定时器 1 中断
7	外部中断 3（GPIO7）
6	外部中断 2（GPIO6）
5	外部中断 1（GPIO5）
4	外部中断 0（GPIO4）
3	UART2 中断
2	UART1 中断
1	AHB 总线错误中断

3.2.3 通用输入/输出接口(GPIO)

具有 8 个通用输入输出端口，而且提供可选择的中断支持。在通用输入/输出端口的每个位能通过设置方向寄存器，分别设定成输入或者输出功能，而且能选择地产生中断，通

用输入输出端口 4/5/6/7 位对应外部中断 0/1/2/3，可以设置为高电平/低电平/上升沿/下降沿 4 种触发方式。

通过设置通用输入输出端口方向寄存器，通用输入输出端口可以设置为输出/输入功能，“1”表示输出，“0”表示输入。通过设置通用输入输出端口数据寄存器，通用输入输出端口可以设置对应的通用输入输出端口脚为输出功能，“1”表示输出高电平，“0”表示输出低电平。

3.2.4 串口(UART)

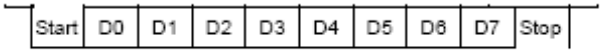
通用异步串行接口提供串行通讯功能，用来跟外部设备通讯。

S698P4-II 提供了 2 个串口接口，其中串口 1 & 2 控制器的结构和功能完全一样，是简单的串口，只包括 TXD 和 RXD 两根信号线。该接口提供串行通信。串口支持 8 数据位、一个可选的校验位和一个停止位的数据帧。为了生成比特速率，每个串口有一个 12 位的可编程时钟分频器。有两个先进先出用于总线和串口之间的数据传输。有两个保持寄存器用于总线和串口之间的数据传输。

发送操作

通过设置“串口控制寄存器”的“TE”位来使能发送操作。当写入发送数据时，数据从“发送保持寄存器”送到“发送移位寄存器”，被转换成串行数据流，通过TXD引脚输出。串口控制器自动在8位数据的前面加上1位起始位，在其后面加上1位可选的奇偶校验位和1位停止位。如下图所示：

数据帧 ，无校验位：



数据帧 ，有校验位：

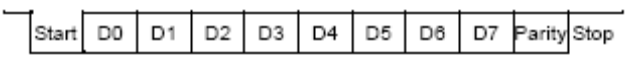


图 3-11　串口数据帧

如果“发送保持寄存器”中没有新的字符，则输出引脚TXD保持高电平，“串口控制寄存器”中的TSRE（发送移位寄存器空）位被置‘1’。当一个新的字符被载入“发送保持寄存器”中，则发送重新开始，TSRE位被清‘0’。当发送被禁止时，发送操作继续进行，直至当前正在发送的字符发完为止。当发送被禁止时，“发送保持寄存器”不能载入数据。

接收操作

通过设置“USART控制寄存器”中的RE（接收使能）位来使能接收操作。当接收器查找到一个从高到低的跳变，表示一个start开始位。”半位”周期后开始接受真正发送过来的原始数据。每接受一位信息后延时一个输入terval，直到接收到stop位。每接受一位，接受移位寄存器就移位，接受完毕后其内的值会保持到新的传送到来为止。

在接收期间，最低有效位最早接收。然后数据被送入接收保持寄存器（RHR），USART状态寄存器中的“数据准备好”位（DR位）即数据 ready位被置位。其他的状态位也在这时被刷新。如果接收保持寄存器或接收移位寄存器中有某一位没有保存，但这时又有了新的传送发生，那么接收移位寄存器中的数据会丢失，串口状态寄存器中的过载状态位会被置位。如果“流控制”（flow control）被激活；则当已接收到开始且接收保持寄存器中还有没被读取的位，RTS*会被置高。当接收保持寄存器被读取，RTS*会被自动重新激活。

波特率设置

每个串口串口都有一个倒计时计数器来产生需要的波特率，计数器的时钟来源于系统时钟且当发生溢出时会发出中断信号。溢出后计数器的值会被串口的重加载寄存器更新。中断的频率应该是所要波特率的8倍。

分频值 = (((hclk * 10) / (波特率 * 8)) – 5) / 10

3.2.5 定时器

系统平台提供了 2 个 32 位定时器，提供硬件定时中断，分别为定时器 1、2。其中，定时器 2 可做为看门狗使用。

通用定时器应用在一个预置数寄存器和 1- 7 个递减计时器。通用定时器作为 AMBA APB 总线从设备。定时器能够在定时器下溢出时断言中断。

预置数寄存器被系统时钟控制而且在每个时钟周期上递减。当预置数寄存器下溢出, 它会从预置数重载寄存器重新装载，同时产生一个定时器滴答（tick）。 定时器共享递减器以保存区域。 在下一个定时器滴答时，下个定时器会以等价于(预置数重载寄存器值 +1) 的有效衰减速率逐渐减少。

控制寄存器控制定时器的操作。一个定时器通过设置控制寄存器中使能位使能操作。然后，定时器计数起的值在每个预置数滴答后渐减。 当一个定时器计数器下溢出, 如果控制器中的复位位为 1 时，它将会自动地重载对应的定时器重载寄存器中的值；否则，它将会停止在 -1 而且复位使能位。

定时器做为一个中断。当具有中断使能的任一个定时器计数器下溢出时，定时器单元将会产生共享中断。如果配置成每个定时器发出各自的中断信号，定时器单元将会在一个定时器单元下溢出时，向适当的线上发送中断信号(如果给现在的定时器的中断使能位被置位)。 在下溢出定时器的控制寄存器的中断未决位将置位，并保持直到通过写'0'清除。

定时器分享相同的递减器。通过设定控制寄存器的链接位，定时器 n 能够与前一个定时器 n-1 配合使用。 当定时器 n-1 下溢出时，定时器 n 计数器值减 1。

当向控制寄存器的载入位写‘1’时，每个定时器都会重载重载寄存器中的值。

定时器 2 可以当作看门狗使用，当定时器 2 计数器下溢出时，就驱动一个看门狗输出信号。

注意：当定时器 2 作看门狗使用时，不能把定时器 2 作其他用途。

3.2.6 以太网（ethernet）

S698P4-II 的以太网控制器提供一个符合 10M/100M 以太网标准的接口。它支持半双工和全双工的 10/100 M 速度。 AMBA 接口包括一个用来配置和控制的 APB 接口，还有一个 AHB 主设备接口，该接口用来处理数据流。此数据流通过 DMA 通道处理。有一个 DMA 引擎用来做发送器，与之对应，还有一个 DMA 引擎作为接收器。两者共享相同的 AHB 主设备接口。以太网接口支持媒体独立接口（MII）接口，他们用来连接外部的 PHY。以太网控制器 通过对 MII 管理接口的访问来配置 PHY 。

同时，以太网控制器也提供了对于用来支持以太网调试通信链接（EDCL）协议的可选择部件。远程调试基于 UDP/IP 基础协议 。

以太网控制器 由 3 个功能的单元组成： DMA 通道, MDIO 接口和以太网调试通信链路(EDCL)。

主要功能是 DMA 通道，它在 AHB 总线和以太网之间传输数据。DMA 通道有一个发射器 DMA 通道和一个接收器 DMA 通道。 DMA 通道的操作是通过 APB 接口访问寄存器控制。

MDIO 接口用来访问在一个或多个的连接到 MAC 的 PHY 的配置和状态寄存器。 这一个接口的操作也通过 APB 接口控制。

太网调试通信链路（EDCL）经过以太网络提供到 AHB 总线的读和写访问。它使用 UDP ， IP, ARP 等用户应用层协议。太网调试通信链路（EDCL）包含不可访问的寄存器，和 DMA 通道并行运行。

媒体独立接口 (MII) 用作与 PHY 通讯。使用媒体独立接口（MII） 接口，以太网发送器在以太网上发送来自 AHB 领域的所有数据。相对而言，以太网接收机存储通过 AHB 总线的来自以太网的所有数据。当传输数据流时，这两个接口都使用先进先出。

3.2.7 CAN 总线控制器

3.2.7.1 概要说明

CAN（Controller Area Network，控制器局域网）总线控制器是依据 CAN 2.0B 串行通讯协议来设计的控制器。该控制器能有效地支持高安全等级的分布实时控制。其应用范围很广，从高速的网络到低价位的多路接线都可以使用，尤其是在汽车电子领域，CAN 总线已经被众多汽车厂商用作车身各部件之间的通信网络。

S698P4-II 的 CAN 总线控制器完整地实现了 CAN 2.0B 协议，功能兼容 Philips 公司的 SJA1000 芯片，具有相同的寄存器映射。CAN 总线控制器支持 BasicCAN 和 PeliCAN 模式。在 PeliCAN 模式下，支持 CAN2.0B 的扩展特征。BasicCAN 和 PeliCAN 两种模式可以通过时钟分频寄存器选择。

CAN 总线控制器的结构框图如下：

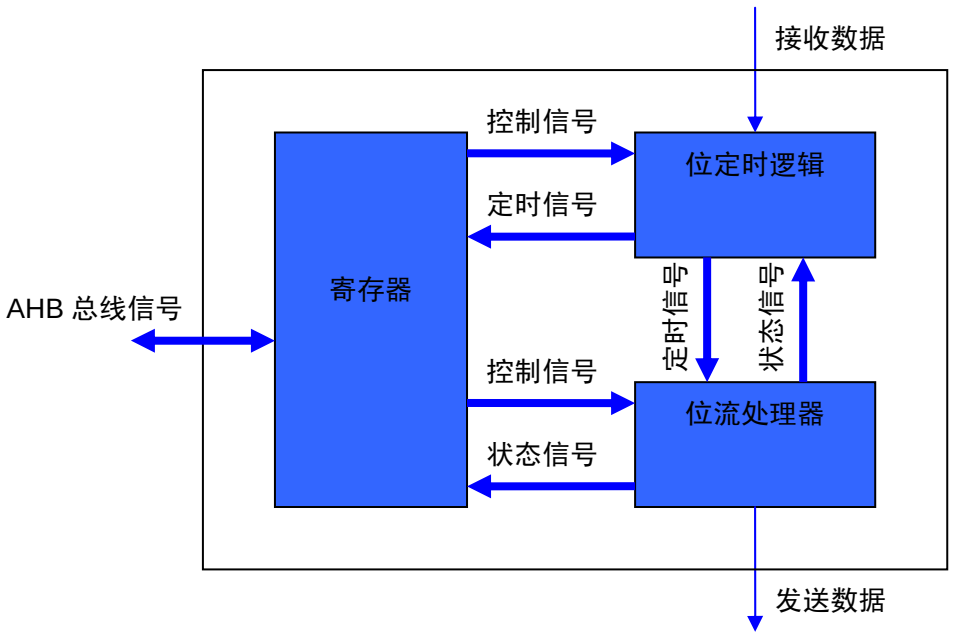


图 3-15 S698P4-II　CAN 总线控制器结构框图

3.2.7.2 主要特征

- PCA82C200 模式（即默认的 BasicCAN 模式）
- 扩展的接收缓冲器（64 字节先进先出 FIFO）
- 和 CAN2.0B 协议兼容
- 同时支持 11 位和 29 位标识符
- 位速率可达 1Mbits/s
- PeliCAN 模式扩展功能
  - 可读/写访问的错误计数器
  - 可编程的错误报警限制
  - 最近一次错误代码寄存器
  - 对每一个 CAN 总线错误的中断
  - 具体控制位控制的仲裁丢失中断
  - 单次发送（无重发）
  - 只听模式（无确认、无活动的出错标志）
  - 支持热插拔（软件位速率检测）
  - 验收过滤器扩展（4 字节代码，4 字节屏蔽）
  - 自身信息接收（自接收请求）

3.2.8 1553B 总线控制器

3.2.8.1 概要说明

S698P4-II 中的 1553B 总线控制器是依据 1553B 总线协议(1553A/B Notice2 协议)和参考 DDC-ACE 系列芯片（主要是 61580） 设计的总线控制器。 1553B 总线控制器主要包括 1553B 通信协议模块、配置寄存器模块、外部接口模块和存储管理模块等。其中 1553B 协议部分用差分曼彻斯特编码实现时分命令响应式串行通讯，主要包括 BC、RT 收发器等；配置寄存器主要实现对 1553B 总线控制器功能的配置，能够间接反映 1553B 总线控制器的功能；外部接口为 1553B 总线控制器与 CPU 的接口，用来实现 CPU 对 1553B 总线控制器

的控制；存储管理为 CPU 和 1553B 总线控制器之间交互数据的管理方式。

3.2.8.2 主要特征

- 通过硬件逻辑方式完全实现 MIL-STD-1553B 标准(国军标 GJB289A-97 标准);
- 操作方式、寄存器设置以及存储器布局等方面同 BU-61580 兼容；
- 支持的通讯类型包括：
  - BC → RT；
  - RT → BC；
  - RT → RT；
  - Broadcast；
  - Mode code；
- 能被配置为 BC、RT、BM 三种类型的控制器；
- 带 4K*16Bit 的集成双口 RAM；
- 与主机接口模式为通用的异步接口；
- 芯片内部集成了总线收发器，不需外部再接收发器;
- 带 A、B 双冗余通道；
- BC 性能：
  - 支持 A/B 区域；
  - 具有自动重发功能；
  - 可编程的消息间隔时间；
  - 帧自动重复发送；
  - 可编程的超时响应时间；
- RT 性能：
  - 可编程的 RT 地址，子地址；
  - 支持单缓冲存储器管理方式；
  - 支持循环缓冲存储器管理方式；
  - 支持双缓冲存储器管理方式；
  - 可编程的非法命令表；
  - 可编程的方式代码中断表；
  - 可编程的子地址忙表；
- BM 性能：
  - 能够实时侦听总线上的数据流，可以将所有的数据流记录下来，也可以有选择地进行数据监听；
  - 支持命令堆栈半满、全满溢出；

- 支持数据堆栈半满、全满溢出；
- 命令堆栈与数据堆栈独立；
- 对每条消息有相应的属性标志；

3.3 寄存器描述

表 3-6 系统寄存器地址分配表

地址	读/写	说明
0x80000000	R/W	存储器配置寄存器1
0x80000004	R/W	存储器配置寄存器 2
0x80000008	R/W	存储器配置寄存器 3
0x80000100	R/W	UART1 数据寄存器
0x80000104	R	UART1 状态寄存器
0x80000108	R/W	UART1 控制寄存器
0x8000010C	R/W	UART1 分频寄存器
0x80000200	R/W	中断水平寄存器
0x80000204	R	中断挂起寄存器
0x8000020C	R/W	中断清除寄存器
0x80000210	R/W	多处理器状态寄存器
0x80000240	R/W	处理器 0 中断屏蔽寄存器
0x80000244	R/W	处理器 1 中断屏蔽寄存器
0x80000248	R/W	处理器 2 中断屏蔽寄存器
0x8000024C	R/W	处理器 3 中断屏蔽寄存器
0x80000280	W	处理器 0 中断强制寄存器
0x80000284	W	处理器 1 中断强制寄存器
0x80000288	W	处理器 2 中断强制寄存器
0x8000028C	W	处理器 3 中断强制寄存器
0x80000300	R/W	预置数
0x80000304	R/W	预置数重载值
0x80000308	R/W	配置寄存器

0x8000030C	-	未使用
0x80000310	R/W	定时器 1 计数寄存器
0x80000314	R/W	定时器 1 重载值寄存器
0x80000318	R/W	定时器 1 控制寄存器
0x8000031C	-	未使用
0x80000320	R/W	定时器 2 计数寄存器
0x80000324	R/W	定时器 2 重载值寄存器
0x80000328	R/W	定时器 2 控制寄存器
0x80000900	R/W	UART2 数据寄存器
0x80000904	R	UART2 状态寄存器
0x80000908	R/W	UART2 控制寄存器
0x8000090C	R/W	UART2 分频寄存器
0x80000b00	R/W	I/O 数据输入寄存器
0x80000b04	R/W	I/O 数据输出寄存器
0x80000b08	R/W	I/O 方向寄存器
0x80000b0C	R/W	I/O 中断屏蔽寄存器
0x80000b10	R/W	I/O 中断极性寄存器
0x80000b14	R/W	I/O 中断边沿寄存器
0x80000f00	R/W	以太网控制器控制寄存器
0x80000f04	R/W	以太网控制器状态/中断源寄存器
0x80000f08	R/W	以太网控制器 MAC 地址 MSB
0x80000f0C	R/W	以太网控制器 MAC 地址 LSB
0x80000f10	R/W	以太网控制器 MDIO控制/状态寄存器
0x80000f14	R/W	以太网控制器 发送描述符指针
0x80000f18	R/W	以太网控制器 接收器描述符指针

备注：CAN 总线控制器地址请参考 CAN 控制器寄存器描述部分。

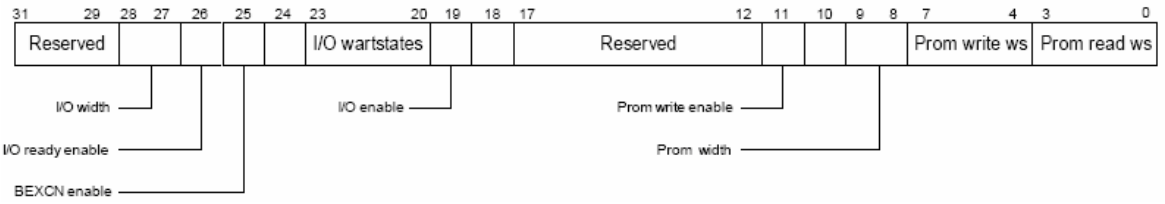
3.3.1 存储器配置寄存器

存储器控制器通过寄存器列表如下。

表 3-7 存储器控制寄存器

APB 地址	寄存器
0x80000000	存储器配置寄存器（MCFG1）
0x80000004	存储器配置寄存器（MCFG2）
0x80000008	存储器配置寄存器（MCFG3）

3.3.1.1 存储器配置寄存器 1(MCFG1)



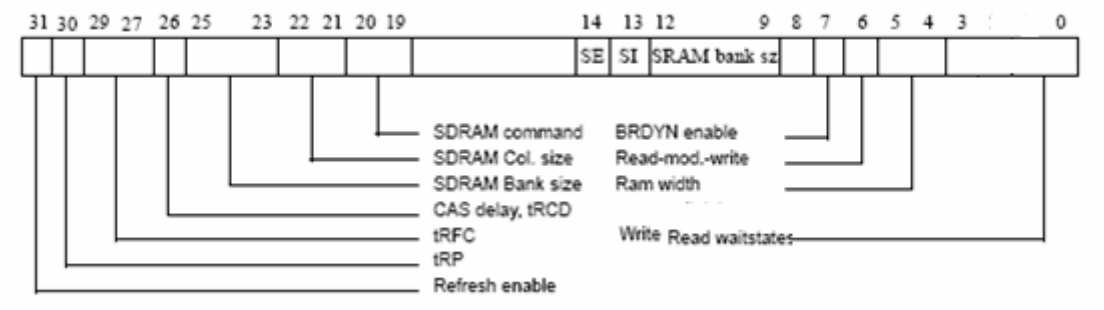
- [3：0]：Prom read ws（PROM读等待周期）。配置PROM读周期时等待周期的值。(“0000”=0, “0001”=1,… “1111”=15)；
- [7：4]：Prom write ws（PROM写等待周期）。配置PROM写周期时等待周期的值。(“0000”=0, “0001”=1,… “1111”=15)；
- [9：8]：Prom width(PROM宽度)。配置PROM数据总线的宽度(“10”=32位)，只支持32位；
- [10]：保留；
- [11]：Prom write enable（PROM写使能）。如配置为‘1’，则使能写PROM功能；
- [17：12]：保留；
- [19]：I/O区访问使能。如配置为‘1’，则I/O区访问使能；
- [23：20]：I/O waitstates（I/O区访问等待周期）。配置访问I/O区时等待周期的值

(“0000”=0,“0001”=1,“0010”=2,…,“1111”=15);

- [25]：Bus 错误(BEXC*) enable（总线错误使能）;
- [26]：Bus ready(BRDY*) enable（总线准备好使能）;
- [28： 27]： I/O bus width （I/O 总线宽度）。配置 I/O 区数据总线的宽度(“00”=8,“01”=16,“10”=32);
- [31：29]：保留;

3.3.1.2 存储器配置寄存器 2(MCFG2)

存储器配置寄存器 2 用于控制静态存贮器和动态随机存贮器的计时器。



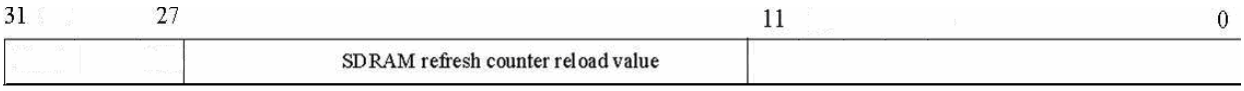
- [3：0]：存贮器 读/写等待周期.定义在存贮器读/写等待周期的数目 (“0000”=0, “0001”=1… “1110”=14, “1111”=15)
- [5：4]：存贮器 位宽. 定义存贮器的数据宽度(“1X”= 32位)，只支持32位
- [6]：读-修改-写。需要置为“1”
- [7]： 总线准备使能. 如果设定，将使能存贮器区域BRDYN
- [12：9]：存贮器 块大小.定义每个存贮器块的大小 (“0000”=8 千字节, “0001”=16 千字节... “1111”=256 兆字节).
- [13]：SI – 静态存贮器 无效.如果与14位一起设定，静态存贮器访问将被禁止
- [14]：SE - 动态随机存贮器 使能. 如果设定，动态随机存贮器控制器将被使能
- [20：19]：动态随机存贮器命令.写一个非零值将产生一个动态随机存贮器命令： “01”=预充电,“10”=自刷新, “11”=加载命令寄存器。 这个区域在命令执行后复位
- [22：21]：动态随机存贮器 列大小. 当位[25：23]= “111”时,“00”=256, “01”=512,

“10”=1024, “11”=4096,否则为2048

- [25：23]：动态随机存贮器块大小.定义对动态随机存贮器 片选的块大小： “000”=4 兆字节, “001”=8 兆字节, “010”=16 兆字节 .... “111”=512 兆字节
- [26]：动态随机存贮器 CAS 延时.选择 2 或3 个周期的CAS 延时(0/1). 当修改时，将发出一个加载命令寄存器命令。同样也设置RAS/CAS的延时。
- [29：27]：动态随机存贮器 TRFC定时。 tRFC将等待3+区域值的系统时钟
- [30]： 动态随机存贮器 TRP 参数。 tRP将等于2或3个系统时钟（0/1）
- [31]:：动态随机存贮器刷新.如果设定，动态随机存贮器刷新新将被使能

3.3.1.3 存储器配置寄存器 3(MCFG3)

MCFG3 包括用于动态随机存贮器 刷新计数器的重载入值。



- [31：27]： 保留
- [26：12]： 动态随机存贮器更新计数器重载值
- [11：0]： 保留

每个自刷新命令之间的周期按照如下公式计算：

刷新时间＝（动态随机存贮器更新计数器重载值＋1）/ HCLK

3.3.2 中断寄存器

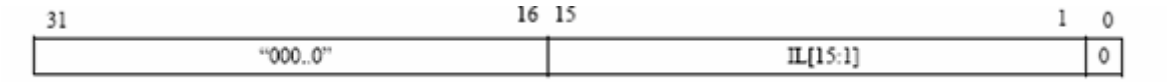
S698P 通过映射到 APB 地址空间的寄存器进行编程。

表 3-8 中断控制寄存器

APB 地址	寄存器

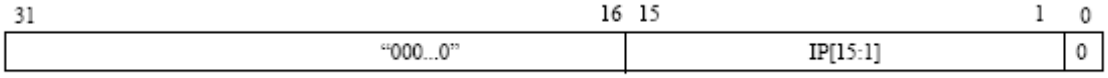
0x80000200	中断水平寄存器
0x80000204	中断挂起寄存器
0x8000020C	中断清除寄存器
0x80000210	多处理器状态寄存器
0x80000240	处理器 0 中断屏蔽寄存器
0x80000244	处理器 1 中断屏蔽寄存器
0x80000248	处理器 2 中断屏蔽寄存器
0x8000024C	处理器 3 中断屏蔽寄存器
0x80000280	处理器 0 中断强制寄存器
0x80000284	处理器 1 中断强制寄存器
0x80000288	处理器 2 中断强制寄存器
0x8000028C	处理器 3 中断强制寄存器

3.3.2.1 中断水平寄存器



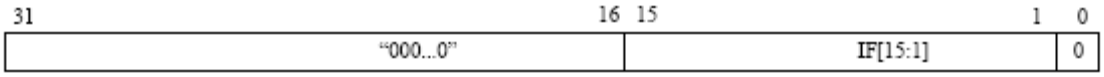
- [31：16] 保留，读为0。
- [15：1] 中断水平n (IL[n])： 中断n的中断水平，例如：can中断源连接到13,则can中断号为13，如果IL[13]为0，则can中断水平为0，寄存器[13]为1，则can中断水平为1。
- [0] 保留，读为0。

3.3.2.2 中断挂起寄存器



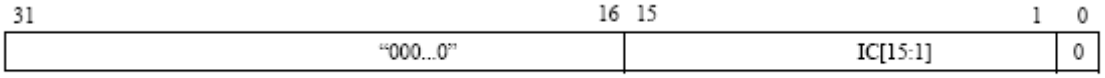
- [31：16] 保留，读为0。
- [15：1] 中断挂起n (IP[n])： 中断n的是否中断挂起。
- [0] 保留，读为0。

3.3.2.3 中断强制寄存器



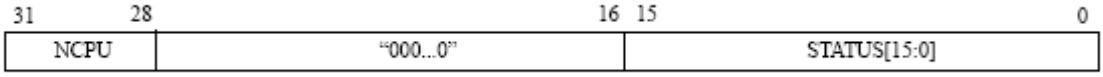
- [31：16] 保留，读为0。
- [15：1] 中断强制n (IF[n])： 如IF[n]设置位‘1’，则中断控制器强制产生相应的中断n。
- [0] 保留，读为0。

3.3.2.4 中断清除寄存器



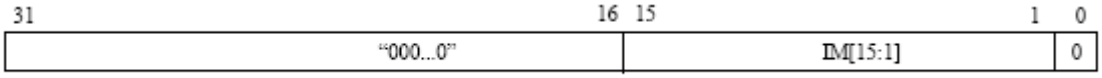
- [31：16] 保留，读为0。
- [15：1] 中断清除n (IC[n])： 写1到IC[n]，则中断悬挂寄存器中的相应位被清除。
- [0] 保留，读为0。

3.3.2.5 多处理器状态寄存器



- [31：28]NCPU：该值设置为3
- [27：16]保留位。
- [15：1]处理器停止状态位：‘1’= 关机, ‘0’= 运行。写‘1’进入运行状态

3.3.2.8 处理器中断屏蔽寄存器



- [31：16] 保留，读为0。
- [15：1] 中断屏蔽n (IM[n])： 写1到IM[n]，允许中断n，否则中断屏蔽。
- [0] 保留，读为0。

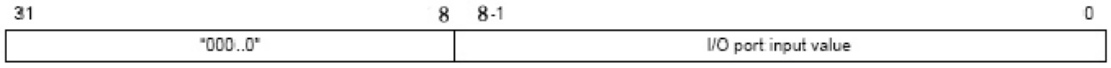
3.3.3 通用输入/输出口寄存器(GPIO)

GPIO 寄存器列表：

表 3-9　GPIO 寄存器

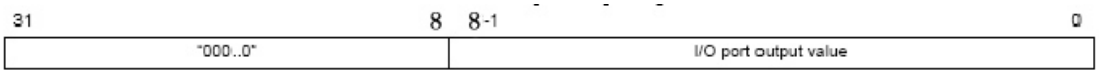
APB 地址	**寄存器**
0x80000b00	GPIO数据输入寄存器
0x80000b04	GPIO数据输出寄存器
0x80000b08	GPIO方向寄存器
0x80000b0C	GPIO中断屏蔽寄存器
0x80000b10	GPIO中断极性寄存器
0x80000b14	GPIO中断边沿寄存器

3.3.3.1 GPIO 数据寄存器-输入



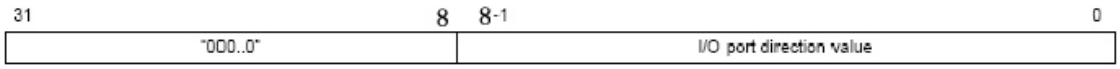
- 7-0：输入GIPO数据值

3.3.3.2 GPIO 数据寄存器输出



- 7-0：输出GPIO数据值

3.3.3.3 GPIO 方向寄存器



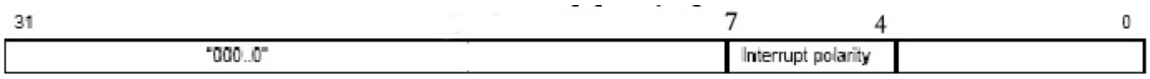
- 7-0：GPIO方向寄存器值，（0=输入，1=输出）

GPIO 口也复用成外部中断输入口。GPIO4~7 对应 4 个外部中断。

3.3.3.4 GPIO 中断屏蔽寄存器

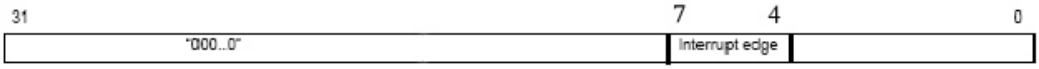
- 7-4：中断屏蔽，（0=屏蔽，1=中断允许）

3.3.3.5 GPIO 中断极性寄存器



- 7-4：中断极性，（0=低点平/下降沿，1=高点平/上升沿）

3.3.3.6 GPIO 中断方式寄存器



- 7-4：中断方式，（0=电平，1=边沿）

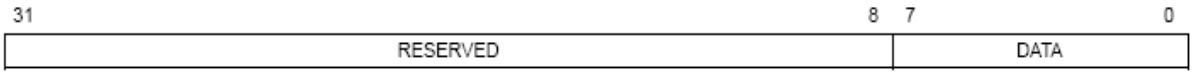
3.3.4 串口寄存器

表 3-10 UART寄存器

地址	寄存器
0x80000100	UART1 数据寄存器
0x80000104	UART1 状态寄存器
0x80000108	UART1 控制寄存器

0x8000010C	UART1 分频寄存器
0x80000900	UART2 数据寄存器
0x80000904	UART2 状态寄存器
0x80000908	UART2 控制寄存器
0x8000090C	UART2 分频寄存器

3.3.4.1 UART 数据寄存器



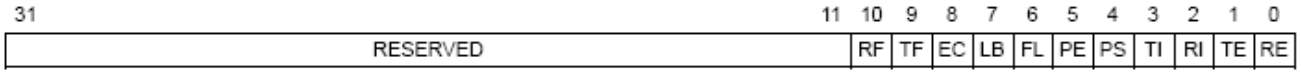
- 7： 0 接收器保持寄存器（读访问）
- 7： 0 发送器保持寄存器（写访问）

3.3.4.2 UART 状态寄存器

- 31： 26 接收器先进先出计数(RCNT) -显示在接收器先进先出中的数据帧数目
- 25： 20 发送器先进先出 计数(TCNT) -显示在发送机先进先出中的数据帧的数目.
- 10 接收器先进先出 满(RF) - .声明接收器先进先出是满的
- 9 发送器先进先出满(TF) -声明发送器先进先出是满的
- 8 接收器先进先出 半满(RH) -声明持有至少半个先进先出数据
- 7 发送器 先进先出半满l (TH) -声明先进先出少于半满
- 6 帧错误(FE) -声明帧错误被检测到
- 5 奇偶错误 (PE) -声明奇偶错误被检测到
- 4 溢出(OV) -声明一个或更多的字符由于溢出丢失
- 3 突发接收 (BR) - 声明一个BREAK被检测到。
- 2 发送器 先进先出 空(TE) -声明发送器先进先出是空的
- 1 发送器移位寄存器空 (TS) -声明发送器移动寄存器是空的

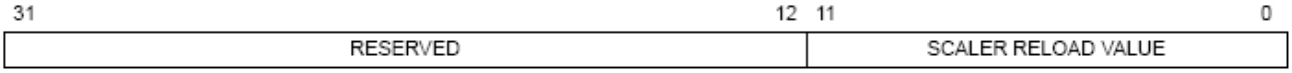
- 0　　数据准备位 (DR) -声明新的数据在接收器保持寄存器是有效的

3.3.4.3 UART 控制寄存器



- 10 接收器 先进先出 中断使能 (RF) -当被设定，接收器先进先出层中断使能
- 9 发送器 先进先出 中断使能 (TF) -当被设定，发送器先进先出层中断使能
- 8 外部时钟 (EC) –当前无意义
- 7 回环 (LB) – 如果设定，回环模式被使能
- 6 流控制(FL) -如果设定，使用CTS/RTS流控制使能
- 5 奇偶使能 (PE) -如果设定，奇偶生成和校验被使能
- 4 奇偶选择(PS) -选择奇偶 (0 = 偶校验, 1 =奇校验)
- 3 发送器中断使能 (TI) -如果设定，当发送完一个帧，中断生成
- 2 接收器中断使能 (RI) -如果设定，当接收到一个帧，中断生成
- 1 发送器使能 (TE) -如果设定，使能发送
- 0 接收器使能 (RE) -如果设定，使能接收

3.3.4.4 UART 分频寄存器



- 11： 0 分频重载值

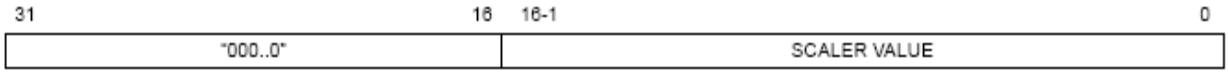
3.3.5 通用定时器寄存器

表 3-11　通用定时器寄存器

地址	寄存器
0x80000300	预置数

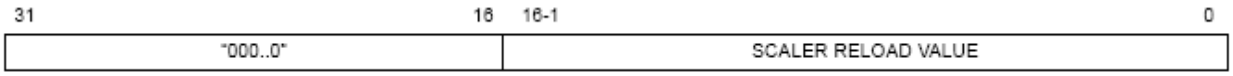
0x80000304	预置数重载值
0x80000308	配置寄存器
0x8000030C	未使用
0x80000310	定时器 1 计数寄存器
0x80000314	定时器 1 重载值寄存器
0x80000318	定时器 1 控制寄存器
0x8000031C	未使用
0x80000320	定时器 2 计数寄存器
0x80000324	定时器 2 重载值寄存器
0x80000328	定时器 2 控制寄存器

3.3.5.1 预置数寄存器



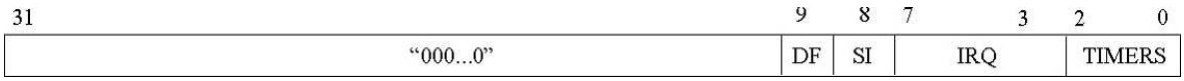
- 16-1：预置数值

3.3.5.2 预置数重载值寄存器



- 16-1：预置数重载值

3.3.5.3 定时器配置寄存器



- [31：10]： 保留
- 9：定时器冻结无效位DF。无意义。

- 8：单独中断位。如果定时器单元为每个定时器生成单独的中断，读‘1’；否则，读‘0’。此位为只读位。
- [7：3]：APB中断。如果配置使用公用中断，所有定时器将会驱动APB中断nr.IRQ，否则定时器n将会驱动中断IRQ-n（不得的少于MAXIRQ）。此位为只读位。
- [2：0]：定时器的数目。此位为只读位。

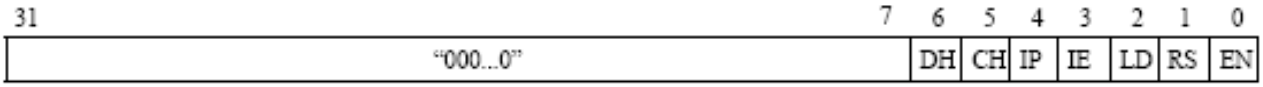
3.3.5.4 定时器计数寄存器

- [31：0]：定时器计数器值。每当预置数值递减到0时， 计数值递减1。

3.3.5.5 定时器计数重载寄存器

- [31：0]：定时器计数重载值。当‘1’写入定时器控制寄存器载入位或当控制寄存器中的RS位设置且定时器下溢出时 ，这个值会下载到定时器计数器值寄存器中。

3.3.5.6 定时器控制寄存器



- [31：7]：保留位。通常读作“000……0”。
- 6：调试停止位。调试模式用来冻结寄存器。只读位。
- 5：链接位。前面所述定时器的链接。如果为定时器 n 设定，当定时器 (n-1) 下溢出时，定时器 n 减1。

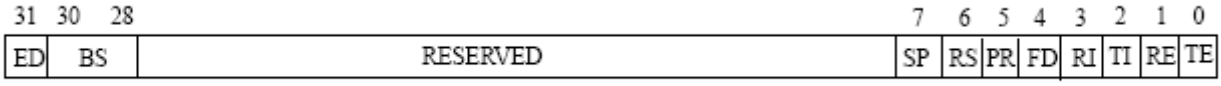
- 4：中断挂起。当中断产生时置位。 这一位会保持为‘1’，直到写 '0' 给这一位。
- 3：中断使能位。如果设定，定时器在下溢出时，发出中断信号。
- 2：加载位。从定时器计数值重载寄存器加载值到定时器计数寄存器。
- 1：复位位。如果设定，当寄存器下溢出时，定时器计数器值寄存器会重载计数值重载寄存器中的值。
- 0：使能位。使能定时器。

3.3.6 以太网控制器寄存器

表 3-12 以太网控制器寄存器

地址	寄存器
0x80000f00	控制寄存器
0x80000f04	状态/中断源寄存器
0x80000f08	MAC 地址 MSB
0x80000f0C	MAC 地址 LSB
0x80000f10	MDIO 控制/状态寄存器
0x80000f14	发送描述符指针
0x80000f18	接收器描述符指针
0x80000f1C	EDCL IP

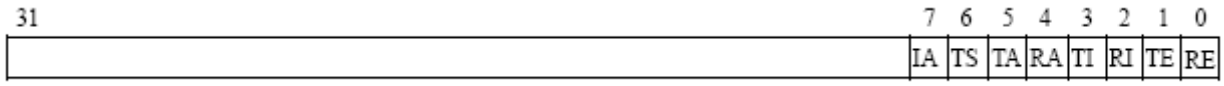
3.3.6.1 以太网控制器控制器寄存器



- [31] EDCL 可用 (ED) -如果EDCL 可用，设定‘1’
- [30：28] EDCL 缓冲器大小 (BS) – 显示用于EDCL缓冲器的存储器大小。 0 = 1 kB, 1 = 2 kB, ..., 6 = 64 kB
- [7] 速度 (SP) -设置现在的速度模式。 0 = 10 M位, 1 = 100 M位. 只在 RMII 模式使用(rmii = 1)。默认值自动在复位后从PHY读取

- [6] 复位 (RS) – 该位写1后，将复位这个核。复位值为0
- [5] 混杂模式(PM) -模式，如果设定，意味着以太网控制器将不管目的地址，接收到所有的包。没有复位值
- [4] 全双工 (FD) -如果设定，以太网控制器工作在一个全双工模式，否则它将运行半双工。没有复位
- [3] 接收中断(RI) -使能接收器中断。当这位为1时，每次接收到一个包，将会产生一个中断。不管这个包正确接收还是因为错误而终止接收，这个中断都会产生。没有复位值
- [2] 发送中断(TI) -使能发送器中断。当这位为1时，每次一个包发出，将会产生一个中断。不管这个包正确传输还是因为错误而终止发送，这个中断都会产生。没有复位值
- [1] 接收使能(RE) -在每次新的描述符被使能后应该置‘1’。只要这位是1，以太网控制器将读取新的描述符，直到它遭遇一个未被使能的描述符，此时它将停止直到RE被重新设定。这一位应该在新的描述符被使能后才写入1。复位值为‘0’
- [0] 发送使能 (TE) -应该在每次新的描述器被使能后写入1。只要这位是1，以太网控制器将读取新的描述符，直到它遭遇一个未被使能的描述符，此时它将停止直到TE被重新设定。这一位应该在新的描述符被使能后才写入1。复位值为‘0’

3.3.6.2 以太网控制器状态寄存器

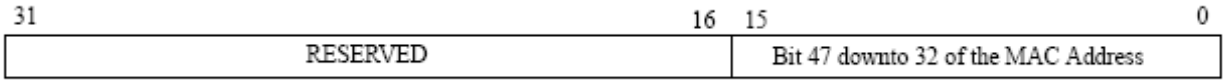


- [7] 无效地址 (IA) –MAC接收到一个不接受地址的包。当写‘1’时清除。复位值为‘0’
- [6] 过小 (TS) -接收到一个包，这个包的大小比允许的最小值小。当写入‘1’时被擦除。复位值为‘0’
- [5] 发送器 AHB 错误 (TA) -在发送器DMA器中发生AHB错误。当写入‘1’时被清除。没有复位值
- [4] 接收器 AHB 错误 (RA) -在接收器DMA器中发生AHB错误。当写入‘1’时被清

除。没有复位值

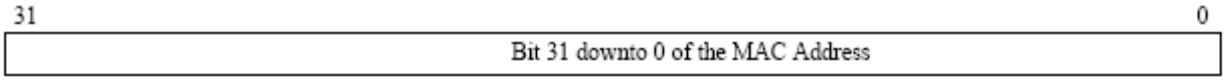
- [3] 发送器中断 (TI) -一个包被正确发送。当写入‘1’被清除。没有复位值
- [2] 接收器中断 (RI) -一个包被正确接收。当写入‘1’被清除。没有复位值
- [1] 发送器错误 (TE) -一个包将被发送，但是发送以错误结束。当写入‘1’清除。没有复位值
- [0] 接收器错误 (RE) -一个包将被接收，但是接收以错误结束。当写入‘1’清除。没有复位值

3.3.6.3 MAC 地址 MSB



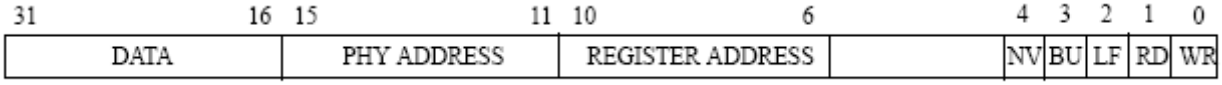
- [15：0] MAC地址的高2个字节

3.3.6.4 MAC 地址 LSB



- [31：0] MAC地址的高4个字节

3.3.6.5 MDIO 控制/状态寄存器

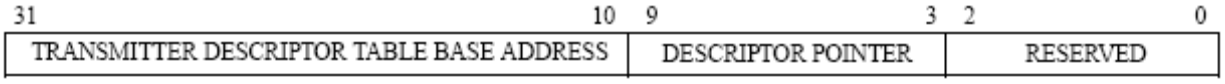


- [0] 写（WR） 。在操作接口开始写操作。从资料领域取数。复位值为‘0’。
- [1] 读(RD)。在操作接口开始读操作。数据存储在数据领域。复位值为‘0’。
- [2] 连接失败(LF)。当操作完成（BUSY = 0），如果一个功能性管理连结没有检测到这

位置位。没有复位值。

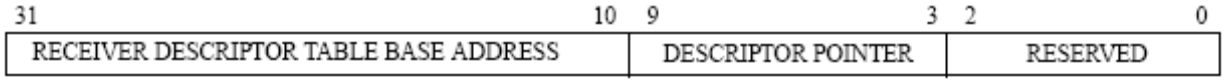
- [3] 忙(BU)。当操作启动，这位被设定为‘1’。一旦操作完成，管理连接空闲，这位清零。复位值为‘0’。
- [4] 无效位(NV)。当操作完成(BUSY = 0)，这位指示是否接收到有效资料，依据是数据域包含正确的资料。无复位值。
- [10：6] 寄存器地址。这个领域包含要访问的寄存器地址。
- [15：11] PHY 地址。这个领域包含要访问的PHY地址。没有复位值。
- [31：16] 数据。包含读操作时的数据或者需要发送的数据。没有复位值。

3.3.6.6 以太网控制器发送描述符表基地址寄存器



- [31：10]发送描述附表的基地址。没复位值。
- [9：3] 有效描述符偏移地址。被以太网MAC自动增加。
- [2：0] 保留，读为0。

3.3.6.7 以太网控制器接收描述符表基地址寄存器



- [31：10]接收描述附表的基地址。没复位值。
- [9：3] 有效描述符偏移地址。被以太网MAC自动增加。
- [2：0] 保留，读为0。

3.3.7 CAN 总线控制器寄存器

3.3.7.1 BasicCAN 模式

3.3.7.1.1 BasicCAN 寄存器映射

在 BasicCAN 模式下，寄存器的基地址为 0xfffc0000，偏移地址如表 3-20 所示。

表 3-20 BasicCAN 地址分配

地址	工作模式		复位模式	
	读	写	读	写
0	控制	控制	控制	控制
1	(0xFF)	命令	(0xFF)	命令
2	状态	-	状态	-
3	中断	-	中断	-
4	(0xFF)	-	验收过滤代码	验收过滤代码
5	(0xFF)	-	验收过滤屏蔽	验收过滤屏蔽
6	(0xFF)	-	总线定时 0	总线定时 0
7	(0xFF)	-	总线定时 1	总线定时 1
8	(0x00)	-	(0x00)	-
9	(0x00)	-	(0x00)	-
10	发送id1	发送id1	(0xFF)	-
11	发送id2, rtr, dlc	发送id2, rtr, dlc	(0xFF)	-
12	发送数据字节 1	发送数据字节 1	(0xFF)	-
13	发送数据字节 2	发送数据字节 2	(0xFF)	-
14	发送数据字节 3	发送数据字节 3	(0xFF)	-
15	发送数据字节 4	发送数据字节 4	(0xFF)	-
16	发送数据字节 5	发送数据字节 5	(0xFF)	-
17	发送数据字节 6	发送数据字节 6	(0xFF)	-
18	发送数据字节 7	发送数据字节 7	(0xFF)	-

19	发送数据字节 8	发送数据字节 8	(0xFF)	-
20	接收id1	-	接收id1	-
21	接收id2, rtr, dlc	-	接收id2, rtr, dlc	-
22	接收数据字节 1	-	接收数据字节 1	-
23	接收数据字节 2	-	接收数据字节 2	-
24	接收数据字节 3	-	接收数据字节 3	-
25	接收数据字节 4	-	接收数据字节 4	-
26	接收数据字节 5	-	接收数据字节 5	-
27	接收数据字节 6	-	接收数据字节 6	-
28	接收数据字节 7	-	接收数据字节 7	-
29	接收数据字节 8	-	接收数据字节 8	-
30	(0x00)	-	(0x00)	-
31	时钟分频	时钟分频	时钟分频	时钟分频

3.3.7.1.2 控制寄存器

控制寄存器包含中断允许位和复位请求位。

7	6	5	4	3	2	1	0
-	-	-	OIE	EIE	TIE	RIE	RR

- 7: 5 保留
- 4 超载中断允许 - 1-允许，0-禁止
- 3 错误中断允许 - 1-允许，0-禁止
- 2 发送中断允许 - 1-允许，0-禁止
- 1 接收中断允许 - 1-允许，0-禁止
- 0 复位请求 - 往该位写 1 将会中止任何进行中的传送并进入复位模式。往该位写 0 将会返回工作模式。

3.3.7.1.3 命令寄存器

往寄存器的相应位写 1 将引起被支持的动作。

7	6	5	4	3	2	1	0
-	-	-	-	CDO	RRB	AT	TR

- 7: 5 保留
- 4 未使用(在 SJA1000 中用于进入睡眠模式)
- 3 清除数据超载 - 清除数据超载状态位
- 2 释放接收缓冲 - 为新的接收释放当前的接收缓冲
- 1 中止发送 - 中止未开始的发送
- 0 发送请求 - 开始发送在发送缓冲里的信息

往 CMR.0 写 1 开始发送。只有在传输未开始时往 CMR.1 写 1 才能中止。如果发送已经开始，置位 CMR.1 将不会中止发送动作，但如果发生错误则不会重发。

在读接收缓冲以后应该发出释放接收缓冲命令用来释放存储器。如果有另外一个信息在 FIFO 里等待，一个新的接收中断将会产生（如果使能），并且接收缓冲状态位将被重新置位。

往 CMR.3 写 1 将清除数据超载状态位。

3.3.7.1.4 状态寄存器

状态寄存器反映模块的当前状态并且是只读的。

7	6	5	4	3	2	1	0
BS	ES	TS	RS	TCS	TBS	DOS	RBS

- 7 总线状态 - 当总线关闭并退出总线活动时为 1
- 6 错误状态 - 至少一个错误计数器到达或超过 CPU 警告限制（96）
- 5 发送状态 - 当正在发送信息时为 1
- 4 接收状态 - 当正在接收信息时为 1
- 3 发送完成 - 1 表明最后一个信息被成功发送
- 2 发送缓冲状态 - 1 表明 CPU 可以往发送缓冲写数据
- 1 数据超载状态 - 如果因为 FIFO 没有空间导致信息丢失则为 1

- 0　　接收缓冲状态 - 如果接收 FIFO 里面有信息则为 1

当释放接收缓冲命令发出时，接收缓冲状态将被清除，当 FIFO 里有更多可得到的信息，它将被置位。

数据超载状态表明接收的信息不能放在 FIFO 里，因为没有足够的剩余空间。注意：此位与 SJA1000 的行为不同，它在 FIFO 被读出后首先置位。

当发送缓冲状态是高电平时，发送缓冲可以被 CPU 写入。在发送期间缓冲被锁定并且此位为 0。

当发送请求发出以后发送完成位被置为 0，并且直到信息被成功发送以后才被置为 1。

3.3.7.1.5 中断寄存器

中断寄存器通知 CPU 是什么引起了中断。只有在控制寄存器里相应的中断允许位置 1 时中断位才置 1。

7	6	5	4	3	2	1	0
-	-	-	-	DOI	EI	TI	RI

- 7: 5　　保留
- 4　　未使用(在 SJA1000 中用于唤醒中断)
- 3　　数据超载中断 - 当 SR.1 从 0 到 1 变化则置位
- 2　　错误中断 - 当错误状态或总线状态改变时则置位
- 1　　发送中断 - 当发送缓冲被释放时则置位（状态位 0->1）
- 0　　接收中断 - 当接收 FIFO 里还有信息时则置位

此寄存器在被读后复位，除了 IR.0。注意这与 SJA1000 的行为不同，在 BasicCAN 模式下，SJA1000 的所有位都在被读后复位。当发出释放接收缓冲命令后，接收中断位将被复位（类似 PeliCAN 模式）。

还要注意 IR.5 到 IR.7 的位读出的值都是 1，但 IR.4 是 0。

3.3.7.1.6 发送缓冲

发送缓冲存储来自 CPU 的将要通过本模块发送的数据。在 BasicCAN 模式下只有标准帧格式信息可以被发送和接收，扩展帧格式信息将被忽略。

表 3-21 发送缓冲结构

地址	名称	位							
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
10	ID字节 1	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
11	ID字节 2	ID.2	ID.1	ID.0	RTR	DLC.3	DLC.2	DLC.1	DLC.0
12	发送数据 1	发送字节 1							
13	发送数据 2	发送字节 2							
14	发送数据 3	发送字节 3							
15	发送数据 4	发送字节 4							
16	发送数据 5	发送字节 5							
17	发送数据 6	发送字节 6							
18	发送数据 7	发送字节 7							
19	发送数据 8	发送字节 8							

如果 RTR 位被置位，将没有数据字节被发送，但 DLC 仍然是帧的一部分并且必须根据请求帧来确定。注意指定大于 8 字节的 DLC 是允许的，但为了保持兼容性不应该这样做。如果 DLC>8，只有 8 字节被发送。

3.3.7.1.7 接收缓冲

位于地址 20 至 29 的接收缓冲是 64 字节接收 FIFO 的可见部分。它的结构与发送缓冲相同。

3.3.7.1.8 验收过滤

应用验收过滤代码和验收过滤屏蔽寄存器，信息可以根据它们的标识符（ID）被过滤。11 位的标识符的高 8 位与验收过滤代码寄存器中相应的验收过滤屏蔽寄存器中设为 0 的位比较，如果匹配则储存进 FIFO。

3.3.7.2 PeliCAN 模式

3.3.8.2.1 PeliCAN 寄存器映射

基地址为 0xfffc0000，偏移地址如表 3-22 所示。

表 3-22 PeliCAN 地址分配

地址	工作模式				复位	
	读		写		读	写
0	模式		模式		模式	模式
1	(0x00)		命令		(0x00)	命令
2	状态		-		状态	-
3	中断		-		中断	-
4	中断允许		中断允许		中断允许	中断允许
5	保留 (0x00)		-		保留 (0x00)	-
6	总线定时 0		-		总线定时 0	总线定时 0
7	总线定时 1		-		总线定时 1	总线定时 1
8	(0x00)		-		(0x00)	-
9	(0x00)		-		(0x00)	-
10	保留 (0x00)		-		保留 (0x00)	-
11	仲裁丢失捕捉		-		仲裁丢失捕捉	-
12	错误代码捕捉		-		错误代码捕捉	-
13	错误警告限制		-		错误警告限制	错误警告限制
14	接收错误计数器		-		接收错误计数器	接收错误计数器
15	发送错误计数器		-		发送错误计数器	发送错误计数器
16	接收帧信息SFF	接收帧信息EFF	发送帧信息SFF	发送帧信息EFF	验收代码 0	验收代码 0
17	接收ID 1	接收ID 1	发送ID 1	发送ID 1	验收代码 1	验收代码 1

18	接收ID 2	接收ID 2	发送ID 2	发送ID 2	验收代码 2	验收代码 2
19	接收数据 1	接收ID 3	发送数据 1	TX ID 3	验收代码 3	验收代码 3
20	接收数据 2	接收ID 4	发送数据 2	TX ID 4	验收屏蔽 0	验收屏蔽 0
21	接收数据 3	接收数据 1	发送数据 3	发送数据 1	验收屏蔽 1	验收屏蔽 1
22	接收数据 4	接收数据 2	发送数据 4	发送数据 2	验收屏蔽 2	验收屏蔽 2
23	接收数据 5	接收数据 3	发送数据 5	发送数据 3	验收屏蔽 3	验收屏蔽 3
24	接收数据 6	接收数据 4	发送数据 6	发送数据 4	保留 (0x00)	-
25	接收数据 7	接收数据 5	发送数据 7	发送数据 5	保留 (0x00)	-
26	接收数据 8	接收数据 6	发送数据 8	发送数据 6	保留 (0x00)	-
27	FIFO	接收数据 7	-	发送数据 7	保留 (0x00)	-
28	FIFO	接收数据 8	-	发送数据 8	保留 (0x00)	-
29	接收信息计数器		-		接收信息计数器	-
30	(0x00)		-		(0x00)	-
31	时钟分频器		时钟分频器		时钟分频器	时钟分频器

根据将要发送/接收的是标准帧格式（SFF）还是扩展帧格式（EFF），发送和接收缓冲有不同的结构。见下面具体的部分。

3.3.8.2.2 模式寄存器

7	6	5	4	3	2	1	0
-	-	-	-	AFM	STM	LOM	RM

- 7: 5 保留
- 4 未使用(在 SJA1000 中用于睡眠模式)
- 3 验收过滤模式 - 1-单过滤模式, 0-双过滤模式
- 2 自测试模式 - 如果置 1 则控制器处于自测试模式
- 1 只听模式 - 如果置 1 则控制器处于只听模式
- 0 复位模式 - 往该位写 1 将会中止任何进行中的传送并进入复位模式。往该位写 0 将会返回工作模式。

只有事先进入了复位模式才能写 MOD.1-3。

在只听模式下核不会发送任何应答。注意不像 SJA1000，S698P4-II 的 CAN 总线控制器不会变成被动错误状态并且活动错误帧仍然被发送。

当在自检测模式下，如果发出自接收请求命令，核可以在没有应答的情况下完成一个成功的发送。注意核仍然必须连接在实际的总线上，它没有内部的回环。

3.3.8.2.3 命令寄存器

往寄存器的相应位写 1 将引起被支持的动作。

7	6	5	4	3	2	1	0
-	-	-	SRR	CDO	RRB	AT	TR

- 7: 5 保留
- 4 自接收请求 - 发送的同时接收信息
- 3 清除数据超载 - 清除数据超载状态位
- 2 释放接收缓冲 - 为新的接收释放当前的接收缓冲
- 1 中止发送 - 中止未开始的发送
- 0 发送请求 - 开始发送在发送缓冲里的信息

往 CMR.0 写 1 开始发送。只有在传输未开始时往 CMR.1 写 1 才能取消。同时置位 CMR.0 和 CMR.1 将会引起所谓的单次发送，如果第一次发送不成功将不会重发。

在读接收缓冲以后应该发出释放接收缓冲命令用来释放存储器。如果有另外一个信息在 FIFO 里等待，一个新的接收中断将会产生（如果使能），并且接收缓冲状态位将被重新置位。

自接收请求位和自检测模式使得在没有其他核在总线时，此核的自检测变得可能。信息被同时发送和接收，并且发送和接收中断都会产生。

3.3.8.2.4 状态寄存器

状态寄存器反映模块的当前状态并且是只读的。

7	6	5	4	3	2	1	0
BS	ES	TS	RS	TCS	TBS	DOS	RBS

- 7　总线状态 - 当总线关闭并退出总线活动时为 1
- 6　错误状态 - 至少一个错误计数器到达或超过 CPU 警告限制
- 5　发送状态 - 当正在发送信息时为 1
- 4　接收状态 - 当正在接收信息时为 1
- 3　发送完成 - 1 表明最后一个信息被成功发送
- 2　发送缓冲状态 - 1 表明 CPU 可以往发送缓冲写数据
- 1　数据超载状态 - 如果因为 FIFO 没有空间导致信息丢失则为 1
- 0　接收缓冲状态 - 如果接收 FIFO 里面有信息则为 1

当 FIFO 里没有信息时接收缓冲状态将被清除。数据超载状态表明接收的信息不能放在 FIFO 里，因为没有足够的剩余空间。注意：此位与 SJA1000 的行为不同，它在 FIFO 被读出后首先置位。

当发送缓冲状态是高电平时，发送缓冲可以被 CPU 写入。在发送期间缓冲被锁定并且此位为 0。

当发送请求或自接收请求发出以后发送完成位被置为 0，并且直到信息被成功发送以后才被置为 1。

3.3.8.2.5 中断寄存器

中断寄存器通知 CPU 是什么引起了中断。只有在中断允许寄存器里相应的中断允许位

置 1 时中断位才置 1。

7	6	5	4	3	2	1	0
BEI	ALI	EPI	-	DOI	EI	TI	RI

- 7 总线错误中断 - 如果侦测到总线上有错误则置位
- 6 仲裁丢失中断 - 当丢失仲裁则置位
- 5 错误被动中断 - 当从错误活动状态到错误被动状态转变时则置位
- 4 未使用(在 SJA1000 中用于唤醒中断)
- 3 数据超载中断 - 当数据超载状态位置位时则置位
- 2 错误警告中断 - 当错误状态或总线状态改变时则置位
- 1 发送中断 - 当发送缓冲被释放时则置位
- 0 接收中断 - 当 FIFO 不为空时则置位

此寄存器在读后复位，除了 IR.0 例外，它在 FIFO 清空以后复位。

3.3.8.2.6 中断允许寄存器

在中断允许寄存器里可以允许/禁止独立的中断源。如果被允许，则中断寄存器里的相应位可以被置 1，同时将产生一个中断。

7	6	5	4	3	2	1	0
BEIE	ALIE	EPIE	-	DOIE	EIE	TIE	RIE

- 7 总线错误中断 - 1-允许, 0-禁止
- 6 仲裁丢失中断 - 1-允许, 0-禁止
- 5 错误被动中断 - 1-允许, 0-禁止
- 4 未使用(在 SJA1000 中用于唤醒中断)
- 3 数据超载中断 - 1-允许, 0-禁止
- 2 错误警告中断 - 1-允许, 0-禁止
- 1 发送中断 - 1-允许, 0-禁止
- 0 接收中断 - 1-允许, 0-禁止

3.3.8.2.7 仲裁丢失捕捉寄存器

7	6	5	4	3	2	1	0
-	-	-	BITNO				

- 7: 5 保留
- 4: 0 位编号 - 仲裁丢失的位的位置

当核丢失仲裁时，位流处理器的当前位位置被捕捉到仲裁丢失捕捉寄存器中。此寄存器直到被读出后才会再次改变它的内容。

3.3.8.2.8 错误代码捕捉寄存器

7	6	5	4	3	2	1	0
ERRC		DIR	SEG				

- 7: 6 错误代码 - 错误代码编号
- 5 方向 - 1-接收, 0-发送
- 4: 0 段 - 在帧的何处发生错误

当总线错误产生时，错误代码捕捉寄存器会根据是什么类型的错误，是正在发送还是接收和在帧的什么位置发生了错误而被设置。与仲裁丢失捕捉寄存器一样，错误代码捕捉寄存器直到被读出以后才会改变它的值。

下表显示错误代码捕捉寄存器的 7-6 位的解释。

表 3-23 错误代码说明

ECC.7-6	**描述**
0	位错误
1	格式错误
2	填充错误
3	其他

表 3-24 ECC.4-0 说明

ECC.4-0	**描述**

0x03	帧开始
0x02	ID.28 - ID.21
0x06	ID.20 - ID.18
0x04	SRTR位
0x05	IDE位
0x07	ID.17 - ID.13
0x0F	ID.12 - ID.5
0x0E	ID.4 – ID.0
0x0C	RTR位
0x0D	保留位 1
0x09	保留位 0
0x0B	数据长度代码
0x0A	数据区
0x08	CRC序列
0x18	CRC定界符
0x19	应答间隙
0x1B	应答定界符
0x1A	帧结束
0x12	帧间歇
0x11	活动错误标志
0x16	被动错误标志
0x13	显性位误差
0x17	错误定界符
0x1C	超载标志

3.3.8.2.9 错误警告限制寄存器

该寄存器允许设置 CPU 错误警告的限制。默认值是 96。注意该寄存器只在复位模式下可写。

3.3.8.2.10 接收错误计数器寄存器

该寄存器显示接收错误计数器的值。它在复位模式下可写。总线关闭事件会把它复位为 0。

3.3.8.2.11 发送错误计数器寄存器

该寄存器显示接收错误计数器的值。它在复位模式下可写。总线关闭事件会把它复位为 0。

3.3.8.2.12 发送缓冲

发送缓冲被映射为地址 16 至 28 并且是只写的。发送缓冲的结构取决于将要发送的是标准帧（SFF）还是扩展帧（EFF），如下所示：

表 3－25

#	写 (SFF)	写(EFF)
16	发送帧信息	发送帧信息
17	发送ID 1	发送ID 1
18	发送ID 2	发送ID 2
19	发送数据 1	发送ID 3
20	发送数据 2	发送ID 4
21	发送数据 3	发送数据 1
22	发送数据 4	发送数据 2
23	发送数据 5	发送数据 3
24	发送数据 6	发送数据 4
25	发送数据 7	发送数据 5
26	发送数据 8	发送数据 6
27	-	发送数据 7
28	-	发送数据 8

发送帧信息（此位场在 SFF 和 EFF 帧中具有相同的结构）

7	6	5	4	3	2	1	0
FF	RTR	-	-	DLC.3	DLC.2	DLC.1	DLC.0

- 7： FF选择帧格式，例如被解释为扩展帧或标准帧。1 = EFF，0 = SFF。
- 6： 对于远程发送请求帧RTR应被置为1。
- [5：4]： 不考虑。
- 位3：0 – DLC指定数据长度代码并且应该是0到8之间的数值。如果大于8，则8个字节将被发送。

发送标识符 1（此位场在 SFF 帧和 EFF 帧中相同）

7	6	5	4	3	2	1	0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

- [7：0] 标识符的高8位。

发送标识符 2，SFF 帧

7	6	5	4	3	2	1	0
ID.20	ID.19	ID.18	-	-	-	-	-

- [7：5] SFF标识符的低3位。
- [4：0] 不考虑。

发送标识符 2，EFF 帧

7	6	5	4	3	2	1	0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

- [7：0] 29位EFF标识符的第20到第13位。

发送标识符 3，EFF 帧

7	6	5	4	3	2	1	0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

- [7：0] 29位EFF标识符的第12到第5位。

发送标识符 4，EFF 帧

7	6	5	4	3	2	1	0
ID.4	ID.3	ID.2	ID.1	ID.0	-	-	-

- [7：3] 29位EFF标识符的第4到第0位。
- [2：0] 不考虑。

数据位场

对于 SFF 帧，数据位场位于地址 19 到 26，对于 EFF 帧，位于 21 到 28。数据从位于最低地址的 MSB（最高位字节）开始发送。

3.3.8.2.13 接收缓冲

表 3-26

#	读(SFF)	读(EFF)
16	接收帧信息	接收帧信息
17	接收ID 1	接收ID 1
18	接收ID 2	接收ID 2
19	接收数据 1	接收ID 3
20	接收数据 2	接收ID 4
21	接收数据 3	接收数据 1
22	接收数据 4	接收数据 2
23	接收数据 5	接收数据 3
24	接收数据 6	接收数据 4
25	接收数据 7	接收数据 5
26	接收数据 8	接收数据 6
27	FIFO里的下一个信息的接收帧信息	接收数据 7
28	FIFO里的下一个信息的接收ID 1	接收数据 8

接收帧信息（此位场在 SFF 和 EFF 帧中具有相同的结构）

7	6	5	4	3	2	1	0
FF	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0

- [7] 已接收信息的帧格式。1 = EFF，0 = SFF。
- [6] RTR帧时为1。
- [5：4] 总为0。
- [3：0] DLC指定数据长度代码。

接收标识符 1（此位场在 SFF 帧和 EFF 帧中相同）

7	6	5	4	3	2	1	0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

- [7：0] 标识符的高8位。

接收标识符 2，SFF 帧

7	6	5	4	3	2	1	0
ID.20	ID.19	ID.18	RTR	0	0	0	0

- [7：5] SFF标识符的低3位。
- [4] RTR帧时为1。
- [3：0] 总为0。

接收标识符 2，EFF 帧

7	6	5	4	3	2	1	0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

- [7：0] 29位EFF标识符的第20到第13位。

接收标识符 3，EFF 帧

7	6	5	4	3	2	1	0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

- [7：0] 29位EFF标识符的第12到第5位。

接收标识符 4，EFF 帧

7	6	5	4	3	2	1	0
ID.4	ID.3	ID.2	ID.1	ID.0	RTR	0	0

- [7：3] 29位EFF标识符的第4到第0位。
- [2] RTR帧时为1。
- [1：0] 不考虑。

数据场

对于接收到的 SFF 帧，数据场位于地址 19 到 26，对于 EFF 帧则位于 21 到 28。

3.3.8.2.14 验收过滤

验收过滤器可以用来过滤掉不符合特定要求的信息。如果一个信息被过滤掉，它将不被放进接收 FIFO 里面，CPU 也不必处理它。

有两种不同的过滤模式：单过滤和双过滤。模式寄存器的第 3 位控制使用哪种模式。在单过滤模式下只使用一个 4 个字节的过滤器。在双过滤模式下使用两个更小的过滤器，如果匹配其中任何一个，则信息被接收。每个过滤器由两部分组成：验收代码和验收屏蔽。代码寄存器用来指定匹配的格式而屏蔽寄存器则指定不考虑的位。总共 8 个寄存器被用作验收过滤器，如下表所示。注意它们只在复位模式下被读写。

表 3-27 验收过滤寄存器

地址	描述
16	验收代码 0 (ACR0)
17	验收代码 1 (ACR1)
18	验收代码 2 (ACR2)
19	验收代码 3 (ACR3)
20	验收屏蔽 0 (AMR0)
21	验收屏蔽 1 (AMR1)
22	验收屏蔽 2 (AMR2)
23	验收屏蔽 3 (AMR3)

单过滤模式，标准帧

当在单过滤模式下接收一个标准帧，寄存器 ACR0-3 将会以以下的方式与接收到的信息比较：

ACR0.7-0 和 ACR1.7-5 与 ID.28-18 比较。

ACR1.4 与 RTR 位比较。

ACR1.3-0 未使用。

ACR2 和 ACR3 与数据字节 1 和 2 比较。

AMR 寄存器里相应的位选择是否比较的结果没关系。屏蔽寄存器里一个置 1 的位表示不考虑。

单过滤模式，扩展帧

当在单过滤模式下接收一个扩展帧，寄存器 ACR0-3 将会以以下的方式与接收到的信息比较：

ACR0.7-0 和 ACR1.7-0 与 ID.28-13 比较。

ACR2.7-0 和 ACR3.7-3 与 ID.12-0 比较。

ACR3.2 与 RTR 位比较。

ACR3.1-0 未使用。

AMR 寄存器里相应的位选择是否比较的结果没关系。屏蔽寄存器里一个置 1 的位表示不考虑。

双过滤模式，标准帧

当在双过滤模式下接收一个标准帧，寄存器 ACR0-3 将会以以下的方式与接收到的信息比较：

过滤器 1

ACR0.7-0 和 ACR1.7-5 与 ID.28-18 比较。

ACR1.4 与 RTR 位比较。

ACR1.3-0 与数据字节 1 的高半字节比较。

ACR3.3-0 与数据字节 1 的低半字节比较。

过滤器 2

ACR2.7-0 和 ACR3.7-5 与 ID.28-18 比较。

ACR3.3 与 RTR 位比较。

AMR 寄存器里相应的位选择是否比较的结果没关系。屏蔽寄存器里一个置 1 的位表示不考虑。

双过滤模式，扩展帧

当在双过滤模式下接收一个扩展帧，寄存器 ACR0-3 将会以以下的方式与接收到的信息比较：

过滤器 1

ACR0.7-0 和 ACR1.7-0 与 ID.28-13 比较。

过滤器 2

ACR2.7-0 和 ACR3.7-0 与 ID.28-13 比较。

AMR 寄存器里相应的位选择是否比较的结果没关系。屏蔽寄存器里一个置 1 的位表示不考虑。

3.3.8.2.15 接收信息计数器

位于地址 29 的接收信息计数器寄存器保持当前储存在接收 FIFO 里的信息的数量。最高 3 位总为 0。

3.3.7.3 公共寄存器

在 BasicCAN 和 PeliCAN 模式下有 3 个公共寄存器，它们具有相同的地址和相同的功能。它们是时钟分频寄存器和总线定时寄存器 0 和 1。

3.3.8.3.1 时钟分频寄存器

此寄存器在 S698P4-II 的 CAN 总线控制器中的唯一真实功能是选择 PeliCAN 和 BasicCAN 模式。S698P4-II 的 CAN 总线控制器的输出信号 clkout 没有被连接，它的频率可以在此寄存器里控制。

7	6	5	4	3	2	1	0
CM	-	-	-	CO	CD		

- 7 CAN 模式 - 1 - PeliCAN, 0 - BasiCAN
- 6 未使用 (在 SJA1000 中是 CBP 位)

- 5　　未使用 (在 SJA1000 中是 RXINTEN 位)
- 4　　保留
- 3　　时钟关闭 - 禁止 clkout 输出
- 2: 0　　时钟分频系数 - 选择频率

3.3.8.3.2 总线定时 0 寄存器

7	6	5	4	3	2	1	0
SJW				BRP			

- 7: 6　　SJW - 同步跳转宽度
- 5: 0　　BRP - 波特率预设值

CAN 控制器核的系统时钟由以下计算：

$t_{scl} = 2 * t_{clk} * (BRP + 1)$

其中 t_{scl} 是系统时钟。

同步跳跃宽度定义了在一次重同步中一个位周期里有多少个时钟周期（t_{scl}）可以调整。

3.3.8.3.3 总线定时 1 寄存器

7	6	5	4	3	2	1	0
SAM	TSEG2			TSEG1			

- 7　　SAM - 1-采样 3 次, 0-采样 1 次
- 6: 4　　TSEG2 - 时间段 2
- 3: 0　　TSEG1 - 时间段 1

CAN 总线的位周期由 CAN 的系统时钟和时间段 1 和 2 决定，如下面的等式所示：

$t_{tseg1} = t_{scl} * (TSEG1 + 1)$

$t_{tseg2} = t_{scl} * (TSEG2 + 1)$

$t_{bit} = t_{tseg1} + t_{tseg2} + t_{scl}$

附加的 t_{scl} 项来自初始的同步段。采样在位周期的 TSEG1 和 TSEG2 之间完成。

4. 1553B 寄存器描述

4.1 寄存器地址分配表

表 4-1 寄存器地址分配

地址（HEX）	读/写	有效位宽	默认值(HEX)	寄存器描述
0x00	RD/WR	16	0000	中断屏蔽寄存器(IMR)
0x01	RD/WR	16	0000	配置寄存器 1(CFG1)
0x02	RD/WR	16	0000	配置寄存器 2(CFG2)
0x03	WR	16	0000	启动/复位寄存器(SRR)
	RD	16	0000	BC/RT/BM 命令堆栈指针寄存器(STACK_ADDR)
0x04	RD/WR	16	0000	BM 初始命令堆栈指针寄存器
0x05	RD	16	0000	时间标签寄存器 0(TTR0)
0x06	RD	16	0000	中断状态寄存器(INT_STA)
0x07	RD/WR	16	0000	配置寄存器 3 (CFG3)
0x08	RD/WR	16	0000	配置寄存器 4(CFG4)
0x09	RD/WR	16	0000	配置寄存器 5(CFG5)
0x0A	RD	16	0000	BM 数据堆栈指针寄存器（BM_STACK_ADDR）
0x0B	RD	16	0000	保留
0x0C	RD	16	0000	保留
0x0D	RD	16	0000	RT 上一命令字寄存器(LAST_CMD)
	WR	16	0000	BC 帧时间寄存器
0x0E	RD	16	0000	RT 状态字寄存器（RT_STA）
0x0F	RD	16	0000	RT BIT 字寄存器(RT_BIT_REG)

注: 寄存器基地址为:0x80008000, RAM 基地址为:0x8000c000。由于内部数据位宽为 32

位，因此在进行程序编程正确的地址访问是：基地址＋绝对偏移地址*4， 如仿问配置寄存器 1(CFG1)， 其地址应为 0x80008000+0x01*4，即为 0x80008004。

4.2 蔽寄存器（IMR）

地址：0x00

表 4-2 中断屏蔽寄存器（IMR）

位（BIT）	描述 (DESCRIPTION)
15	保留
14	保留
13	BC/RT 传输器超时（BC/RT TRANSMITTER TIMEOUT）
12	BC/RT 命令堆栈溢出/ BM 命令堆栈半满溢出
11	BM 命令堆栈溢出（BM COMMAND STACK ROLLOVER）
10	BM 数据堆栈溢出（BM DATA STACK ROLLOVER）
9	保留
8	BC 重发/BM 数据半满溢出
7	RT 地址奇偶校验错误（RT ADDRESS PARITY ERROR）
6	时间标签寄存器溢出（TIME TAG ROLLOVER）
5	RT 循环缓存溢出（RT CIRCULAR BUFFER ROLLOVER）
4	BC 消息/RT 子地址控制字消息结束（BC MSG/RT SUBADDRESS CONTROL WORD EOM）
3	BC 帧结束（BC END OF FRAME）
2	格式错误(FORMAT ERROR)
1	BC 状态置位/RT 方式代码（BC STATUS SET/RT MODE CODE）
0	消息结束（END OF MESSAGE）

中断屏蔽寄存器中位如置 1 表示打开对应中断状态寄存器的中断，如置 0 表示关闭对应的中断状态寄存器的中断。

- **BIT13**： 芯片内置看门狗，当使能（置 1）该位则传输编码时间超过 668us 时产生中断。

- **BIT12**： 芯片命令堆栈大小为 256 字，当使能（置 1）该位则在 BC/RT 类型下命令堆栈指针超出 256 则产生中断；当在 BM 类型下则在命令堆栈半满时产生中断。
- **BIT11**：当使能（置 1）该位则 BM 命令堆栈溢出时产生中断。
- **BIT10**：当使能（置 1）该位则 BM 数据堆栈溢出时产生中断。
- **BIT8**：当使能（置 1）该位则在 BC 类型下 BC 重发消息前将产生中断；如果在 BM 类型下数据堆栈半满溢出将产生中断。
- **BIT7**： 在 RT 模式下当使能（置 1）该位那么如果 RTAD4-RTAD0 与 RTADP 共六位进行奇偶的结果是 0 就产生中断。
- **BIT6**： 当使能（置 1）该位那么当时标寄存器计时到 65535 个单位（由最小计时精度确定）时产生中断。
- **BIT5**： 当使能（置 1）该位那么 RT 循环缓存溢出产生中断。
- **BIT4**： 当使能（置 1）该位那么当作 BC 总线控制器时 BC 控制字中的 BIT4 位为 1 则消息结束产生中断；当作 RT 时当 RT 的子地址控制器的 BIT14 或 BIT9 或 BIT4 中的任意一位为 1 则产生中断。
- **BIT3**： 当使能（置 1）该位那么 BC 帧发送结束产生中断。
- **BIT2**： 当使能（置 1）该位那么格式错误时产生中断。
- **BIT1**： 当使能（置 1）该位那么当作 BC 时，BC 收到的状态字中有置 1 的位，则产生中断；当作 RT 时子地址查找表中方式字对应位为 1 则产生中断。
- **BIT0**： 当使能（置 1）该位那么当 BC/RT 发送/接收消息结束产生中断。

4.3 存器 1(CFG1- BC)

地址：0x01

表 4-3 配置寄存器 1(BC-CFG1)

位（BIT）	描述 (DESCRIPTION)
15(MSB)	BC/RT/BM 模式设置（RT/BM*-BC*）,(logic 0)
14	BC/RT/BM 模式设置（BM/RT*- BC*）,(logic 0)
13	A*/B 区域设置（CURRENT AREA A*/B）
12	保留

位（BIT）	描述 (DESCRIPTION)
11	保留
10	保留
9	保留
8	帧自动重复发送使能（FRAME AUTO-REPEAT）
7	保留
6	保留
5	消息间隔时间使能（MESSAGE GAP TIMER ENABLED）
4	消息重发使能（RETRY ENABLED）
3	消息重发一次或二次选择（DOUBLE/SINGLE* RETRY）
2	BC 使能（BC ENABLED），*该位只读*
1	BC 帧信息忙指示（BC FRAME IN PROGRESS），*该位只读*
0 (LSB)	BC 消息忙指示（BC MESSAGE IN PROGRESS），*该位只读*

BC 配置寄存器 1 主要用作 1553B 总线控制器工作模式的选择，还有是否使能消息重发、帧重复发送功能，以及 1553B 总线控制器工作状态的指示等。

- **BIT15,BIT14**：此两位组合为 00 设置为 BC 模式，10 设置为 RT 模式,01 设置为 BM 模式，上电默认为 BC 模式。
- **BIT13**：该位为 0 则使用 RAM 的 A 区域，该位为 1 则使用 RAM 的 B 区域。
- **BIT8**：该位为 0 则 BC 发送完一帧数据就停止，若该位为 1 则帧重复发送直到启动/复位寄存器(SSR)中的 BIT0(RESET)、BIT5(STOP_ON_FRAME)，BIT6(STOP_ON_MESSAGE)中任意位为 1 才会停止。
- **BIT5**：该位为 0 则消息之间间隔时间固定为近似 8 到 11us,这位为 1 则消息之间的间隔时间通过 BC 命令堆栈的第三个字指定，其指定范围为最小约 8us 到最大约 65535us, 时间精度为 1us。当为 10Mbps 传输速度时则相应为以前的 0.2 倍。
- **BIT4**：该位为 0，BC 对所有的消息都不重发，该位为 1 且 BC 控制字的 BIT8 也为 1 那么该消息在返回状态字出错，响应时间超时则重发消息。

- **BIT3**： 在配置寄存器 1（CFG1）的 BIT4 为 1 的条件下，该位为 0 则该消息在返回状态字出错，响应时间超时则重发一次; 该位为 1 则该消息在返回状态字出错，响应时间超时则重发消息两次。
- **BIT2**： 该位为只读位，含义同 BIT1。
- **BIT1**： 该位为只读位，在帧的第一个消息启动后到帧的最后一个消息结束一直被设为 1，在帧自动重复发送模式下则一直保持为 1 直到帧重复发送结束。
- **BIT0**： 该位在 BC 总线控制器每个消息开始传输时置为 1，在消息结束传输时清为 0。

4.4 存器 1 (CFG1-RT)

地址：0x01

表 4-4 配置寄存器 1(CFG1-RT)

位（BIT）	描述 (DESCRIPTION)
15(MSB)	BC/RT/BM 模式设置（RT/BM*-BC*）,(logic 1)
14	BC/RT/BM 模式设置（BM/RT*- BC*）,(logic 0)
13	保留
12	保留
11	动态总线控制接收*（DYNAMIC BUS CONTROL ACCEPTANCE*）
10	忙*（BUSY*）
9	服务请求*（SERVICE REQUEST*）
8	子系统标志*（SUBSYSTEM FLAG*）
7	RT 标志*（RTFLAG*）
6	保留
5	保留
4	保留
3	保留

位（BIT）	描述 (DESCRIPTION)
2	保留
1	保留
0	BC 消息忙指示（BC MESSAGE IN PROGRESS），*该位只读*

- **BIT15,BIT14**：此两位组合为 00 设置为 BC 模式，10 设置为 RT 模式,01 设置为 BM 模式, 上电默认为 BC 模式。
- **BIT11**： 该位为 0 则 RT 状态字寄存器的 BIT1 位为 1。
- BIT10： 该位为 0 则 RT 状态字寄存器的 BIT3 位为 1。
- BIT9： 该位为 0 则 RT 状态字寄存器的 BIT8 位为 1。
- BIT8： 该位为 0 则 RT 状态字寄存器的 BIT2 位为 1。
- BIT7： 该位为 0 则 RT 状态字寄存器的 BIT0 位为 1。
- BIT0： 该位在 BC 总线控制器每个消息开始传输时置为 1，在消息结束传输时清为 0。

4.5 存器 1 (CFG1-BM)

地址：0x01

表 4-5 配置寄存器 1(CFG1-BM)

位（BIT）	描述 (DESCRIPTION)
15(MSB)	BC/RT/BM 模式设置（RT/BM*-BC*）,(logic 0)
14	BC/RT/BM 模式设置（BM/RT*- BC*）,(logic1)
13	A*/B 区域设置（CURRENT AREA A*/B）
12	保留
11	保留
10	保留
9	保留
8	保留
7	保留

位（BIT）	描述 （DESCRIPTION）
6	保留
5	保留
4	保留
3	保留
2	保留
1	保留
0	BM 忙指示（BM Busy），*该位只读*

- **BIT15,BIT14**：此两位组合为 00 设置为 BC 模式，10 设置为 RT 模式,01 设置为 BM 模式, 上电默认为 BC 模式。
- **BIT13**：该位为 0 则使用 RAM 的 A 区域，该位为 1 则使用 RAM 的 B 区域。
- BIT0： 该位在 BM 接收每个消息时置为 1，在消息结束时清为 0。

4.6 存器 2 (CFG2)

地址：0x02

表 4-6 配置寄存器 2(CFG2)

位（BIT）	描述 （DESCRIPTION）
15	保留
14	保留
13	忙查找表使能(BUSY LOOK UP TABLE ENABLE)
12-10	保留
9-7	时间标签最小精度设置（TIME TAG RESOLUTION2，1，0）
6	同步清除时标寄存器使能（CLEAR TIME TAG ON SYNCHRONIZE）
5	同步重载时标寄存器使能（LOAD TIME TAG ON SYNCHRONIZE）

位（BIT）	描述 （DESCRIPTION）
4	中断状态自动清除（INTERRUPT STATUS AUTO CLEAR）
3	电平/脉冲中断(LEVEL/PULSE *INTERRUPT REQUEST)
2	清除服务请求(CLEAR SERVICE REQUEST)
1-0	保留（其中 BIT1 可进行读写，但没有实际意义）

- BIT13： 该位为 1 则使能 RT 的忙位查找表。
- BIT9，BIT8，BIT7：000 则最小精度为 64us，001 则最小精度为 32us，010 则最小精度为 16us，011 则最小精度为 8us，100 则最小精度为 4us，101 则最小精度为 2us，110 则最小精度为 1us，111 则最小精度为 128us。当传输速率为 10Mbps 时则最小精度缩小 5 倍，也就是 000 则最小精度为 12.8us，001 则最小精度为 6.4us 依此类推。
- BIT6： 该位为 1 则当 RT 收到同步方式字(方式代码为 00001)时 RT 的时间标签寄存器清 0。
- BIT5： 该位为 1 则当 RT 收到同步方式字(方式代码为 10001)时 RT 的时间标签寄存器重新导入方式代码带的数据值。
- BIT4： 该位为 1 则 CPU 读出中断状态寄存器的值后，中断状态字寄存器自动清 0。
- BIT3： 该位为 0 产生脉冲中断信号，为 1 则产生电平中断信号，在该芯片中建议用电平中断。
- BIT2： 该位为 1 则当 RT 收到方式字(方式代码为 10000)时，将自动将服务请求撤消。也就是将 RT 配置寄存器 1 的 BIT9 置 1， RT 状态寄存器的 BIT8 置 0。

4.7 复位寄存器（SRR)

地址：0x03

表 4-7 启动/复位寄存器（SRR）

位（BIT）	描述 (DESCRIPTION)
15-7	保留
6	BC 停止消息发送（BC STOP-ON-MESSAGE）
5	BC 停止帧发送（BC STOP-ON-FRAME）

位（BIT）	描述 (DESCRIPTION)
4	保留
3	时间标签寄存器清零（TIME TAG RESET）
2	中断状态寄存器清零（INTERRUPT RESET）
1	BC/BM 启动（BC/BM START）
0	系统软复位（RESET）

启动 / 复位寄存器(SSR)用作“命令”类型的功能，能实现软复位，BC 启动，中断状态寄存器复位，时间标签寄存器(TTR)复位，在帧自动重复发送时还可以停止帧的自动复重发送。

- **BIT6**： 置 1 则在一个正在发送的消息发送完毕后即停止 BC 工作，如果没有消息在处理则立即停止 BC 工作。
- **BIT5**： 置 1 则在一个正在发送的帧发送完毕后即停止 BC 工作，如果没有帧在处理则立即停止 BC 工作。

BIT3： 置 1 清时间标签寄存器为 0。

- **BIT2**： 置 1 除了中断状态寄存器 的 BIT 7 位（RT 地址奇偶位错）不被清除其余位均被清除到 0。
- **BIT1**： 置 1 时在 BC 模式下启动帧传输;在 BM 模式下启动 BM 监视。。
- **BIT0**： 置 1 则进行软复位，在 BC/RT 模式时立即停止正在进行的处理。所有的寄存器和内部状态都被复位到上电时的初始态。

4.8 存器（STACK_ADDR)

地址：0x03

表 4-8 BC/RT 命令堆栈指针寄存器(STACK_ADDR)

位（BIT）	描述 （DESCRIPTION）
15-0	BC/RT/BM 命令堆栈指针

BC/RT/BM 命令堆栈寄存器主要寄存 BC/RT/BM 命令堆栈指针, 当作 BC 时将消息数据读出后该指针递增 4；当作 RT 时 RT 接到新的消息时该指针递增 4； 当作 BM 时 BM 接到新的消息时该指针递增 4。

4.9 初始命令堆栈指针寄存器（INIT_STACK_ADDR)

地址：0x04

表 0-9 BM 初始命令堆栈指针寄存器（INIT_STACK_ADDR)

位（BIT）	描述 (DESCRIPTION)
15-0	BM 命令堆栈指针初始位置

BM 初始命令堆栈指针寄存器主要用于设置最初的命令堆栈指针,默认为 0X0000H 即从 RAM 的第一个单元开始保存接收到的数据。

4.10 时间标签寄存器 0（TTR0）

地址：0x05

表 4-10 时间标签寄存器 0（TTR）

位（BIT）	描述 （DESCRIPTION）
15-0	时间计时标签位

时间标签寄存器 0 用于寄存 OBT1553 计时结果的 BIT15-BIT0 位。

4.11 中断状态寄存器（INT_STA）

地址：0x06

表 4-11 中断状态寄存器（INT_STA）

位（BIT）	描述 (DESCRIPTION)
15	中断请求（MASTER INTERRUPT）
14	保留
13	BC/RT 传输器超时（BC/RT TRANSMITTER TIMEOUT）
12	BC/RT 命令堆栈溢出/BM 命令堆栈半满溢出
11	BM 命令堆栈溢出（BM COMMAND STACK ROLLOVER）
10	BM 数据堆栈溢出（BM DATA STACK ROLLOVER）
9	保留
8	BC 重发（BC RETRY）/BM 数据半满溢出

位（BIT）	描述 (DESCRIPTION)
7	RT 地址奇偶校验错误（RT ADDRESS PARITY ERROR）
6	时间标签寄存器溢出（TIME TAG ROLLOVER）
5	RT 循环缓存溢出（RT CIRCULAR BUFFER ROLLOVER）
4	BC 消息/RT 子地址控制字消息结束（BC MSG/RT SUBADDRESS CONTROL WORD EOM）
3	BC 帧结束（BC END OF FRAME）
1	BC 状态置位/RT 方式代码（BC STATUS SET/RT MODE CODE）
0	消息结束（END OF MESSAGE）

- **BIT15**： BIT14-BIT0 中的任意一位为 1 则该位为 1。
- **BIT13**： 芯片内置看门狗，当传输编码时间超过 668us 时该位置 1。
- **BIT12**： 芯片命令堆栈大小为 256 字，当在 BC/RT 类型下命令堆栈指针超出 256 时该位置 1；当在 BM 类型下则在命令堆栈半满时该位置 1。
- **BIT11**： BM 命令堆栈溢出时则该位为 1。
- **BIT10**： BM 数据堆栈溢出时则该位为 1。
- **BIT8**： 在 BC 类型下 BC 重发消息前将时该位置 1；如果在 BM 类型下数据堆栈半满溢出时该位置 1。
- **BIT7**： 在 RT 模式下那么如果 RTAD4-RTAD0 与 RTADP 共六位进行奇偶的结果是 0 就时该位置 1。
- **BIT6**： 当时标寄存器计时到 65535 个单位（由最小计时精度确定）时该位置 1。
- **BIT5**： RT 循环缓存溢出时该位置 1。
- **BIT4**：当作 BC 总线控制器时 BC 控制字中的 BIT4 位为 1 则消息结束时该位置 1；当作 RT 时当 RT 的子地址控制器的 BIT14 或 BIT9 或 BIT4 中的任意一位为 1 时该位置 1。
- **BIT3**： BC 帧发送结束时该位置 1。
- **BIT2**： 格式错误时该位置 1，格式错误是指响应超时、奇偶校验错、编码错、计数错等。
- **BIT1**： 当作 BC 时，BC 收到的状态字中有置 1 的位，则产生中断；当作 RT 时子

地址查找表中方式字对应位为 1 时该位置 1。

- **BIT0**： 当 BC/RT 发送/接收消息结束时该位置 1。

4.12 配置寄存器 3（CFG3）

地址：0x07

表 4-12 配置寄存器 3(CFG3)

位（BIT）	描述 (DESCRIPTION)		
15-13	保留		
12-11	BM 命令堆栈大小设置位 1,0	BIT12,BIT11	全满消息条数
		00	20
		01	40
		10	80
		11	160
10-8	BM 数据堆栈大小设置位 2-0	BIT10,BIT9,BIT8	全满字个数
		000	3328
		001	1664
		010	832
		011	416
		100	208
		101	624
		110	1248
		111	2496
7	非法命令查找表使能（ILLEGALIZATION DISABLED）		
4	接收非法命令屏蔽（ILLEGAL RX TRANSFER DISABLE）		
3	接收忙屏蔽(BUSY RX TRANSFER ENABLE)		
2-1	保留		
0	增强方式代码功能(ENHANCED MODE CODE HANDLING)		

- **BIT12,BIT11：** 这两位为 BM 命令堆栈大小设置位，“00”时表示收到 20 条消息时全满溢出，半满溢出则是收到 10 条消息；“01”时表示收到 40 条消息时全满溢出，半满溢出则是收到 20 条消息；“10”时表示收到 80 条消息时全满溢出，半满溢出则是收到 40 条消息；“11”时表示收到 160 条消息时全满溢出，半满溢出则是收到 80 条消息。
- **BIT10-BIT8：**这三位为 BM 数据堆栈大小设置位，当全满溢出时接收到的数据字个数如表中所示，半满溢出则少一半。如“000”时表示收到 3328 个数据字时全满溢出，半满溢出则是收到 1664 个数据字。
- **BIT7：** 该位为 1，使能 RT　RAM 中的非法命令查找表。
- **BIT4：** 该位为 0，则在接收到非法命令时将接收到的数据写入 RAM 中，否则在接收到非法命令时不将接收到的数据写入 RAM 中。
- **BIT3：** 该位为 0，则在忙时将接收到的数据写入 RAM 中，否则在忙时不将接收到的数据写入 RAM 中。
- **BIT0：** 该位为 1，使能 RT　RAM 中的方式代码查找表。

4.13 配置寄存器 4(CFG4)

地址：0x08

表 4-13　配置寄存器 4(CFG4)

位（BIT）	描述 (DESCRIPTION)
15-9	保留
8	第一次重发通道选择（FIRST RETRY ALT/SAME* BUS）
7	第二次重发通道选择（SECOND RETRY ALT/SAME* BUS）
6-4	保留
3	RT 地址配置使能(LATCH RT ADDRESS WITH CFG REG #5)
2-0	保留

- **BIT8：** 置 0 则在最初发送的消息失败后第一次重发的消息与最初发送的消息在同一通道上传输。置 1 则在最初发送的消息失败后第一次重发的消息不再最初发送的消息的通道上传输。

- **BIT7**： 置 0 则在第一次重发的消息失败后第二次重发的消息与第一次重发的消息在同一通道上传输。置 1 则在第一次重发的消息失败后第二次重发的消息不再第一次重发的消息的通道上传输。此位只有在配置寄存器 1（CFG1）的 BIT3 位为 1 才有效。
- **BIT3**： 当该位为 1 时配置寄存器 5 的 BIT5-BIT0 才可写。

4.14 配置寄存器 5 (CFG5)

地址：0x09

表 4-14 配置寄存器 5(CFG5)

位（BIT）	描述 (DESCRIPTION)
15-11	保留
10-9	超时响应时间设置(RESPONSE TIMEOUT SELECT1,0)
8-6	保留（其中 BIT6 可进行读写，但没有实际意义）
5-1	RT 地址位 4-0(RT ADDRESS4-ADDRESS0)
0	RT 地址奇偶位(RT ADDRESS PARITY)

- **BIT10.BIT9**： 超时响应时间设置,如为 00 是 19us, 01 是 23us, 10 是 51us, 11 是 130us。当为 10Mbps 的传输速度时则均为 1M 的 0.2 倍，如 00 是 3.8us。
- **BIT5-BIT1**： 配置 RT 地址。
- **BIT0**： 配置 RT 校验位，该位与 BIT5-BIT1 的异或结果要为 1。

4.15BM 数据堆栈指针寄存器（BM_STACK_ADDR)

地址：0x0A

表 4-15 BM 数据堆栈指针寄存器（BM_STACK_ADDR)

位（BIT）	描述 (DESCRIPTION)
15-0	BM 数据堆栈指针

BM 数据堆栈寄存器主要寄存 BM 数据堆栈指针, BM 接到新的数据时该指针递增 1。

4.16BC 帧时间/RT 上一命令字寄存器(LAST_CMD)

地址：0x0D

表 4-16 BC 帧时间/RT 上一命令字寄存器（LAST_CMD）

位（BIT）	描述 (DESCRIPTION)
15-0	BC 帧时间/RT 上一命令字寄存器

该寄存器用作 BC 帧时间设置寄存器时为可写，RT 上一命令字时为可读。

4.17RT 状态字寄存器(RT_STA)

地址：0x0E

表 4-17 RT 状态字寄存器(RT_STA)

位（BIT）	描述 (DESCRIPTION)
15-11	均为 0
10	消息错误(MESSAGE ERROR)
9	测试手段(INSTRUMENTATION)
8	服务请求(SERVICE REQUEST)
7-5	保留
4	广播指令接收(BROADCAST COMMAND RECEIVED)
3	忙(BUSY)
2	子系统标志(SYBSYSTEM FLAG)
1	动态总线控制接收(DYNAMIC BUS CONTROL ACCEPT)
0	终端标志(TERMINAL FLAG)

- **BIT10**： 如该位为 1 则表明有消息错误。消息错误是指响应超时、奇偶校验错、编码错、计数错，非法命令等。
- **BIT9**： 该位在 OBT1553B IP 中一直为 0。
- **BIT8**： 如该位为 1 则表明 RT 有服务请求。
- **BIT4**： 如该位为 1 则表明通讯方式是广播通讯方式。
- **BIT3**： 如该位为 1 则表明系统正忙。

- **BIT2**： 子系统标志位。
- **BIT1**： 动态总线控制接收标志位。
- **BIT0**： 终端标志位。

4.18RT BIT 字寄存器(RT_BIT_REG)

地址：0x0F

表 4-18 RT BIT 字寄存器(RT_BIT_REG)

位（BIT）	描述 (DESCRIPTION)
15	传输器超时（TRANSMITTER TIMEOUT）
14-12	保留
11	通道 B 发送器关闭（TRANSMITTER SHUTDOWN B）
10	通道 A 发送器关闭（TRANSMITTER SHUTDOWN A）
9	终端标志禁止(TERMINAL FLAG INHIBITED)
8	总线传输通道 B/A*(CHANNEL B/A*)
7	保留
6	字计数错（WORD COUNT ERROR）
5	错误数据同步头（INCORRECT SYNC RECEIVED）
4	奇偶/位计数错(PARITY/BIT COUNT ERROR)
3	RT-RT 同步头/地址错(RT-RT SYNC/ADDRESS ERROR)
2	RT-RT 响应超时（RT-RT NO RESPONSE ERROR)
1	RT-RT 第二个命令字错(RT-RT 2ND COMMAND WORD ERROR)
0	命令字内容错(COMMAND WORD CONTENTS ERROR)

- **BIT15**： 传输器超时该位置 1。
- **BIT11**： 通道 B 发送器关闭该位置 1。
- **BIT10**： 通道 A 发送器关闭该位置 1。
- **BIT9**： 终端标志禁止该位置 1。
- **BIT8**： 消息传输在 A 通道进行为 0，消息传输在 B 通道进行为 1。
- **BIT6**： 字计数错该位置 1。

- **BIT5**： 错误数据同步头该位置 1。
- **BIT4**： 奇偶/位计数错该位置 1。
- **BIT3**： RT-RT 同步头/地址错则该位置 1。
- **BIT2**： RT-RT 响应超时则该位置 1。
- **BIT1**： RT-RT 第二个命令字错如奇偶错则该位置 1。
- **BIT0**： RT-RT 第二个命令字内容出错如地址错则该位置 1。

说明：从表 3-19 开始后面的表不是寄存器，它们占用 RAM 空间。

4.19BC 控制字（BC_CTRL）

表 4-19 BC 控制字（BC_CTRL）

位（BIT）	描述 (DESCRIPTION)
15	保留
14	消息格式错误屏蔽（M.E.MASK）
13	服务请求位屏蔽(SERVICE REQUEST BIT MASK)
12	忙位屏蔽(SUBSYS BUSY BIT MASK)
11	子系统标志位屏蔽(SUBSYS FLAG BIT MASK)
10	终端标志位屏蔽(TERMINAL FLAG BIT MASK)
9	保留
8	重试使能(RETRY ENABLED)
7	总线通道选择 A/B*(bus channel a/b*)
6	保留
5	保留
4	EOM 中断使能(EOM INTERRUPT ENABLE)
3	保留
2	模式命令(MODE CODE FORMAT)
1	广播命令(BROADCAST FORMAT)

位（BIT）	描述 (DESCRIPTION)
0	RT2RT(RT-TO-RT FORMAT)

- **BIT14-BIT10**： 这 5 位均是为 1 则屏蔽相应的 RT 状态字位，如果 RT 状态字位某位被屏蔽则该位不影响中断状态寄存器的（INT_STA）的 BIT1 位，但影响 BC 块状态字的 BIT7 位。
- **BIT8**： 置 1 且配置寄存器 1（CFG1）的 bit4 为 1 则在响应超时和格式错误时消息重发。
- **BIT7**： 置 1 消息传输通过 A 通道，置 0 消息传输通过 B 通道。
- **BIT4**： 该位置 1 且中断屏蔽寄存器（IMR）的 BIT4 也置 1 那么则消息结束中断状态寄存器(INT_STA)的 BIT4 为 1。
- **BIT2,BIT1,BIT0**：置为 000 且命令字的 T/R*位是 0 则是 BC-> RT 通讯方式, 置为 000 且命令字的 T/R*位是 1 则是 RT->BC 的通讯方式； 置为 001 是 RT-RT 通讯方式；置为 010 是 Broadcast 通讯方式；置为 100 是 Mode Code 通讯方式； 置为 110 是 Broadcast Mode Code 通讯方式。 其它两种组合没用到。

4.20BC 命令字（BC_CMD）

表 4-20 BC 命令字(BC_CMD)

位（BIT）	描述 (DESCRIPTION)
15-11	远程终端地址（REMOTE TERMINAL ADDRESS）
10	发送/接收* (T/R*)
9-5	子地址/方式(SUBADDRESS/MODE)
4-0	数据字计数/方式代码(DATA WORD COUNT/MODE CODE)

4.21BC 块状态字(BC_BLK)

表 4-21 BC 块状态字(BC_BLK)

位（BIT）	描述 (DESCRIPTION)
15	消息传输结束标志位（EOM）
14	保留

位（BIT）	描述 (DESCRIPTION)
13	传输通道指示(CHANNEL B/A*)
12	出错标志(ERROR FLAG)
11	状态设置（STATUS SET）
10	格式错误(FORMAT ERROR)
9	响应超时(NO RESPONSE TIMEOUT)
8	保留
7	屏蔽状态设置（MASKED STATUS SET）
6-5	重发消息次数(RETRY COUNT 1,0)
4	数据传输正常（GOOD DATA BLOCK TRANSFER）
3	错误的状态地址（WRONG STATUS ADDRESS）
2	字计数错误（WORD COUNT ERROR）
1	同步头出错(INCORRECT SYNC TYPE)
0	无效字（INVALID WORD）

该字主要用来说明消息发送/接收后的状态。该存储器字如果有重发消息，只表明的是最后一个消息的状态。

- **BIT15**： 消息转输结束该位置 1。
- **BIT13**： 消息传输在 A 通道进行为 0，消息传输在 B 通道进行为 1。
- **BIT12**： 有格式错误或响应超时产生该位为 1。
- **BIT11**： 接受的 RT 返回字中 BIT10-BIT0 中只要有一位为 1 则该位为 1。
- **BIT10**： 格式错误如计数个数、同步头、奇偶等错误则该位为 1。
- **BIT9**： 响应超时则该位置 1。
- **BIT6,5**： 记录 BC 重发消息的个数，00 表示没有重发，01 表示重发了一次，11 表示重发了两次。
- **BIT4**： 当 RT2BC,RT2RT,或者传输带数据方式代码，没有消息格式错则为 1，有消息格式错为 0；当 BC2RT，带数据方式代码收和不带数据方式代码，为 0。
- **BIT3**： RT 返回的状态字中 RT 地址有错则该位为 1。

- **BIT2**： 当 RT2BC,RT2RT,或者传输带数据方式代码，字计数有错则为 1，没错为 0；当 BC2RT，带数据方式代码收和不带数据方式代码，为 0。
- **BIT1**： 同步头出错该位置 1。
- **BIT0**： 当数据传输中有奇偶错、位计数错和同步头错则该位置 1。

4.22RT 子地址控制字(RT_SUB_CTRL)

表 4-22 RT 子地址控制字(RT_SUB_CTRL)

位（BIT）	描述 (DESCRIPTION)
15	RX：接收双缓存使能（DOUBLE BUFFER ENABLE）
14	TX：发送消息结束中断使能（EOM INT）
13	TX：发送循环缓存溢出使能(CIRC BUF INT)
12-10	TX：发送存储器管理设置（MEMORY MANAGEMENT2,1,0）
9	RX：接收消息结束中断使能（EOM INT）
8	RX：接收循环缓存溢出使能(CIRC BUF INT)
7-5	RX：接收存储器管理设置（MEMORY MANAGEMENT2,1,0）
4	BCST：广播消息结束中断使能（EOM INT）
3	BCST：广播循环缓存溢出使能(CIRC BUF INT)
2-0	BCST：广播存储器管理设置（MEMORY MANAGEMENT2,1,0）

- **BIT15**： 该位为 1 则选择 RT 为双缓冲存储器管理方式，为 0 则为单缓冲或循环缓冲存储器管理方式。
- **BIT14**： 置 1 发送消息结束中断使能。
- **BIT13**： 置 1 发送循环缓存溢出使能。
- **BIT12-BIT10**：发送存储器管理设置,000 是单消息模式，001 是在循环缓存中大小为 128 字，010 是 256 字依此类推。
- **BIT9**： 置 1 接收消息结束中断使能。
- **BIT8**： 置 1 接收循环缓存溢出使能。
- **BIT7-BIT5**： 接收存储器管理设置，000 是单消息模式，001 是在循环缓存中大小为 128 字，010 是 256 字依此类推。

- **BIT4**： 置 1 广播消息结束中断使能。
- **BIT3**： 置 1 广播循环缓存溢出使能。
- **BIT2-BIT0**： 广播存储器管理设置，000 是单消息模式，001 是在循环缓存中大小为 128 字，010 是 256 字依此类推。

4.23RT/BM 块状态字(RT/BM_BLK)

表 4-23 RT/BM 块状态字(RT/BM_BLK)

位（BIT）	描述 (DESCRIPTION)
15	消息传输结束标志（EOM）
14	消息传输开始标志（SOM）
13	传输通道指示（CHANNEL B/A*）
12	出错标志(ERROR FLAG)
11	RT-RT 通讯方式标志(RT-RT FORMAT)
10	格式错误标志(FORMAT ERROR)
9	响应超时标志(NO RESPONSE TIMEOUT)
8	保留
7	循环缓存溢出（DATA STACK ROLLOVER）
6-5	非法命令字(ILLEGAL COMMAND WORD)
4	字计数个数错(WORD COUNT ERROR)
3	数据同步头出错(INCORRECT DATA SYNC)
2	无效字（INVALID WORD）
1	RT-RT 同步头/地址错（RT-RT SYNCH/ADDRESS ERROR）
0	RT-RT 第二个命令字错（RT-RT 2ND COMMAND ERROR）

- **BIT15**： 消息传输结束该位置 1。
- **BIT14**： 消息传输开始该位置 1。
- **BIT13**： 消息传输在 A 通道进行为 0，消息传输在 B 通道进行为 1。
- **BIT12**： 有格式错误或响应超时产生该位为 1。
- **BIT11**： 在 RT-RT 通讯时接收 RT 时被设置为 1， 发送 RT 时与该位无关。

- **BIT10**： 格式错误如计数个数，同步头，奇偶等错误则该位为 1。
- **BIT9**： RT-RT 通讯时响应超时则该位置 1。
- **BIT7**： 循环缓存溢出则该位置 1。
- **BIT6**： 非法命令字则该位置 1。
- **BIT5**： RT 接收的字个数出错时该位置 1。
- **BIT4**： 数据同步头出错出错时该位置 1。
- **BIT3**： 当数据传输中有奇偶错、位计数错则该位置 1。
- **BIT2**： RT-RT 同步头/地址错则该位置 1。
- **BIT1**： RT-RT 第二个命令字错如奇偶错则该位置 1。
- **BIT0**： RT-RT 第二个命令字内容出错如地址错则该位置 1。

5.功能模块描述

5.1 BC 总线控制器工作方式

5.1.1BC 存储器地址分配

表 5-1 BC 存储器地址分配(4K 双口 RAM)

地址（HEX）	描述
0000-00FF	堆栈 A（STACK A）
0100	堆栈指针 A (STACK POINTER A)
0101	消息个数设置 A (MESSAGE COUNT A)
0102-0103	保留
0104	堆栈指针 B (STACK POINTER B)
0105	消息个数设置 B (MESSAGE COUNT B)
0106-0107	保留
0108-012D	消息块 0(MESSAGE BLOCK0)
012E-0153	消息块 1(MESSAGE BLOCK1)
…	…
0ED6-0EFB	消息块 93(MESSAGE BLOCK 93)
…	…
0F00-0FFF	堆栈 B（STACK B）

5.1.2BC 存储器管理

BC 存储器管理如图 BC 存储器管理图所示。该图说明了命令堆栈区包含四个描述符，即块状态字，时间标签字，消息间隔时间字和消息块地址字。块状态字包括消息状态、完成、有效性及总线通道信息；时间标签字寄存了当前消息结束时时间标签寄存器的值；消息间隔时间字存储的是设定的消息间隔时间；消息

块地址字寄存的是指向消息块第一个字的地址。程序通过 RAM 的 0X0100H 地址取命令堆栈指针，0X0101 地址取消息个数值。

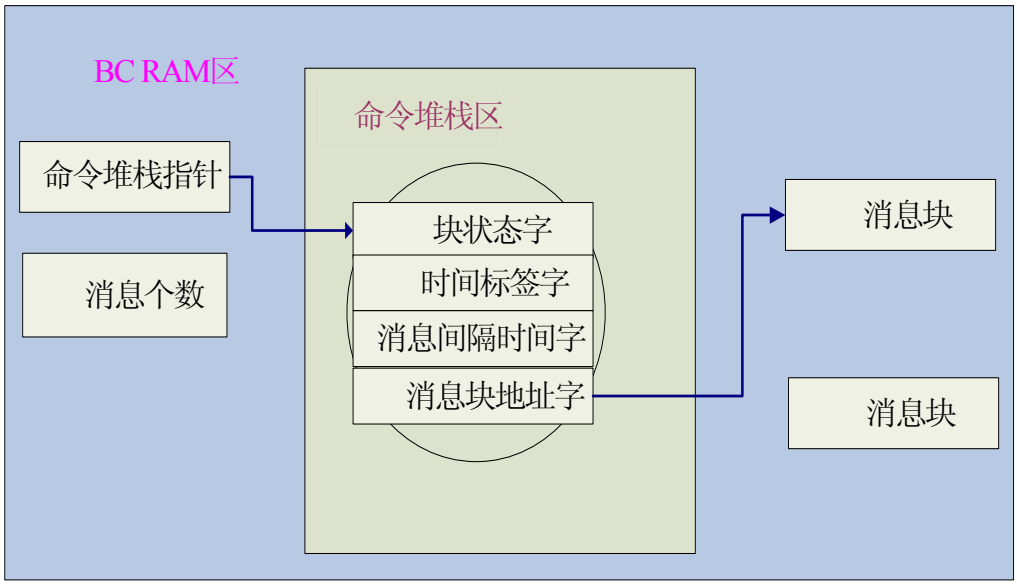


图 5-1 BC 存储器管理

5.1.3BC 消息格式

表 5-2 BC 消息格式

BC 到 RT 的传输	RT 到 BC 的传输	RT 到 RT 的传输	不带字的方式命令
控制字	控制字	控制字	控制字
接收命令字	发送命令字	接收命令字	方式命令字
数据字 1	发送命令字的回应字	发送命令字	方式回应字
数据字 2	状态字	发送命令字的回应字	状态字
…	数据字 1	发送终端的状态字	
	数据字 2	数据字 1	

最后一个数据字	…	…	
最后数据字的回应字		最后一个数据字	
状态字	最后一个数据字	接收终端的状态字	

表 5-3 BC 消息格式（接上表）

带字的发送方式命令	带字的接收方式命令	广播命令	不带字的广播方式命令	带字的广播方式命令
控制字	控制字	控制字	控制字	控制字
发送方式命令字	接收方式命令字	广播命令字	广播方式命令字	广播方式命令字
方式命令回应字	数据字	数据字 1	广播方式命令字的回应字	数据字
接收状态字	接收命令字的回应字	数据字 2 …		广播方式命令字的回应字
数据字	接收状态字	最后的数据字		

5.2RT 远程终端工作方式

5.2.1RT 存储器地址分配

表 5-4 RT 存储器地址分配(4K 双口 RAM)

地址（HEX）	描述
0000-00FF	堆栈 (STACK)
0100	堆栈指针(STACK POINTER)

地址（HEX）	描述
0101-0107	保留
0108-010F	方式代码选择中断表（MODE CODE SELECTIVE INTERRUPT TABLE）
0110-013F	方式代码数据(MODE CODE DATA)
0140-01BF	查找表(LOOKUP TABLE)
01C0-023F	保留
0240-0247	忙位查找表（BUSY BIT LOOKUP TABLE）
0248-025F	（没有使用）
0260-027F	数据块 0 (DATA BLOCK 0)
0280-02FF	数据块 1-4　(DATA BLOCK 1-4)
0300-03FF	非法命令表 (CINNABD UKKEGAKUZUBG TABKE)
0400-041F	数据块 5 (DATA BLOCK 5)
0420-043F	数据块 6 (DATA BLOCK 6)
…	…
0FE0-0FFF	数据块 100 (DATA BLOCK 100)

5.2.2RT 存储器查找表

表 5-5　RT 存储器查找表(LOOK_UP TABLE)

地址（HEX）	对应子地址	描述
0140 … 015F	Rx_SA0 … Rx _SA31	接收查找表
0160	Tx_SA0	发送查找表

… 017F	… Tx _SA31	
0180 … 19F	Bcst_SA0 … Bcst _SA31	广播查找表
01A0 … 01BF	SACW_SA0 … SACW_SA31	子地址控制字查找表

5.2.3RT 存储器非法命令表地址分配

表 5-6 RT 存储器非法命令地址分配表(COMMAND ILLEGALIZING TABLE)

位（BIT）	描述 (DESCRIPTION)
15-10	均为逻辑 0
9-8	均为逻辑 1
7	广播*/本身地址（BROADCAST*/OWN ADDRESS）
6	发送/接收*（T/R*）
5-1	子地址 4－子地址 0（SA4-SA0）
0	子计数 4/方式字 4（WC4/MC4）

RT 非法命令表，在 RT 中占用 0x300～0x3FF 的地址空间。当 RT 接收到命令字后，如果使能（为 1）非法化命令检测。通过广播/RT 地址、T/R*、SA4~SA0 和 WC4/MC4 共 8 位在 0x300～0x3FF 中，查找 WC3~WC0（MC3~MC0）收到的给 RT 某一子地址、某些个数的命令字是否非法。

5.2.4RT 存储器忙位查找表地址分配

表 5-7 RT 存储器忙位查找表地址分配表(BUSY BIT LOOKUP TABLE)

位（BIT）	描述 (DESCRIPTION)
15-10	均为逻辑 0
9	逻辑 1
8	逻辑 0
7	逻辑 0
6	逻辑 1
5-3	均为逻辑 0
2	广播/本身地址*（BROADCAST/OWN ADDRESS*）
1	发送/接收*（T/R*）
0	子地址 4（SA4）

RT 忙位查找表，在 RT 中占用 0x240～0x247 的地址空间。当 RT 接收到命令字后，如果使能（为 1）忙位查找表检测。通过广播/RT 地址、T/R*、SA4 共 3 位在 0x240～0x247 中，查找 SA3~SA0 收到的给 RT 某一子地址的命令字是否忙。

5.2.5RT 存储器方式代码选择中断表

表 5-8 RT 存储器方式代码选择中断表（MODE CODE SELECTIVE INTERRUPT TABLE）

位（BIT）	描述 (DESCRIPTION)
0108	接收方式命令 0-15(undefined)
0109	发送方式命令 16-31(WITH DATA)
010A	发送方式命令 0-15(WITHOUT DATA)
010B	发送方式命令 16-31(WITH DATA)

位（BIT）	描述 (DESCRIPTION)
010C	广播接收方式命令 0-15(undefined)
010D	广播接收方式命令 16-31(WITH DATA)
010E	广播发送方式命令 0-15(WITHOUT DATA)
010F	广播发送方式命令 16-31(UNDEFINED)

5.2.6RT 存储器方式代码选择中断地址分配

表 5-9 RT 存储器方式代码选择中断表地址分配表(MODE CODE SELECTIVE INTERRUPT TABLE)

位（BIT）	描述 (DESCRIPTION)
15-9	均为逻辑 0
8	逻辑 1
7	逻辑 0
6	逻辑 0
5	逻辑 0
4	逻辑 0
3	逻辑 1
2	广播/本身地址*（BROADCAST/OWN ADDRESS*）
1	发送/接收*（T/R*）
0	方式代码 4（MC4）

RT 方式代码选择中断表，在 RT 中占用 0x108～0x10F 的地址空间。当 RT 接收到命令字后，如果使能（为 1）式代码选择中断表检测。通过广播/RT 地址、T/R*、MC4 共 3 位在 0x108～0x10F 中，查找 MC3~MC0 收到的给 RT 某一方式代码是否有中断。

5.2.7RT 方式代码数据表

表 5-10 RT 存储器方式代码数据表（MODE CODE DATA）

地 址（HEX）	方式代码（MODE CODE）
0110	没有定义
0111	同步带数据
0112-11F	没有定义
0120	发送矢量字
0121-13F	没有定义

5.2.8 芯片实现的方式代码

表 5-11 芯片实现的方式代码

发/收*	方 式 代 码	功能	是否带数据字	是否允许广播指令
1	00000	动态总线控制	否	否
1	00001	同步	否	是
1	00010	发送上一状态字	否	否
1	00011	启动自测试[1]	否	是
1	00100	发送器关闭	否	是
1	00101	取消发送器关闭	否	是
1	00110	禁止终端标志位	否	是
1	00111	取消禁止终端标志位	否	是
1	01000	复位远程终端[2]	否	是
1	01001	备用	否	待定
…	…	…	…	…

发/收*	方 式 代 码	功能	是否带数据字	是否允许广播指令
1	01111	备用	否	待定
1	10000	发送矢量字	是	否
0	10001	同步[3]	是	是
1	10010	发送上一指令字	是	否
1	10011	发送自检测字	是	否

5.2.9RT 单缓冲存储器管理

RT 单缓冲存储器管理如图 RT 单缓冲存储器管理图所示。该图说明了命令堆栈区包含四个描述符，即块状态字，时间标签字，数据块指针字和接收命令字。块状态字包括消息状态、完成、有效性及总线通道信息；时间标签字寄存了当前消息结束时时间标签寄存器的值；数据块指针字存储指向数据块的起始地址；接收命令字存储 RT 接收到的命令字。程序通过 RAM 的 0X0100H 地址取命令堆栈指针。

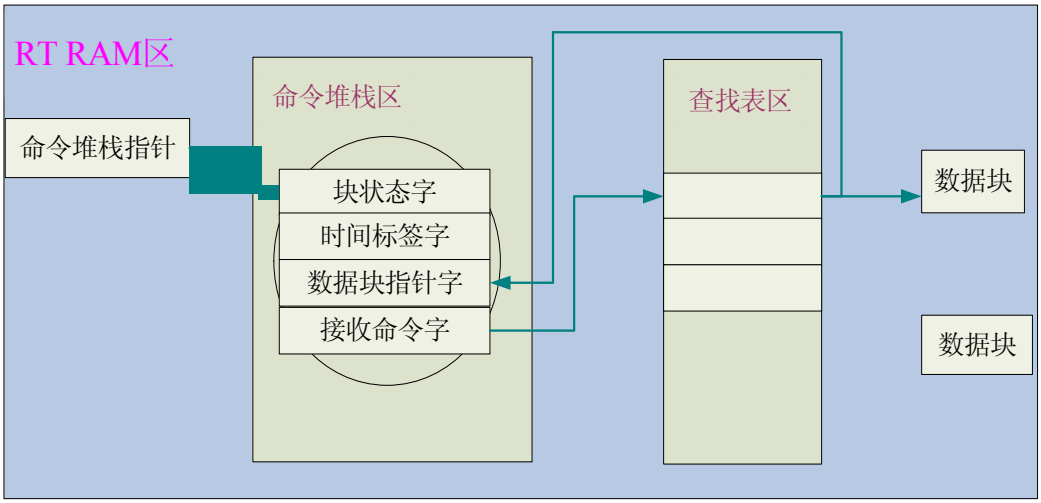


图 5-2 RT 单缓冲存储器管理

5.2.10RT 循环缓冲存储器管理

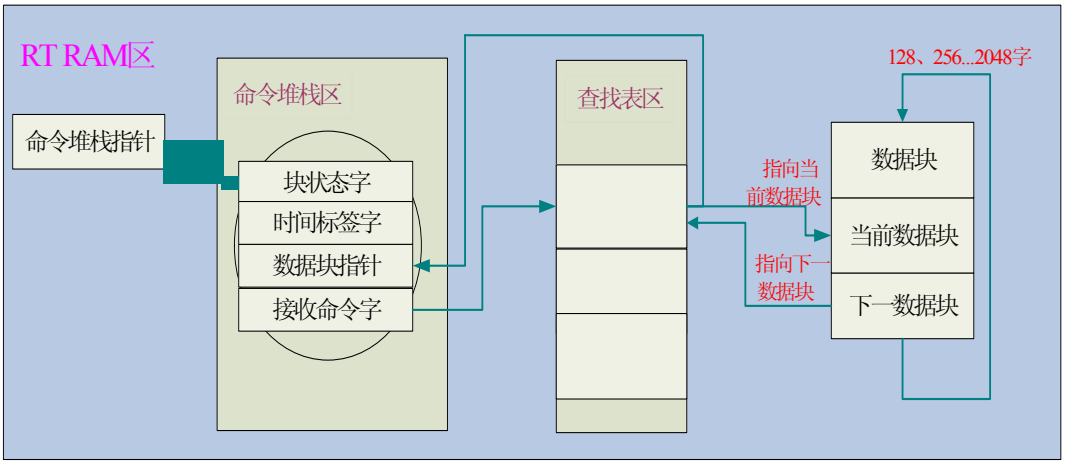


图 5-3 RT 循环缓冲存储器管理

5.2.11RT 双缓冲存储器管理

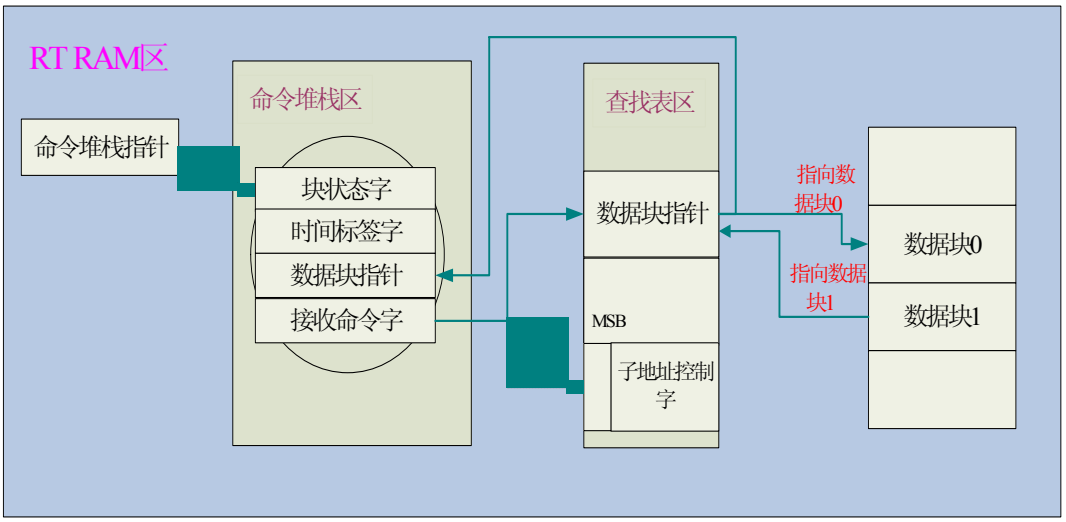


图 5-4 RT 双缓冲存储器管理

5.3BM 总线监视器工作方式

5.3.1BM 存储器地址分配

表 5-12 BM 存储器地址分配

地址（HEX）	描述
0000-027F	命令堆栈区域 (STACK AREA)

地址（HEX）	描述
0280-02FF	子地址选择设置区域(SUBADDRESS SELECT AREA)
0300-0FFF	数据块区域(DATA BLOCK AREA)

5.3.2BM 存储器管理

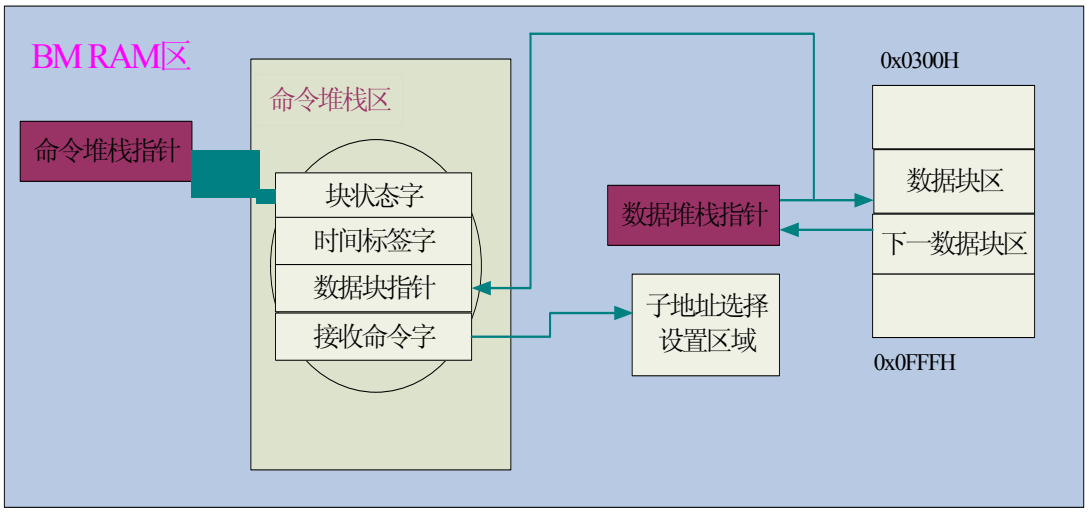


图 5-5 BM 存储器管理

5.3.3BM 子地址选择设置区地址分配

表 5-13 BM 子地址分配

位（BIT）	描述 (DESCRIPTION)
15-11	逻辑 0
10	逻辑 0
9	逻辑 1
8	逻辑 0
7	逻辑 1
6	RT 地址 4（RT ADDRESS 4）
5	RT 地址 3（RT ADDRESS 3）

位（BIT）	描述 (DESCRIPTION)
4	RT 地址 2（RT　ADDRESS 2）
3	RT 地址 1（RT　ADDRESS 1）
2	RT 地址 0（RT　ADDRESS 0）
1	发送/接收*（T/R*）
0	子地址 4（SA4）

BM 子地址选择设置区地址分配表，在 BM 中占用 0x280～0x2FF 的地址空间。当 BM 接收到命令字后，如果使能（为 1）子地址选择设置检测。通过 RT 地址、T/R*、SA4 共 7 位在 0x280～0x2FF 中，查找 SA3~SA0 收到的给 BM 某一子地址的内容接收，否则不做接收。

6 封装和信号定义

6.1 PQFP208 塑料封装尺寸

塑封 PQFP208 尺寸图：

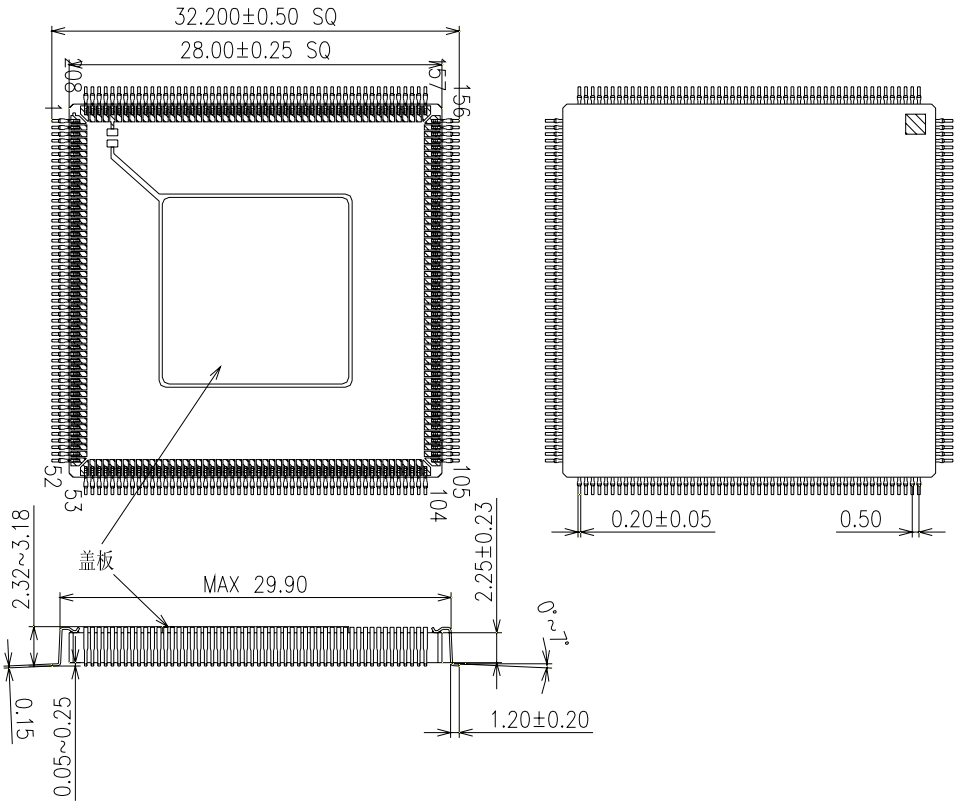


图 6-1 PQFP208 封装

6.2 PQFP208 塑料封装信号定义

表 6-1 信号描述

引脚编号	引脚名	类型	功能	有效
1	ADDR[0]	输出	外部地址线	高
2	ADDR[1]	输出	外部地址线	高
3	ADDR[2]	输出	外部地址线	高
4	ADDR[3]	输出	外部地址线	高
5	VDDH	——	+3.3V	——
6	VSSH	——	地	——
7	ADDR[4]	输出	外部地址线	高
8	ADDR[5]	输出	外部地址线	高
9	ADDR[6]	输出	外部地址线	高
10	ADDR[7]	输出	外部地址线	高
11	VDD	——	+1.2V	——
12	GND	——	地	——
13	ADDR[8]	输出	外部地址线	高
14	ADDR[9]	输出	外部地址线	高
15	ADDR[10]	输出	外部地址线	高
16	ADDR[11]	输出	外部地址线	高
17	VDDH	——	+3.3V	——
18	VSSH	——	地	——
19	ADDR[12]	输出	外部地址线	高

20	ADDR[13]	输出	外部地址线	高
21	ADDR[14]	输出	外部地址线	高
22	ADDR[15]	输出	外部地址线	高
23	ADDR[16]	输出	外部地址线	高
24	ADDR[17]	输出	外部地址线	高
25	VDD	——	+1.2V	——
26	GND	——	地	——
27	ADDR[18]	输出	外部地址线	高
28	ADDR[19]	输出	外部地址线	高
29	ADDR[20]	输出	外部地址线	高
30	ADDR[21]	输出	外部地址线	高
31	ADDR[22]	输出	外部地址线	高
32	ADDR[23]	输出	外部地址线	高
33	VDDH	——	+3.3V	——
34	VSSH	——	地	——
35	ADDR[24]	输出	外部地址线	高
36	ADDR[25]	输出	外部地址线	高
37	ADDR[26]	输出	外部地址线	高
38	ADDR[27]	输出	外部地址线	高
39	WRI	输出	写允许	低
40	DATA[16]	双向	存贮器 数据	高
41	VDD	——	+1.2V	——

42	GND	——	地	——
43	DATA[17]	双向	外部数据线	高
44	DATA[18]	双向	外部数据线	高
45	DATA[19]	双向	外部数据线	高
46	DATA[20]	双向	外部数据线	高
47	DATA[21]	双向	外部数据线	高
48	DATA[22]	双向	外部数据线	高
49	VDD	——	+1.2V	——
50	GND	——	地	——
51	DATA[23]	双向	外部数据线	高
52	DATA[24]	双向	外部数据线	高
53	DATA[25]	双向	外部数据线	高
54	DATA[26]	双向	外部数据线	高
55	DATA[27]	双向	外部数据线	高
56	DATA[28]	双向	外部数据线	高
57	DATA[29]	双向	外部数据线	
58	DATA[30]	双向	外部数据线	高
59	VDDH	——	+3.3V	——
60	VSSH	——	地	——
61	DATA[31]	双向	外部数据线	高
62	READ	输出	读允许	高
63	OEN	输出	输出允许	低

64	RAMSN[4]	输出	静态存贮器片选	低
65	RAMSN[3]	输出	静态存贮器片选	低
66	VDD	——	+1.2V	——
67	GND	——	地	——
68	RAMSN[2]	输出	静态存贮器片选	低
69	RAMSN[1]	输出	静态存贮器片选	低
70	RAMSN[0]	输出	静态存贮器片选	低
71	BRDYN	输入	总线准备好	低
72	BEXCN	输入	总线异常	低
73	VDDH	——	+3.3V	——
74	VSSH	——	地	——
75	CLKSD	输出	动态随机存贮器时钟域 1	——
76	CLKSD2	输出	动态随机存贮器时钟域 2	——
77	VDD	——	+1.2V	——
78	GND	——	地	——
79	SDCKE[1]	输出	动态随机存贮器时钟允许	高
80	SDCKE[0]	输出	动态随机存贮器时钟允许	高
81	SDCSN[1]	输出	动态随机存贮器片选	低
82	SDCSN[0]	输出	动态随机存贮器片选	低

83	SDWEN	输出	动态随机存贮器写允许	低
84	SDRASN	输出	动态随机存贮器行地址允许	低
85	VDD	——	+1.2V	——
86	GND	——	地	——
87	SDCASN	输出	动态随机存贮器列地址允许	低
88	SDDQM[3]	输出	动态随机存贮器数据掩码	低
89	SDDQM[2]	输出	动态随机存贮器数据掩码	低
90	SDDQM[1]	输出	动态随机存贮器数据掩码	低
91	SDDQM[0]	输出	动态随机存贮器数据掩码	低
92	VDDH	——	+3.3V	——
93	VSSH	——	地	——
94	RAMOEN[4]	输出	静态存贮器输出允许	低
95	RAMOEN[3]	输出	静态存贮器输出允许	低
96	RAMOEN[2]	输出	静态存贮器输出允许	低
97	RAMOEN[1]	输出	静态存贮器输出允许	低
98	RAMOEN[0]	输出	静态存贮器输出允许	低
99	VDD	——	+1.2V	——

100	GND	——	地	——
101	GPIO[0]	双向	通用接口 I/O	——
102	GPIO[1]	双向	通用接口 I/O	——
103	GPIO[2]	双向	通用接口 I/O	——
104	GPIO[3]	双向	通用接口 I/O	——
105	GPIO[4]	双向	通用接口 I/O	——
106	GPIO[5]	双向	通用接口 I/O	——
107	GPIO[6]	双向	通用接口 I/O	——
108	GPIO[7]	双向	通用接口 I/O	——
109	VDDH	——	+3.3V	——
110	VSSH	——	地	——
111	NC	——		——
112	NC	——		——
113	AVDD	——	+1.2V	——
114	AGND	——	地	——
115	VDD	——	+1.2V	——
116	GND	——	地	——
117	TMODE[0]	输入	时钟倍频设置： 当 tmode[1..0] = 00 表示 5 倍频，01 表示 10 倍频，10 表示 14 倍频，11 表示 17 倍频	——

118	TMODE[1]	输入		——
119	PLLBYPASS	输入	当为高时，PLL 被旁路；当为低时，系统时钟为 CPU 时钟的 1/4	——
120	CLK	输入	晶振输入	——
121	CLKO	输出	晶振输出	——
122	CLKSYS	输出	系统时钟	——
123	VDDH	——	+3.3V	——
124	VSSH	——	地	——
125	ERRORN	OD	系统错误	低
126	WDOGN	OD	看门狗输出	低
127	RESETOUTN	OD	复位输出	低
128	RESETN	输入	系统复位	低
129	DSUEN	输入	DSU 允许	高
130	DSUBRE	输入	DSU 中断	高
131	VDD	——	+1.2V	——
132	GND	——	地	——
133	DSUTX	输出	DSU 发送	
134	DSURX	输入	DSU 接收	
135	TXD2	输出	UART 发送数据通道 2	
136	RXD2	输入	UART 接收数据通	

			道 2	
137	TXD1	输出	UART 发送数据通道 1	
138	RXD1	输入	UART 发送数据通道 1	
139	VDDH	——	+3.3V	——
140	VSSH	——	地	——
141	CAN_TXD	输出	CAN 发送	
142	CAN_RXD	输入	CAN 接收	
143	ETHERNET_RXDV	输入	以太网接收数据	高
144	ETHERNET_RXERR	输入	以太网接收数据错误	高
145	ETHERNET_RXCOL	输入	以太网碰撞检测	高
146	ETHERNET_RXCRS	输入	以太网接收检测	高
147	VDD	——	+1.2V	——
148	GND	——	地	——
149	ETHERNET_RXCLK	输入	以太网接收时钟	——
150	ETHERNET_RXD[3]	输入	以太网接收数据通道 3	——
151	ETHERNET_RXD[2]	输入	以太网接收数据通道 2	——
152	ETHERNET_RXD[1]	输入	以太网接收数据通道 1	——
153	ETHERNET_RXD[0]	输入	以太网接收数据通道	——

			0	
154	ETHERNET_TXD[3]	输出	以太网发送数据通道 3	——
155	ETHERNET_TXD[2]	输出	以太网发送数据通道 2	——
156	ETHERNET_TXD[1]	输出	以太网发送数据通道 1	——
157	ETHERNET_TXD[0]	输出	以太网发送数据通道 0	——
158	ETHERNET_TXCLK	输入	以太网发送时钟	——
159	ETHERNET_TXEN	输入	以太网发送允许	高
160	ETHERNET_TXERR	输出	以太网发送错误	高
161	ETHERNET_MDC	输出	以太网 MDIO 时钟	——
162	ETHERNET_MDIO	输出	以太网 MDIO 输出数据	——
163	VDD	——	+1.2V	——
164	GND	——	地	——
165	TXA_1553B	输出	1553B A 通道正输出端	高
166	TXAN_1553B	输出	1553B A 通道负输出端	高
167	TXAINH_1553B	输出	1553B A 通道禁止输出使能端	高
168	TXB_1553B	输出	1553B B 通道正输出	高

			端	
169	TXBN_1553B	输出	1553B B 通道负输出端	高
170	TXBINH_1553B	输出	1553B B 通道禁止输出使能端	高
171	VDDH	——	+3.3V	——
172	VSSH	——	地	——
173	CLK_1553B	输入	16MHz 时钟输入	——
174	RXA_1553B	输入	1553B A 通道正输入端	——
175	RXAN_1553B	输入	1553B A 通道负输入端	——
176	RXB_1553B	输入	1553B B 通道正输入端	——
177	RXBN_1553B	输入	1553B B 通道负输入端	——
178	RWEN[3]	输出	存贮器写允许	低
179	RWEN[2]	输出	存贮器写允许	低
180	VDD	——	+1.2V	——
181	GND	——	地	——
182	RWEN[1]	输出	存贮器写允许	低
183	RWEN[0]	输出	存贮器写允许	低
184	IOSN	输出	I/O 片选	低

185	ROMSN[1]	输出	PROM 片选	低
186	ROMSN[0]	输出	PROM 片选	低
187	DATA[0]	双向	外部数据线	高
188	VDDH	——	+3.3V	——
189	VSSH	——	地	——
190	DATA[1]	双向	外部数据线	高
191	DATA[2]	双向	外部数据线	高
192	DATA[3]	双向	外部数据线	高
193	DATA[4]	双向	外部数据线	高
194	DATA[5]	双向	外部数据线	高
195	VDD	——	+1.2V	——
196	GND	——	地	——
197	DATA[6]	双向	外部数据线	高
198	DATA[7]	双向	外部数据线	高
199	DATA[8]	双向	外部数据线	高
200	DATA[9]	双向	外部数据线	高
201	DATA[10]	双向	外部数据线	高
202	VDDH	——	+3.3V	——
203	VSSH	——	地	——
204	DATA[11]	双向	外部数据线	高
205	DATA[12]	双向	外部数据线	高
206	DATA[13]	双向	外部数据线	高

207	DATA[14]	双向	外部数据线	高
208	DATA[15]	双向	外部数据线	高

备注：

1. 表中引脚 111, 引脚 112　N/C 表示此脚只能悬空，外面禁止接其它信号;

6.3 CQFP256 陶瓷封装尺寸

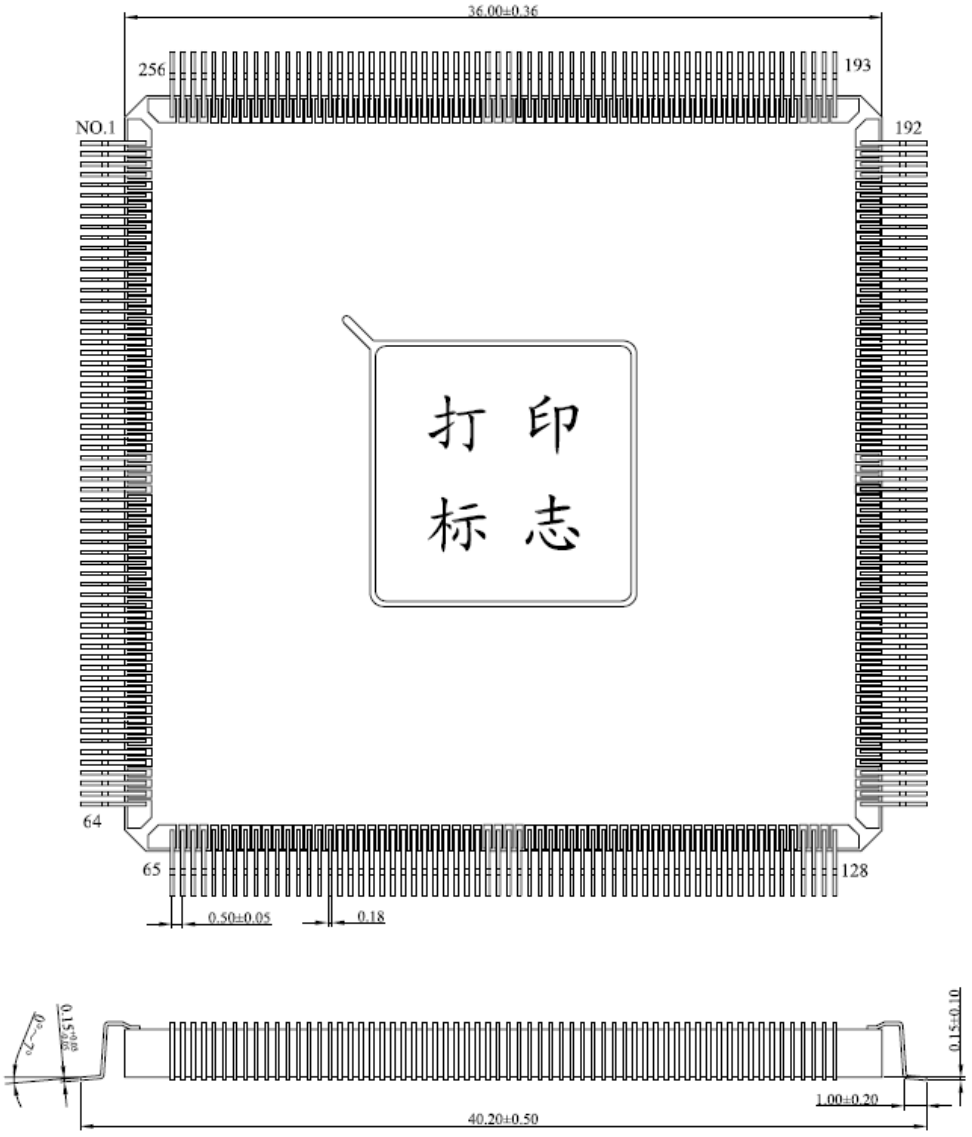


图 6-2 CQFP256 封装

6.4 CQFP256 陶瓷封装信号定义

表 6-2 信号描述

引脚编号	引脚名	类型	功能	有效
1	data[25]	双向	外部数据线	高
2	data[26]	双向	外部数据线	高
3	data[27]	双向	外部数据线	高
4	data[28]	双向	外部数据线	高
5	data[29]	双向	外部数据线	高
6	data[30]	双向	外部数据线	高
7	VDDH	——	+3.3V	——
8	VSSH	——	地	——
9	data[31]	输出	外部地址线	高
10	read	输出	读允许	高
11	oen	输出	输出允许	低
12	ramsn[4]	输出	静态存贮器片选	低
13	ramsn[3]	输出	静态存贮器片选	低
14	VDD	——	+1.2V	——
15	GND	——	地	——
16	ramsn[2]	输出	静态存贮器片选	低
17	ramsn[1]	输出	静态存贮器片选	低
18	ramsn[0]	输出	静态存贮器片选	低
19	brdyn	输入	总线准备好	低

20	bexcn	输入	总线异常	低
21	VDDH	——	+3.3V	——
22	VSSH	——	地	——
23	clksd	输出	动态随机存贮器时钟域 1	——
24	clksd2	输出	动态随机存贮器时钟域 2	——
25	VDD	——	+1.2V	——
26	GND	——	地	——
27	sdcke[1]	输出	动态随机存贮器时钟允许	高
28	sdcke[0]	输出	动态随机存贮器时钟允许	高
29	sdcsn[1]	输出	动态随机存贮器片选	低
30	sdcsn[0]	输出	动态随机存贮器片选	低
31	sdwen	输出	动态随机存贮器写允许	低
32	sdrasn	输出	动态随机存贮器行地址允许	低
33	VDD	——	+1.2V	——
34	GND	——	地	——
35	sdcasn	输出	动态随机存贮器列地址允许	低
36	sddqm[3]	输出	动态随机存贮器数据	低

			掩码	
37	sddqm[2]	输出	动态随机存贮器数据掩码	低
38	sddqm[1]	输出	动态随机存贮器数据掩码	低
39	sddqm[0]	输出	动态随机存贮器数据掩码	低
40	VDDH	——	+3.3V	——
41	VSSH	——	地	——
42	ramoen[4]	输出	静态存贮器输出允许	低
43	ramoen[3]	输出	静态存贮器输出允许	低
44	ramoen[2]	输出	静态存贮器输出允许	低
45	ramoen[1]	输出	静态存贮器输出允许	低
46	ramoen[0]	输出	静态存贮器输出允许	低
47	VDD	——	+1.2V	——
48	GND	——	地	——
49	gpio[0]	双向	通用接口 I/O	——
50	gpio[1]	双向	通用接口 I/O	——
51	gpio[2]	双向	通用接口 I/O	——
52	gpio[3]	双向	通用接口 I/O	——
53	VDDH	——	+3.3V	——
54	VSSH	——	地	——
55	iocsn[2]	输出	IO 片选	低

56	iocsn[3]	输出	IO 片选	低
57	VDD	——	+1.2V	——
58	GND	——	地	——
59	gpio[14]	双向	通用接口 I/O	——
60	gpio[15]	双向	通用接口 I/O	——
61	gpio[16]	双向	通用接口 I/O	——
62	gpio[17]	双向	通用接口 I/O	——
63	gpio[18]	双向	通用接口 I/O	——
64	gpio[19]	双向	通用接口 I/O	——
65	gpio[4]	双向	通用接口 I/O	——
66	gpio[5]	双向	通用接口 I/O	——
67	gpio[6]	双向	通用接口 I/O	——
68	gpio[7]	双向	通用接口 I/O	——
69	VDDH	——	+3.3V	——
70	VSSH	——	地	——
71	NC	——	悬空	——
72	NC	——	悬空	——
73	AVDD	——	+1.2V	——
74	AGND	——	地	——
75	VDD	——	+1.2V	——
76	GND	——	地	——
77	tmod[0]	输入	时钟倍频设置：	——

			当 tmode[1..0] = 00 表示 5 倍频，01 表示 10 倍频，10 表示 14 倍频，11 表示 17 倍频	
78	tmod[1]	输入		——
79	pllbypass	输入	当为高时，PLL 被旁路；当为低时，系统时钟为 CPU 时钟的 1/4	——
80	clk	输入	晶振输入	——
81	clk_xo	输出	晶振输出	——
82	clksys	输出	系统时钟	——
83	VDDH	——	+3.3V	——
84	VSSH	——	地	——
85	errorn	OD	系统错误	低
86	wdog	OD	看门狗输出	低
87	resetoutn	OD	复位输出	低
88	resetn	输入	系统复位	低
89	dsuen	输入	DSU 允许	高
90	dsubre	输入	DSU 中断	高
91	VDD	——	+1.2V	——
92	GND	——	地	——
93	dsutx	输出	DSU 发送	

94	dsurx	输入	DSU 接收	
95	txd2	输出	UART 发送数据通道 2	
96	rxd2	输入	UART 接收数据通道 2	
97	txd1	输出	UART 发送数据通道 1	
98	rxd1	输入	UART 接收数据通道 1	
99	VDDH	——	+3.3V	——
100	VSSH	——	地	——
101	can_txd	输出	CAN 发送	
102	can_rxd	输入	CAN 接收	
103	erx_dv	输入	以太网接收数据	高
104	erx_er	输入	以太网接收数据错误	高
105	erx_col	输入	以太网碰撞检测	高
106	erx_crs	输入	以太网接收检测	高
107	VDD	——	+1.2V	——
108	GND	——	地	——
109	clk_erx	输入	以太网接收时钟	——
110	erxd[3]	输入	以太网接收数据通道 3	——
111	erxd[2]	输入	以太网接收数据通道 2	——

112	erxd[1]	输入	以太网接收数据通道 1	——
113	erxd[0]	输入	以太网接收数据通道 0	——
114	etxd[3]	输出	以太网发送数据通道 3	——
115	etxd[2]	输出	以太网发送数据通道 2	——
116	etxd[1]	输出	以太网发送数据通道 1	——
117	VDDH	——	+3.3V	——
118	VSSH	——	地	——
119	iocsn[4]	双向	通用接口 I/O	——
120	iocsn[5]	双向	通用接口 I/O	——
121	VDD	——	+1.2V	——
122	GND	——	地	——
123	gpio[20]	双向	通用接口 I/O	
124	gpio[21]	双向	通用接口 I/O	
125	gpio[22]	双向	通用接口 I/O	
126	gpio[23]	双向	通用接口 I/O	
127	gpio[24]	双向	通用接口 I/O	——
128	gpio[25]	双向	通用接口 I/O	——
129	etxd[0]	输出	以太网发送数据通道 0	——

130	clk_etx	输入	以太网发送时钟	——
131	etx_en	输入	以太网发送允许	高
132	etx_er	输出	以太网发送错误	高
133	emdc	输出	以太网 MDIO 时钟	——
134	emdio	输出	以太网 MDIO 输出数据	——
135	VDD	——	+1.2V	——
136	GND	——	地	——
137	txa_1553b	输出	1553B A 通道正输出端	高
138	txan_1553b	输出	1553B A 通道负输出端	高
139	txainh_1553b	输出	1553B A 通道禁止输出使能端	高
140	txb_1553b	输出	1553B B 通道正输出端	高
141	txbn_1553b	输出	1553B B 通道负输出端	高
142	txbinh_1553b	输出	1553B B 通道禁止输出使能端	高
143	VDDH	——	+3.3V	——
144	VSSH	——	地	——
145	clk_1553b	输入	16MHz 时钟输入	——

146	rxa_1553b	输入	1553B A 通道正输入端	——
147	rxan_1553b	输入	1553B A 通道负输入端	——
148	rxb_1553b	输入	1553B B 通道正输入端	——
149	rxbn_1553b	输入	1553B B 通道负输入端	——
150	rwen[3]	输出	存贮器写允许	低
151	rwen[2]	输出	存贮器写允许	低
152	VDD	——	+1.2V	——
153	GND	——	地	——
154	rwen[1]	输出	存贮器写允许	低
155	rwen[0]	输出	存贮器写允许	低
156	iosn	输出	I/O 片选	低
157	romsn[1]	输出	PROM 片选	低
158	romsn[0]	输出	PROM 片选	低
159	data[0]	双向	外部数据线	高
160	VDDH	——	+3.3V	——
161	VSSH	——	地	——
162	data[1]	双向	外部数据线	高
163	data[2]	双向	外部数据线	高
164	data[3]	双向	外部数据线	高

165	data[4]	双向	外部数据线	高
166	data[5]	双向	外部数据线	高
167	VDD	——	+1.2V	——
168	GND	——	地	——
169	data[6]	双向	外部数据线	高
170	data[7]	双向	外部数据线	高
171	data[8]	双向	外部数据线	高
172	data[9]	双向	外部数据线	高
173	data[10]	双向	外部数据线	高
174	VDDH	——	+3.3V	——
175	VSSH	——	地	——
176	data[11]	双向	外部数据线	高
177	data[12]	双向	外部数据线	高
178	data[13]	双向	外部数据线	高
179	data[14]	双向	外部数据线	高
180	data[15]	双向	外部数据线	高
181	VDD	——	+1.2V	——
182	GND	——	地	——
183	iocsn[6]	输出	I/O 片选	低
184	iocsn[7]	输出	I/O 片选	低
185	VDDH	——	+3.3V	——
186	VSSH	——	地	——

187	gpio[26]	双向	通用接口 I/O	——
188	gpio[27]	双向	通用接口 I/O	——
189	gpio[28]	双向	通用接口 I/O	——
190	gpio[29]	双向	通用接口 I/O	——
191	gpio[30]	双向	通用接口 I/O	——
192	gpio[31]	双向	通用接口 I/O	——
193	address[0]	输出	外部地址线	高
194	address[1]	输出	外部地址线	高
195	address[2]	输出	外部地址线	高
196	address[3]	输出	外部地址线	高
197	VDDH	——	+3.3V	——
198	VSSH	——	地	——
199	address[4]	输出	外部地址线	高
200	address[5]	输出	外部地址线	高
201	address[6]	输出	外部地址线	高
202	address[7]	输出	外部地址线	高
203	VDD	——	+1.2V	——
204	GND	——	地	——
205	address[8]	输出	外部地址线	高
206	address[9]	输出	外部地址线	高
207	address[10]	输出	外部地址线	高
208	address[11]	输出	外部地址线	高

209	VDDH	——	+3.3V	——
210	VSSH	——	地	——
211	address[12]	输出	外部地址线	高
212	address[13]	输出	外部地址线	高
213	address[14]	输出	外部地址线	高
214	address[15]	输出	外部地址线	高
215	address[16]	输出	外部地址线	高
216	address[17]	输出	外部地址线	高
217	VDD	——	+1.2V	——
218	GND	——	地	——
219	address[18]	输出	外部地址线	高
220	address[19]	输出	外部地址线	高
221	address[20]	输出	外部地址线	高
222	address[21]	输出	外部地址线	高
223	address[22]	输出	外部地址线	高
224	address[23]	输出	外部地址线	高
225	VDDH	——	+3.3V	——
226	VSSH	——	地	——
227	address[24]	输出	外部地址线	高
228	address[25]	输出	外部地址线	高
229	address[26]	输出	外部地址线	高
230	address[27]	输出	外部地址线	高

231	writen	输出	写允许	低
232	data[16]	双向	存贮器 数据	高
233	VDD	——	+1.2V	——
234	GND	——	地	——
235	data[17]	双向	外部数据线	高
236	data[18]	双向	外部数据线	高
237	data[19]	双向	外部数据线	高
238	data[20]	双向	外部数据线	高
239	data[21]	双向	外部数据线	高
240	data[22]	双向	外部数据线	高
241	VDD	——	+1.2V	——
242	GND	——	地	——
243	data[23]	双向	外部数据线	高
244	data[24]	双向	外部数据线	高
245	VDDH	——	+3.3V	——
246	VSSH	——	地	——
247	iocsn[0]	输出	I/O 片选	低
248	iocsn[1]	输出	I/O 片选	低
249	VDD	——	+1.2V	——
250	GND	——	地	——
251	gpio[8]	双向	通用接口 I/O	——
252	gpio[9]	双向	通用接口 I/O	——

253	gpio[10]	双向	通用接口 I/O	——
254	gpio[11]	双向	通用接口 I/O	——
255	gpio[12]	双向	通用接口 I/O	——
256	gpio[13]	双向	通用接口 I/O	——

7 应用例子

为帮助用户尽快熟悉并掌握 S698P4-II 芯片的使用，配套芯片还有相应的开发板。下面介绍了使用 S698P4-II 芯片的常用例子。如用户涉及到软件调试，需要先安装 ORION5.0 软件。

7.1 S698P4-II 如何调试

打开Cygwin环境，进入程序的工作目录。如图5.1-1所示：

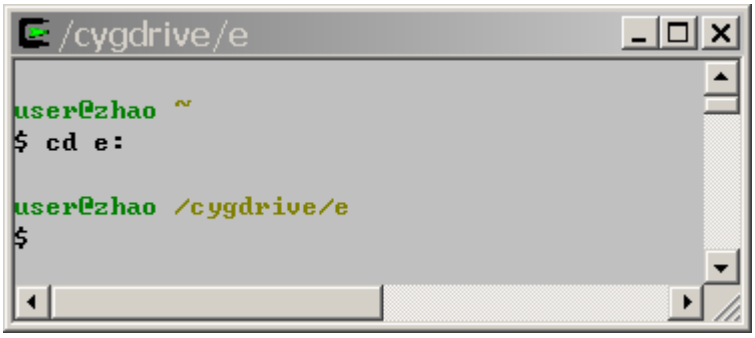


图 7-1

- **通过串口调试：**

1）编写好程序，使用 sparc-elf-gcc 编译器，对源程序进行编译

```
$ sparc-elf-gcc lcd.c
```

默认生产 a.out 文件，接好 DSU 串口线。接线示意图如图 5-2：

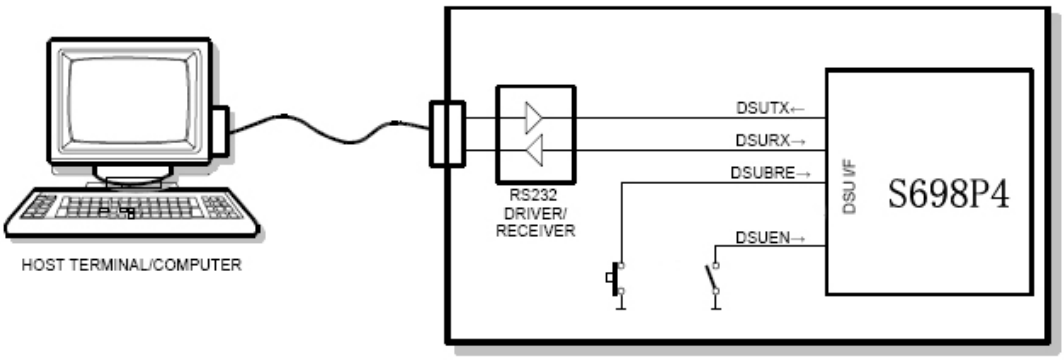


图 7-2　通过串口 DSU 调试示意图

2）在命令行中输入：V8mon –i –u，如下所示：

```
user@zhao ~
$ v8mon.exe -i -u
```

```
V8MON ORBIT debug monitor V2.0
Copyright (C) 2004,2008 Orbita    - all rights reserved.
For latest updates, go to http：//www.Myorbita.net/
Comments or bug-reports to support@Myorbita.net

using port /dev/ttyS0 @ 115200 baud
GRLIB build version：  2314
initialising .....
detected frequency：  200 MHz
Component                                Vendor
ORBIT SPARC V8 Processor                 Orbita
ORBIT SPARC V8 Processor                 Orbita
ORBIT SPARC V8 Processor                 Orbita
ORBIT SPARC V8 Processor                 Orbita
AHB Debug UART                             Orbita
GR Ethernet MAC                            Orbita
ORBIT Memory Controller                    Orbita
AHB/APB Bridge                             Orbita
ORBIT Debug Support Unit                   Orbita
OC CAN controller                          Orbita
Generic APB UART                           Orbita
Multi-processor Interrupt Ctrl             Orbita
Modular Timer Unit                         Orbita
Generic APB UART                           Orbita
General purpose I/O port                   Orbita
Unknown device                             Orbita

Use command 'info sys' to print a detailed report of attached cores

V8mon>
```

3）输入“lo +程序名”下载程序

```
grlib> lo a.out
section：  .text at 0x40000000, size 41872 bytes
section：  .data at 0x4000a390, size 70044 bytes
total size：  111916 bytes (88.2 kbit/s)
read 207 symbols
entry point：  0x40000000
v8mon>
```

4）输入“run”命令，程序即可运行，“LCD TEST”为

```
V8mon> run
LCD TEST
```

- **通过以太网调试**

接好以太网线。接线示意图如图5-3

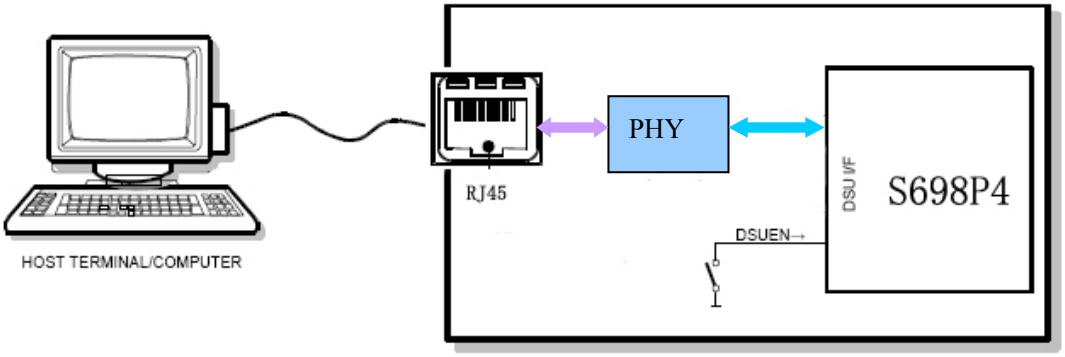


图7-3 通过以太网调试接线示意图

1）在命令行中输入“V8mon.exe -i -u -eth -freq 100”命令，进入后，通过“info sys”命令可查看S698P4-II 内部信息。

```
user@zhao /cygdrive/e
$ V8mon.exe -i -u -eth -freq 100

 V8MON LEON debug monitor v1.1.29

 Copyright (C) 2004,2008 ORBITA - all rights reserved.
 For latest updates, go to http: //www.myorbita.com/
 Comments or bug-reports to support@myorbita.com

 ethernet startup.
 S698P4-II  build version:  2314

 initialising ................

 Component                            Vendor
 ORBITA SPARC V8 Processor             ORBITA
 ORBITA SPARC V8 Processor             ORBITA
 ORBITA SPARC V8 Processor             ORBITA
 ORBITA SPARC V8 Processor             ORBITA
 AHB Debug UART                       ORBITA
```

```
 GR Ethernet MAC                         ORBITA
 ORBITA Memory Controller                 ORBITA
 AHB/APB Bridge                          ORBITA
 ORBITA Debug Support Unit                ORBITA
 OC CAN controller                       ORBITA
 Generic APB UART                        ORBITA
 Multi-processor Interrupt Ctrl          ORBITA
 Modular Timer Unit                      ORBITA
 Generic APB UART                        ORBITA
 General purpose I/O port                ORBITA
 Unknown device                          ORBITA

 Use command 'info sys' to print a detailed report of attached cores

v8mon> info sys
00.01: 003   ORBITA  ORBITA SPARC V8 Processor (ver 0x0)
             ahb master 0
01.01: 003   ORBITA  ORBITA SPARC V8 Processor (ver 0x0)
             ahb master 1
02.01: 003   ORBITA  ORBITA SPARC V8 Processor (ver 0x0)
             ahb master 2
03.01: 003   ORBITA  ORBITA SPARC V8 Processor (ver 0x0)
             ahb master 3
04.01: 007   ORBITA  AHB Debug UART (ver 0x0)
             ahb master 4
             apb:  80000700 - 80000800
             baud rate 115200, ahb frequency 20.00
05.01: 01d   ORBITA  Ethernet MAC (ver 0x0)
             ahb master 5, irq 12
             apb:  80000f00 - 80001000
             edcl ip 192.168.0.51, buffer 2 kbyte
00.04: 00f   ORBITA Memory Controller (ver 0x1)
             ahb:  00000000 - 20000000
             ahb:  20000000 - 40000000
             ahb:  40000000 - 80000000
             apb:  80000000 - 80000100
             32-bit prom @ 0x00000000
             32-bit static ram:  1 * 2048 kbyte @ 0x40000000
             32-bit sdram:  2 * 32 Mbyte @ 0x60000000, col 9, cas 2, ref 7.7 us
01.01: 006   ORBITA  AHB/APB Bridge (ver 0x0)
```

```
              ahb:  80000000 - 80100000
02.01: 004    ORBITA  ORBITA Debug Support Unit (ver 0x1)
              ahb:  90000000 - a0000000
              AHB trace 1 lines, stack pointer 0x401ffff0
              CPU#0 win 8, hwbp 2, itrace 64, V8 mul/div, lddel 1, OBT-FPU
                    icache 2 * 4 kbyte, 32 byte/line lru
                    dcache 2 * 4 kbyte, 32 byte/line lru
              CPU#1 win 8, hwbp 2, itrace 64, V8 mul/div, lddel 1, OBT-FPU
                    icache 2 * 4 kbyte, 32 byte/line lru
                    dcache 2 * 4 kbyte, 32 byte/line lru
              CPU#2 win 8, hwbp 2, itrace 64, V8 mul/div, lddel 1, OBT-FPU
                    icache 2 * 4 kbyte, 32 byte/line lru
                    dcache 2 * 4 kbyte, 32 byte/line lru
              CPU#3 win 8, hwbp 2, itrace 64, V8 mul/div, lddel 1, OBT-FPU
                    icache 2 * 4 kbyte, 32 byte/line lru
                    dcache 2 * 4 kbyte, 32 byte/line lru
06.01: 019    ORBITA  CAN controller (ver 0x0)
              irq 13
              ahb:  fffc0000 - fffc0100
01.01: 00c    ORBITA  Generic APB UART (ver 0x1)
              irq 2
              apb:  80000100 - 80000200
              baud rate 38461, DSU mode (loop-back)
02.01: 00d    ORBITA  Multi-processor Interrupt Ctrl (ver 0x3)
              apb:  80000200 - 80000300
03.01: 011    ORBITA  Modular Timer Unit (ver 0x0)
              irq 8
              apb:  80000300 - 80000400
              8-bit scaler, 2 * 32-bit timers, divisor 20
09.01: 00c    ORBITA  Generic APB UART (ver 0x1)
              irq 3
              apb:  80000900 - 80000a00
              baud rate 38461
0b.01: 01a    ORBITA  General purpose I/O port (ver 0x0)
              apb:  80000b00 - 80000c00
0e.01: 072    ORBITA  Unknown device (ver 0x0)
              irq 10
              apb:  80008000 - 80010000
v8mon>
```

7.2 eCos

本程序进行 eCos 操作系统的核调度优先级抢占测试和时间片轮转测试，测试在 eCos 系统调度下，每一个任务在每个 CPU 上各运行了多少个时间片，4 个 CPU 是否负载均衡，。

系统首先创建并开始一个最高优先级的主任务，启动调度。主任务控制进行测试的任务数，调用负载测试函数。任务数从 5 个开始递增，一直到 24 个，这样为一次循环，不停的重复循环测试。

负载测试函数：初始化全局数组 slicerun[24][4]（任务在某一个 CPU 上的运行次数）；修改主任务的优先级为 2；创建指定数量的、优先级为 10 的相同的子任务；生成任务后，主任务任务进入休眠，系统开始进行任务调度运行；主任务唤醒后，抢占 CPU，挂起其它的任务，停止任务调度；主任务计算各个 CPU 的负载情况，计算每个任务在各个 CPU 上的执行时间片情况；后输出结果，删除全部子任务，函数返回。

子任务：获取当前 CPU 标号，slicerun[当前任务号][当前 CPU 号]值循环加 1。

程序编译 Makefile 文件如下：

```
    CC=sparc-elf-gcc
    CFLAGS= -I../ecos_install/include -I../install/include  -g
    LDFLAGS = -L../ecos_install/lib -L../install/lib  -Ttarget.ld -g -nostdlib
-Wl,--gc-sections -Wl,-static -Wl,--Map -Wl,
    PROGS = timeslice.exe
    all:  $(PROGS)
    timeslice.exe:  timeslice.c
    $(CC)  $(CFLAGS) -O3 $(LDFLAGS)timeslice.map timeslice.c -o timeslice.exe
    clean:

    -rm $(PROGS) *.map
```

程序流程图见图 7-4：

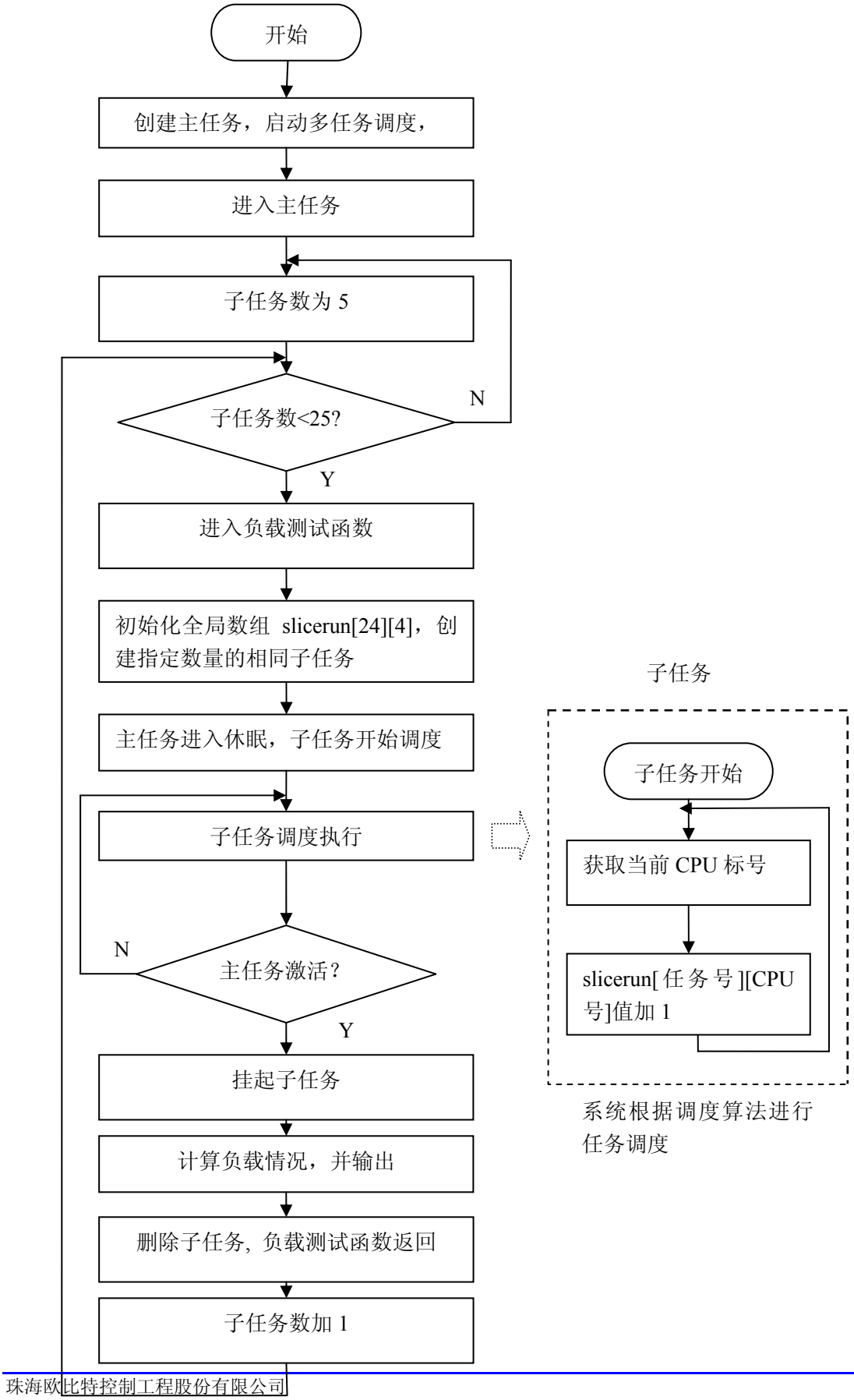

开始
创建主任务，启动多任务调度，
进入主任务
子任务数为 5
子任务数<25?
N
Y
进入负载测试函数
初始化全局数组 slicerun[24][4]，创建指定数量的相同子任务
主任务进入休眠，子任务开始调度
子任务调度执行
主任务激活？
N
Y
挂起子任务
计算负载情况，并输出
删除子任务，负载测试函数返回
子任务数加 1
子任务
子任务开始
获取当前 CPU 标号
slicerun[任务号][CPU 号]值加 1
系统根据调度算法进行任务调度

图7-4 FPU示例程序流程图

5.3 FPU

S698P4-II 有 4 个 CPU，每个 CPU 都带有一个 FPU。下面用一个示例程序做例子进行说明。本程序同时对 S698P4-II 的 FPU 运算能力进行测试，每个 FPU 分别单独执行 N*N 次加减乘除的浮点运算，利用浮点运算次数和所消耗的时间计算出每个 FPU 的每秒浮点数。测试基于 eCos 操作系统。程序一开始开启一个主任务，在主任务中初始化 1 个条件标志和 2 个用于同步的互斥变量（一个保护打印输出，一个保护任务结束变量）；跟着初始化两个大小为 N 的数组 a[N]，b[N] ，N 的值可以调节，数组值为随机的浮点数；创建并开始调度 4 个优先级更高、相同的任务 threada/ threadb/ threadc/ threadd，每个任务分别单独执行 N*N 次加减乘除的浮点运算，计算每种运算的执行时间，利用浮点运算次数和所消耗的时间计算出四核系统的每秒浮点数，并打印结果，任务结束变量加 1，如果任务结束变量为 4 设置条件标志，任务结束；如果主任务检测到 4 个其它任务结束的标志，打印综合结果。程序编译 Makefile 文件如下：

```
    CC=sparc-elf-gcc
    CFLAGS= -I../ecos_install/include -I../install/include  -g
    LDFLAGS = -L../ecos_install/lib -L../install/lib  -Ttarget.ld -g -nostdlib
-Wl,--gc-sections -Wl,-static -Wl,--Map -Wl,

    PROGS =  flops-parallel.exe
    all:  $(PROGS)
    flops-parallel.exe:  flops-parallel.c
     $(CC)  $(CFLAGS) -O2 $(LDFLAGS)flops-parallel.map flops-parallel.c -o flops-parallel.exe
     rm *.map
    clean:

     -rm $(PROGS)
```

程序流程图见图 7-5：

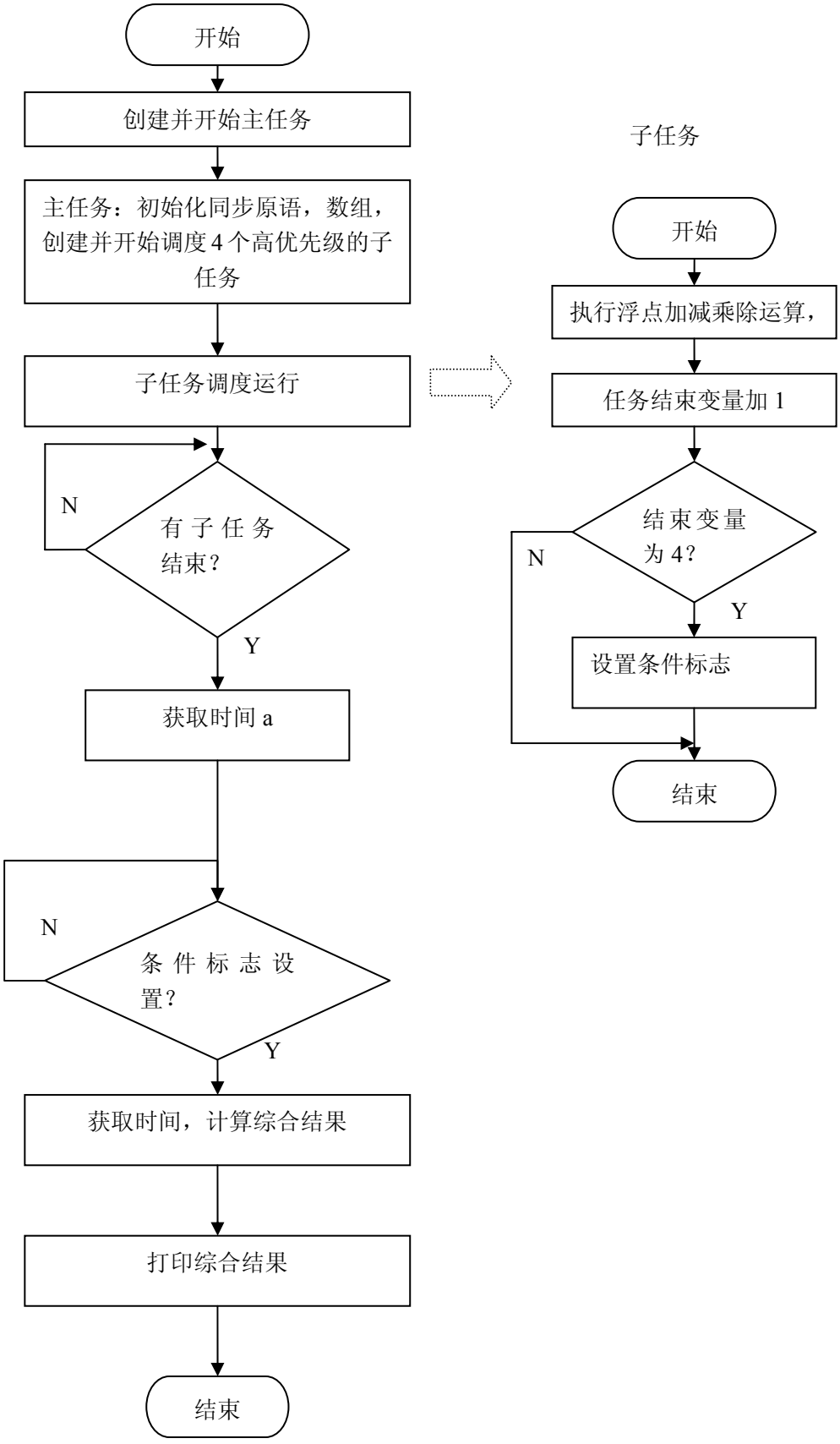

开始
创建并开始主任务
主任务：初始化同步原语，数组，创建并开始调度 4 个高优先级的子任务
子任务调度运行
N
有子任务结束？
Y
获取时间 a
N
条件标志设置？
Y
获取时间，计算综合结果
打印综合结果
结束
子任务
开始
执行浮点加减乘除运算，
任务结束变量加 1
结束变量为 4？
N
Y
设置条件标志
结束

图7-5 FPU示例程序流程图

7.4 数据一致性

硬件支持 Cache 一致性的原理：读取存储区共享数据到 cache 后，如果自己 cache 数据对应的存储区共享数据发生改变时，会自动使 cache 对应的数据无效，cpu 对这个数据进行操作时，从存储区读取数据，并且更新 cache 的数据。如果自己 cache 数据对应的存储区共享数据没有改变，cpu 对这个数据进行操作，直接从 cache 读取数据。

软件支持 Cache 一致性的原理：对于共享数据的操作，应该首先向系统请求锁定这些数据，然后进行操作。当操作完成后，更新数据并解锁。这样，即使是这些数据中间是在寄存器中操作，也可以保证最终结果是正确的。下面用一个示例程序做例子进行说明。

程序假设共享数据 1000 个字节的情况进行测试。程序基于 eCos 操作系统。先创建主任务，声明共享数据、互斥量，螺旋锁。在程序入口初始化共享数据、螺旋锁、互斥量，初始化数组，创建 4 个子线程任务，4 个子任务对应 4 个 cpu。每个 cpu 都不断做循环，根据标志判断当前状态是读数据还是修改数据。如果是读数据：判断共享数据是否等于修改后的值，不等就报错；如果是修改数据：打印,修改共享数据，置本 cpu 为读，置下一个 cpu 为修改数据。要对共享数据进行操作，就先要获得指定螺旋锁。不断循环的修改读写数据。

下面分别列出主程序流程图和子任务流程图：

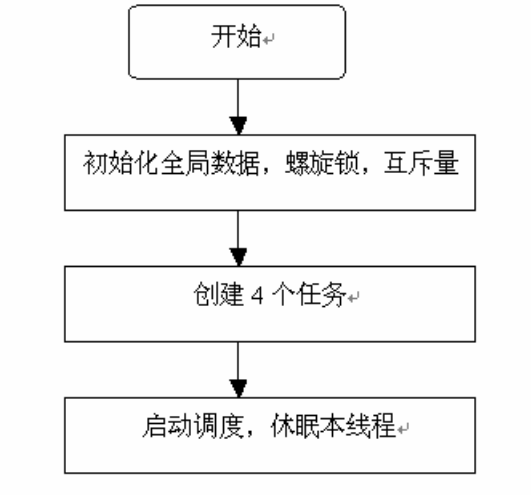


图 7-6 主任务流程图

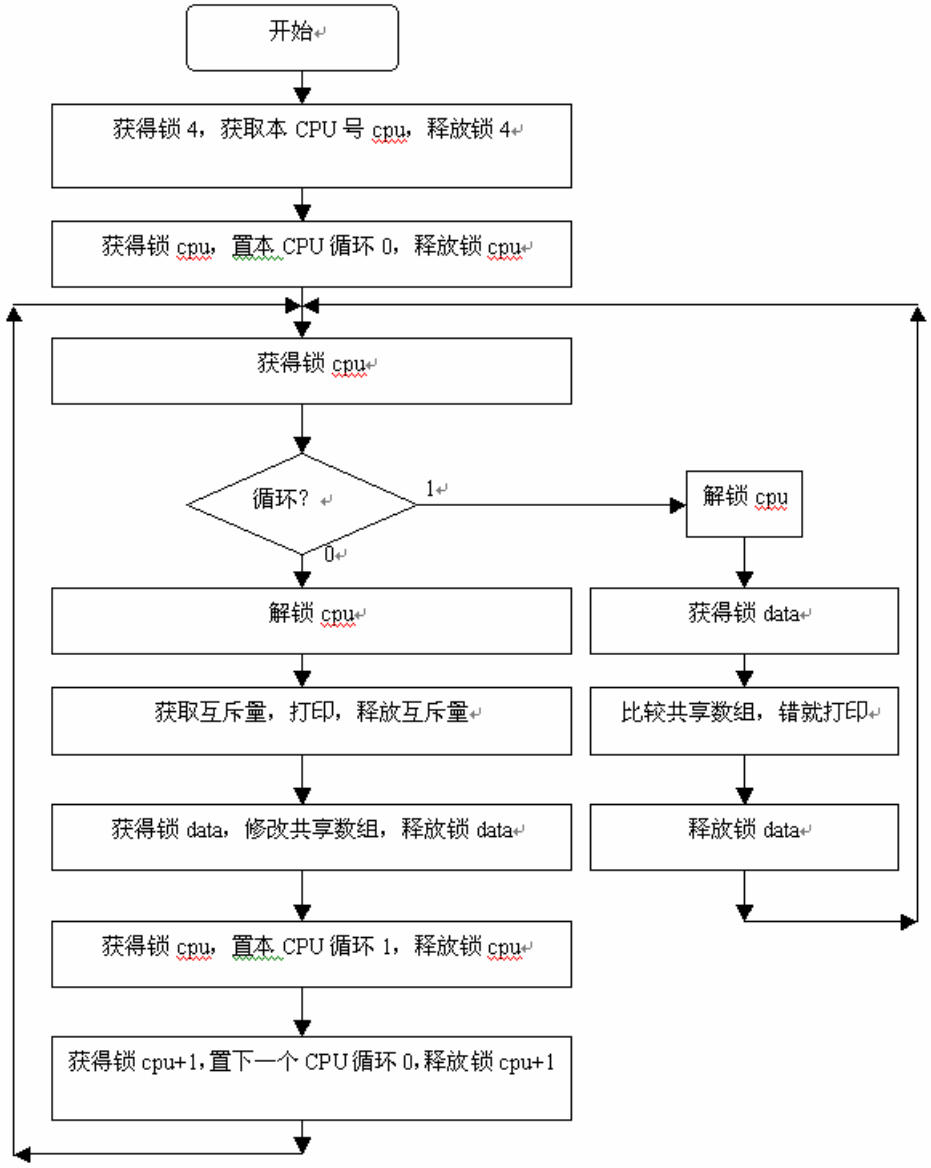


图 7-7 子任务流程图

7. 5 MEMORY 接口

MEMORY总线提供PROM、I/O、SRAM、SDRAM的专用接口，有两个PROM

片选、1个I/O片选、5个SRAM片选、两个SDRAM banks选择。图5-8 展示了几种设备的连接方式。这里需要注意的是：PROM、SRAM、SDRAM数据总线宽度都只支持32位宽方式，只有I/O数据总线宽度8、16、32位宽可选。

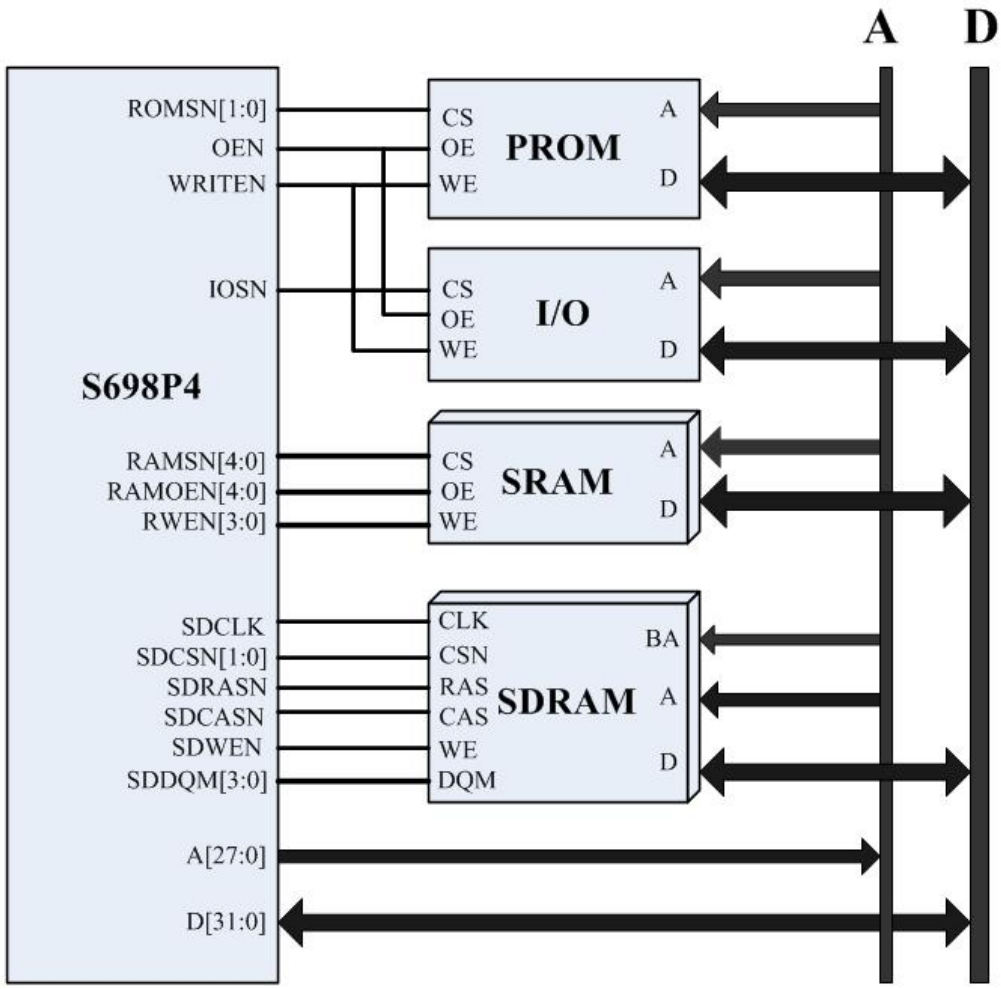


图7-8 MEMORY接口示意图

7.7 以太网

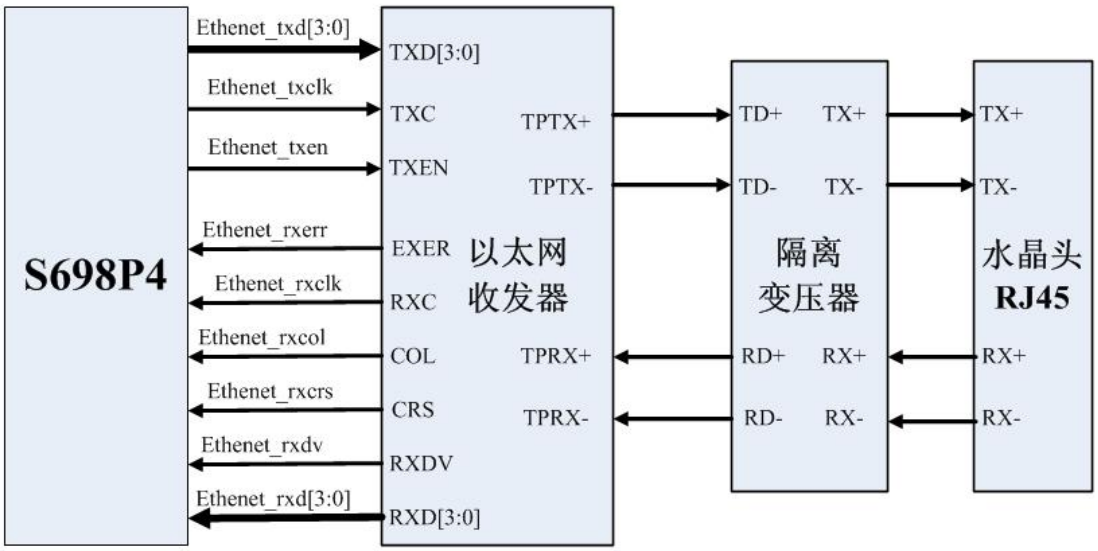


图 7-11 以太网接口原理图

以太网接口原理图可以实现 S698P4-II 以太网控制器功能，支持 10M/100Mbps，同时也可通过以太网接口来调试 S698P4-II 芯片。

以太网 UDP 数据传输示例程序为 eCos 操作系统程序，在对程序编译前，请安装 eCos 相关环境，编译 Makefile 文件如下：

```
CC=sparc-elf-gcc
CFLAGS= -I../ecos_install/include -I../install/include -O2 -g
LDFLAGS = -L../ecos_install/lib -L../install/lib -Ttarget.ld -g -nostdlib -Wl,--gc-sections -Wl,-static -Wl,--Map -Wl,
PROGS = udp_send_test.exe
all： $(PROGS)
udp_send_test.exe： udp_send_test.c
    $(CC) $(CFLAGS) $(LDFLAGS)udp_send_test.map udp_send_test.c -o udp_send_test.exe
    rm *.map
clean：
    -rm *.exe
```

程序流程图如图 7-12

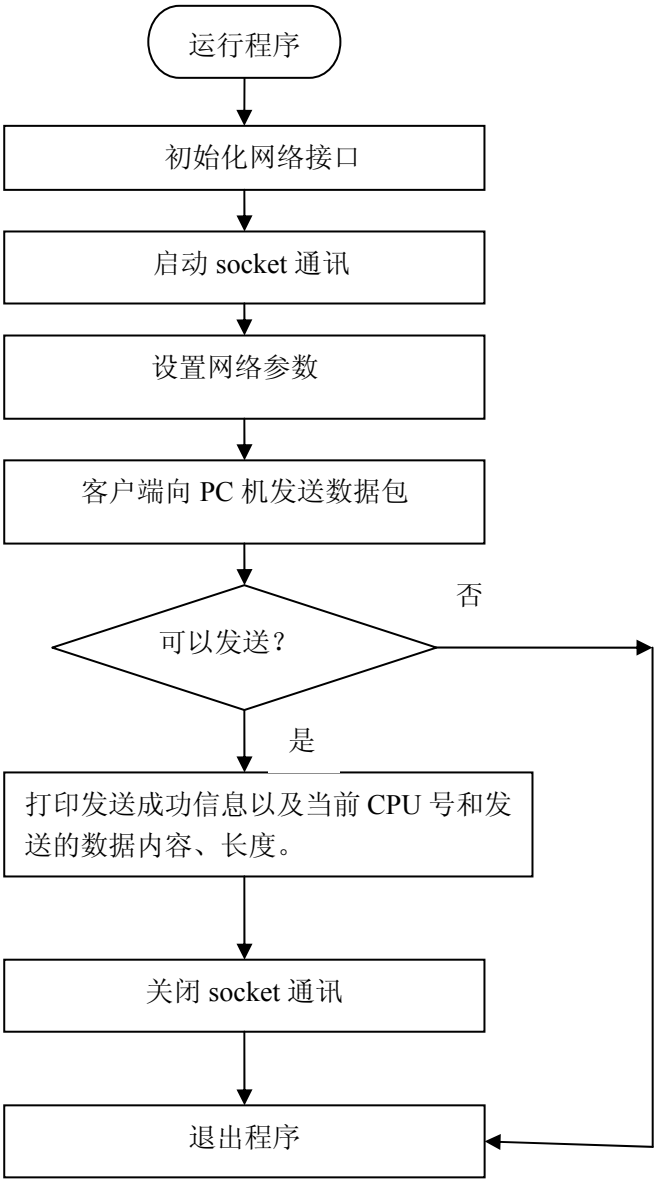


图 7-12 以太网 UDP 数据传输流程图

本程序实现了客户端向服务器发送数据功能。程序首先创建客户端发送数据线程，然后定义服务器端口和 IP 地址，初始化网络，启动 socket 通讯，客户端向 PC 机发送数据包，PC 端打开网络调试助手，设置 IP 地址与 UDP 连接方式，接收到客户端发出的数据后将数据显示出来。PC 端接收到的数据和客户端发送出的数据一致，说明程序运行正确。

7.8 CAN 总线

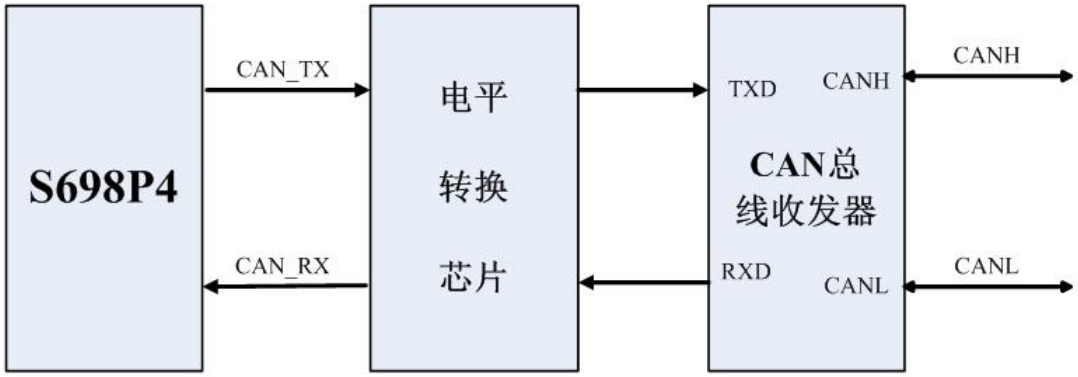


图 7-13　CAN 总线接口原理图

因为 S698P4-II 芯片输出电平为 LVTTL 电平，而 CAN 收发器芯片为 TTL 电平，在 S698P4-II 信号与 CAN 收发器之间需要添加一个电平转换芯片，用来做电平匹配。

以下用测试 CAN 模块的 PELI 模式的自环收发功能做为示例做说明。当采用自环功能时，如果对外发送数据，跟着本身 CAN 模块就会接收到数据，如果接收的数据跟发送的数据相同，就说明该功能正常工作；如果没接收到数据或者数据不相同，都说明测试失败。

程序一开始先屏蔽中断；跟着初始化 CAN 模式：设置 PELI 模式、设置分频系数 7、设置总线速率 250K/BPS、设置自环模式；初始化完成之后，设置中断方式-收发都产生中断，挂中断处理程序，中断使能；如果发送数据区为空，就可以发送数据了。如果发送成功，则会产生一个发送中断；如果产生接收中断，读取数据，如果数据跟发送的数据相同，就说明成功。否则失败。

程序流程图：

主流程　　中断处理流程

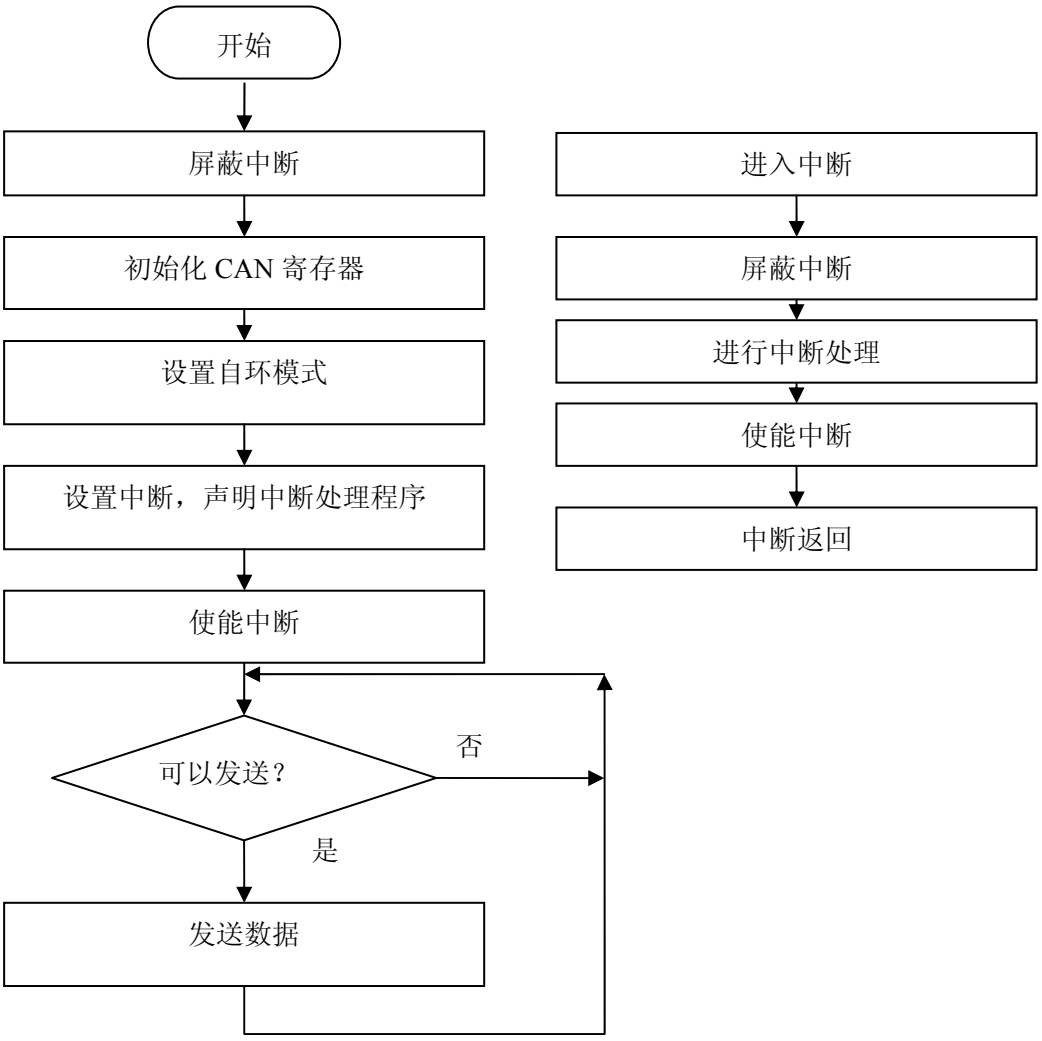


图 7-16 CAN 总线示例程序流程图

7.9 DSU 调试接口

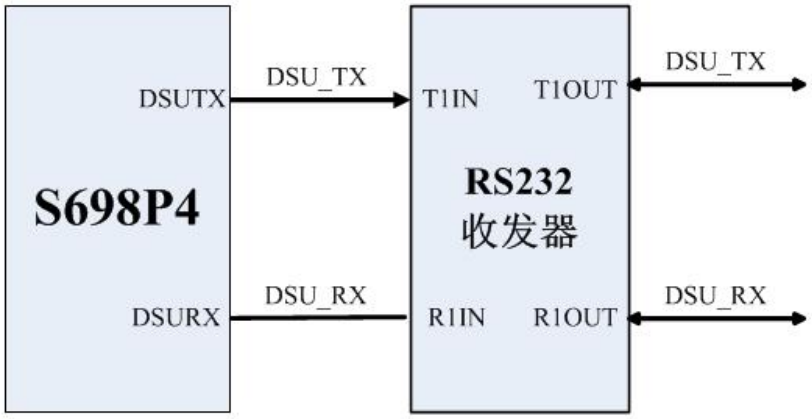


图 5-15 DSU 调试接口示意图

在 DSU 调试中串口调试模式比较方便，电路比较简单，图 5-15 同时也举例了 DSU 调试接口示意图，用户在搭建系统时可参看图 5-15 进行设计，因为 DSU 调试口电路简单，可优先选用 DSU 串口进行调试。

8 软件支持

S698P4-II 相应的配套软件有：操作系统 eCos，集成开发环境 orion5.0。

8.1 操作系统 eCos

8.1.1 eCos 简介

eCos(Embedded Configurable Operating System，嵌入式可配置操作系统)是一种针对 16 位、32 位和 64 位处理器的可移植嵌入式实时操作系统。由于 eCos 源代码是公开的，因而有越来越多的设计人员开始关注 eCos 操作系统。

eCos 最大的特点是配置灵活，提供了大范围的可选项供开发人员配置操作系统，以最好地匹配应用的需求，这样就变成了调整操作系统来适应应用。典型的配置选项包括调度器的类型和任务优先级别数目等。它的另一个优点是使用多任务抢占机制，具有最小的中断延迟，支持嵌入式系统所需的所有同步原语(互斥、条件变量、计数型信号量、邮箱和事件标志)，并拥有灵活的调度策略和中断处理机制，因而具有良好的实时性。

eCos 采用多级队列调度方式支持 SMP，该方式采用优先级抢占式，同级时间片调度方法进行多线程的调度，每一个 CPU 在当前时间均可运行一个线程，从而达到并行计算的目的。

eCos 支持 SMP 目标处理器限制：

- 最大支持处理器个数为 8，典型是 2 或 4；
- 硬件必须提供测试并设置或者比较并交换指令同步机制。eCos 内核使用这些硬件指令实现螺旋锁；
- 所有的 CPU 共享系统 cache，或硬件维护 cache 的一致；
- 所有的 CPU 共享的内存对所有的处理器的地址是一致的；
- 每个 CPU 都可以访问任何外设；
- 中断控制器必须将中断信号传送给指定的 CPU；
- 允许一个 CPU 上的事件导致另一个 CPU 的重新调度（需要允许系统中一个 CPU 中断另一个 CPU 的机制）；
- 某个 CPU 上运行的软件必须能识别其所运行的 CPU；

目前，S698P4-II 就是其中一款满足 eCos 对 SMP 的硬件限制条件的处理器：

- S698P4-II 目前支持 4 核；
- S698P4-II 内有 swapa 指令，可以提供测试并设置同步机制；
- S698P4-II 用总线侦听技术(snooping)维护了 4 个 CPU 的 data cache 的一致性；
- S698P4-II 所有的 CPU 共享系统 sram 和 sdram，其编址是一致的，使编程容易；
- S698P4-II 中所有外设的寄存器，所有 CPU 都能访问到；所有外设的中断，都能引起所有 CPU 的中断处理，并且可以通过设置各 CPU 的中断屏蔽寄存器指明哪个外设的中断由哪个 CPU 处理；
- S698P4-II 各个 CPU 之间通过多核中断控制器(MP IRQCTRL)的中断来通信，允许将中断传递到系统中特定的 CPU，该 CPU 执行中断处理函数，根据消息的内容执行优先级抢占重新调度或者时间片轮转调度；
- S698P4-II 各个 CPU 均有一个称为%asr17 的寄存器，其 31-28 位指明当运运行的是哪个处理器。

8.1.2 eCos 的特点

eCos 的主要特点有：

- 支持对称多处理器（SMP）系统；
- 实时性强，具有极小的中断延迟；
- 内核可配置，提供图形化的配置工具；
- 内核简洁，最小版本仅几百个字节，一个完整的网路应用，其二进制的代码仅 100K 字节左右；
- 提供 Linux 兼容的 API，可以轻松实现 linux 应用移植。

8.1.3 eCos 的体系结构

eCos 被设计成一个由几个主要的软件组件组成的可配置的体系结构。这样做的基本目的就是利用这些可重用的软件组件来开发完整的嵌入式系统；同时，这也使得用户可以根据自己应用的特定需求来设置组件中的每个配置选项，或者完全去掉一些不相关的组件。这样，就可创建一个最适合系统应用需求的 eCos 影

像，软件中只包含需要的组件，因此大小是最精简的。实际上，有些实时操作系统并没有提供这样的配置功能，不关实际应用是否需要，开发的软件几乎包含了操作系统中的所用功能；于此不同的是，基于 eCos 开发的软件运行适度非常快，因为不需要执行那些与应用无关的额外代码。

实时嵌入操作系统包含一些标准的功能，例如中断和异常处理，线程同步机制，调度器和驱动程序等，这些也构成了 eCos 的核心组件。具体包括：

- 硬件抽象层(HAL)：用来向上层提供统一的硬件视图，屏蔽硬件的差异；
- 内核： 包括异常处理，线程同步支持，调度器，定时器，计数器；
- ISO C 和数学库：标准 C 函数库；
- 设备驱动程序：标准串口，以太网卡，Flash ROM 等
- GDB 支持：使得目标机上可以和主机通讯进行交叉调试

图 8-1 说明了如何把 eCos 的核心组件和其他一些可先选组件组合到一起，实现一个特定应用所需要的功能。

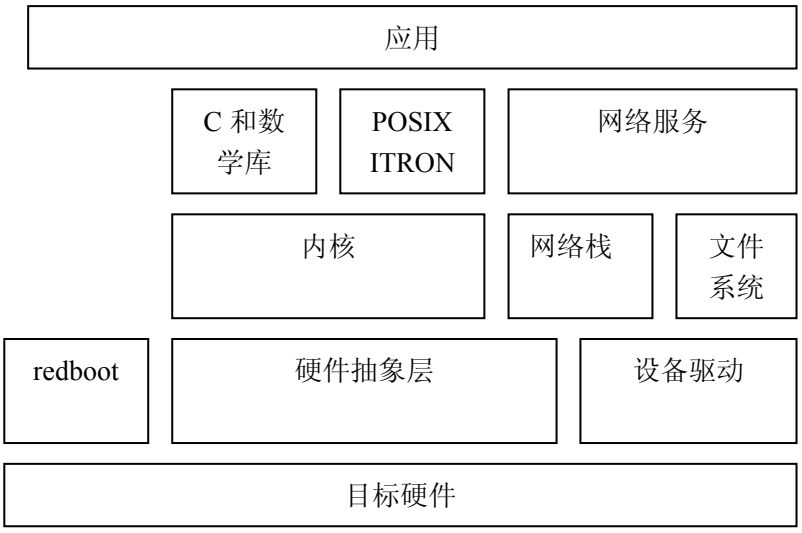


图 6-1 eCos 软件层次结构

可配置性是 eCos 的一个重要特征，因此在 eCos 中也提供了相应的工具来管理组件中从多复杂的配置选项；也可以根据需要利用这个工具来添加或删除组件，最后再用它把 eCos 库中的组件和应用程序链接到一起得到最终的产品。

8.1.4 eCos 的内核结构

eCos 的核心就是内核，内核提供了一个实时操作系统所期望的标准功能，如中断和异常处理、调度、线程和同步等，在 eCos 中，这些构成内核的标准功能组件是完全可配置的，以满足特定的需求。

内核是 eCos 的一个关键包。它提供了开发多线程应用程序的核心方法。

- 创建新线程，在系统启动或者已经运行的时候。
- 控制不同的线程，比如操作线程的优先级。
- 一个调度器，决定哪个线程当前可以运行。
- 一组同步元语，允许多线程通讯和安全共享数据。
- 集成系统中断和异常。

eCos 内核包也是一个可选的。你完全可以写一个单线程的程序，不需要用到内核的任何功能。当需求变得复杂的时候，就适合用多线程来解决，这时就需要内核包了。实际上 eCos 中一些更高级的包，比如 TCP/IP 协议栈，它内部就使用了多线程。因此，当应用需要用到这些包的时候，内核就必须包含，而不再是可选的了。

内核的功能可以通过两种方式来使用。内核提供了它自己的 API，比如 cyg_thread_create 和 cyg_mutex_lock 等一些函数。这些函数可以直接被应用程序或者其它的包调用。还有一种方式是使用一些包中提供的兼容 API，比如 POSIX 或μITRON。这些兼容层允许应用程序调用标准的 API，比如 pthread_create，这些 API 由底层的 eCos 内核 API 实现。应用程序中使用兼容 API 可以使程序更加简单，也可以在其它系统中减少代码量，共享代码，方便移植。

调度器

当一个系统包含多线程的时候，就需要一个调度器来决定哪个线程可以在当前运行。eCos 的内核可以被配置成两种：位图调度和多级队列调度。

调度方式	调度方法	时间片调度	优先级数/个	任务数量/个
位图	优先级抢占式	无	32	32

多级队列	优先级抢占式 同级时间片调度	有	32	无限

两种调度器都使用简单的数字优先级来决定哪个线程应该运行。优先级的级别用数字表示，可以通过 CYGNUM_KERNEL_SCHED_PRIORITIES 选项配置，但是一个典型的系统一般会有 32 个优先级别。因此线程的优先级一般会在 0--31 范围内。0 是最高级别，31 是最低级别。通常只有系统的空闲线程会运行在最低级别。线程的优先级是绝对的，因此内核只会在所有的高级别线程阻塞的时候，才会运行低级别的线程。

位图调度器只允许每个级别一个线程，所以如果系统配置成 32 级别，那么就最多只有 32 个线程--仍然满足大多数应用程序。一个简单的位图调度器可以被用来追踪当前哪些线程可以运行。

多级队列调度器则提供多个线程共用一个优先级别，这意味着系统对线程的数量没有限制，只要在系统内存允许的条件下。但是操作，像查找最高级别的可运行线程，将会比位图开销更大。eCos 采用多级队列调度方式支持 SMP，该方式采用优先级抢占式，同级时间片轮转调度方法进行多线程的调度，每一个 CPU 在当前时间均可运行一个线程，从而达到并行计算的目的。

优先级抢占是停止执行低优先级的线程，然后允许更高级别的线程的执行。

时间片轮转，调度器在一定的时钟滴答数到来时，自动在多个优先级相同的线程间选择运行。时间片轮转只会发生在在两个可运行的线程处在同一优先级别，并且没有更高级别的可运行线程存在的时候。如果时间片轮转被关闭了，那么一个线程就不能抢占另一个相同级别的线程，只能等到那个线程运行完成或者被阻塞的时候（比如等待一个同步元语），才能运行。配置选项 CYGSEM_KERNEL_SCHED_TIMESLICE 和 CYGNUM_KERNEL_SCHED_TIMESLICE_TICKS 控制着时间片轮转。

同步机制

eCos 内核为系统中的线程提供了彼此间通信以及同步访问共享资源的机制，由 eCos 提供的机制有：

- 互斥量(mutex)；
- 条件变量(condition variables)；
- 信号量(semaphore)；
- 消息邮箱(message boxes)；
- 标志量(flags)；
- 螺旋锁(spinlocks)。

互斥量(mutex)允许多个线程安全、串行地分享一个资源：一个线程锁住互斥量，然后操作共享资源，最后解锁互斥量。这个资源可以是一段内存区域或者是一个硬件。一个互斥量类似于一个二进制的信号量，它只有两个状态：锁定和释放。但是，互斥量与二进制信号量之间有点差异，互斥量提供不同优先级的继承保护，而二进制信号量没有这个功能。

条件变量(condition variables)与允许多个线程访问共享数据的互斥量一起使用。当一个线程锁住一个互斥量需要等待某个条件变成真的时候，你应该使用一个条件变量。条件变量本质上是提供给一个线程等待的空间，其它的线程或 DSR 可以使用它来唤醒那个线程。当一个线程等待一个条件变量的时候，它会在之前释放互斥量，并且在唤醒前重新要求得到互斥量，然后才能接着处理。

信号量(semaphore)是一个包含表明资源被锁定或可用的计数器的同步机制。有两种类型的信号量：技术型和二进制型。二进制型的信号量类似于计数型的信号量，但是，它们的计数值在传递一个值之后不会增加，二进制信号量可处于锁定或释放状态。

消息邮箱提供了一种两个线程交换信息的方法。典型的是，一个线程产生消息并将消息送给另一个线程来处理。消息邮箱为线程传输多于一个字节的信息提供了另一种方法。

标志量(flags)是用一个 32 位的字表示的同步机制。每一个标志量中的字节代表一个条件，它允许一个线程等待一个条件或者条件组合。等待的线程指定是否满足所有条件或者某些条件的组合后唤醒。然后激发线程能根据指定条件进行置位或者重置位，以使适当的线程执行。

螺旋锁(spinlocks)是 eCos 提供的一种适用于 SMP 系统上运行的应用程序的同步机制。其它同步机制在 SMP 系统上也起作用。螺旋锁实际上就是在一段特定代码执行之前处理器能对其进行检查的标志量。如果螺旋锁没有被锁定，则处理器能置位标志量并继续执行线程；如果螺旋锁被锁定，则线程就在紧凑地循环，继续检查标志量直到该标志量被释放。螺旋锁在比其它同步机制更低的级别下运行，并且实现的方法跟硬件相关。一些处理器为螺旋锁的实现提供了一个测试并设置指令。没有获得螺旋锁的线程不被挂起，所以，将螺旋锁运行的指令控制在一段很短的时间内是很重要的，典型的就是 10 或 12 条指令。

8.2 集成开发环境 orion5.0

8.2.1 orion5.0 简介

Orion 5.0 是应用于嵌入式软件开发的集成开发环境,运行于 Windows 操作系统，实现了对多核并行处理器 S698P4-II 的支持。

Orion 5.0 它提供高效、清晰且图形化的嵌入式应用软件开发平台，包括一整套完备的、面向嵌入式系统的开发和调试工具：编辑器、编译器、链接器、调试器和工程管理器等。用户可以很方便地在 Orion 5.0 集成开发环境中创建和打开工程，建立、打开和编辑文件，编译、链接、运行、调试各种嵌入式应用程序。

Orion 5.0 同时提供详细的使用文档和参考例程，使用户可以快速、简捷的开发各种基于 S698P4-II 的并行处理程序。

8.2.2 orion5.0 的特点

Orion5.0 的主要特点有：

- 运行于 Windows 操作系统；
- 支持 S698P4-II　32 位四核处理器芯片；
- 支持国际流行的 GRMON 调试器；
- 集成国际流行的 GUN 的 GCC 编译器；

- 支持 eCos 实时操作系统(实现了对 SMP 系统的支持)；
- 提供符合 POSIX 接口标准的多线程库；
- 提供多个典型、成熟的并行编程例程：浮点运算、快速傅立叶、小波变换、Cannon 算法等；
- 支持开发语言：C、C++或汇编；
- 多语言支持，可切换中/英文、简/繁体；
- 界面友好，使用方便，与 Microsoft Visual Studio 相似；
- 图形化的工程管理工具，包括向工程新建、添加、删除文件/类等；
- 支持源码显示和调试：断点设置、单步运行、反汇编、内存观察、寄存器访问、变量自动跟踪、符号表分解等；
- 支持多种调试方式，包括本地仿真调试、本地目标系统在线调试、远程网络目标系统在线调试。

8.2.3 orion5.0 的组成

Orion5.0 按功能可以划分为以下几部分：工程管理；代码编辑；程序编译；程序调试；ROM 映像文件。

工程管理

工程管理是对程序项目和程序文件进行管理。它包括：工作区选择；工程建立；工程打开；工程导入；工程关闭；工程删除；工程文件导入；工程文件删除。工程管理模块使用户直接、方便的管理工程及其文件。

代码编辑

在编写程序代码的过程中，编辑器具有支持语法关键字的色彩显示，具体的显示色彩可由用户自由编辑。这样就方便了用户代码编写，提高了代码的编写质量。编辑器还提供了代码协助功能。在编辑器中输入 C/C++的关键词，然后

按 ALT+/，再根据提示选择你想要的协助代码。除了 Orion5.0 自带的关键词支持，你可以自定义代码协助模板或者编辑现存的代码协助模板(在菜单 Window→Preferences→C/C++的 Templates 设置页里面设置)。

编辑菜单支持标准的 Windows 文本编辑功能：撤消键入、重复键入、剪切、复制、粘贴、清除、全选等。在编辑窗口中，还可通过鼠标选中文本进行拖拉移动操作。快捷键设置和 Windows 文本编辑器的快捷键设置基本一致:

CTRL+C:复制

CTRL+X:剪切

CTRL+V:粘贴

CTRL+F:查找/替换

CTRL+D:删除整行

要查看全部的快捷键定义:CTRL+SHIFT+L

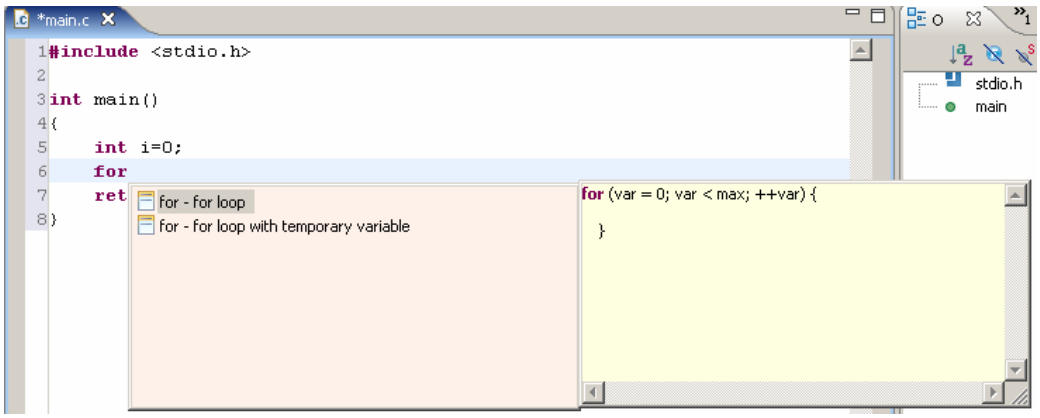


图 8-2 代码协助功能

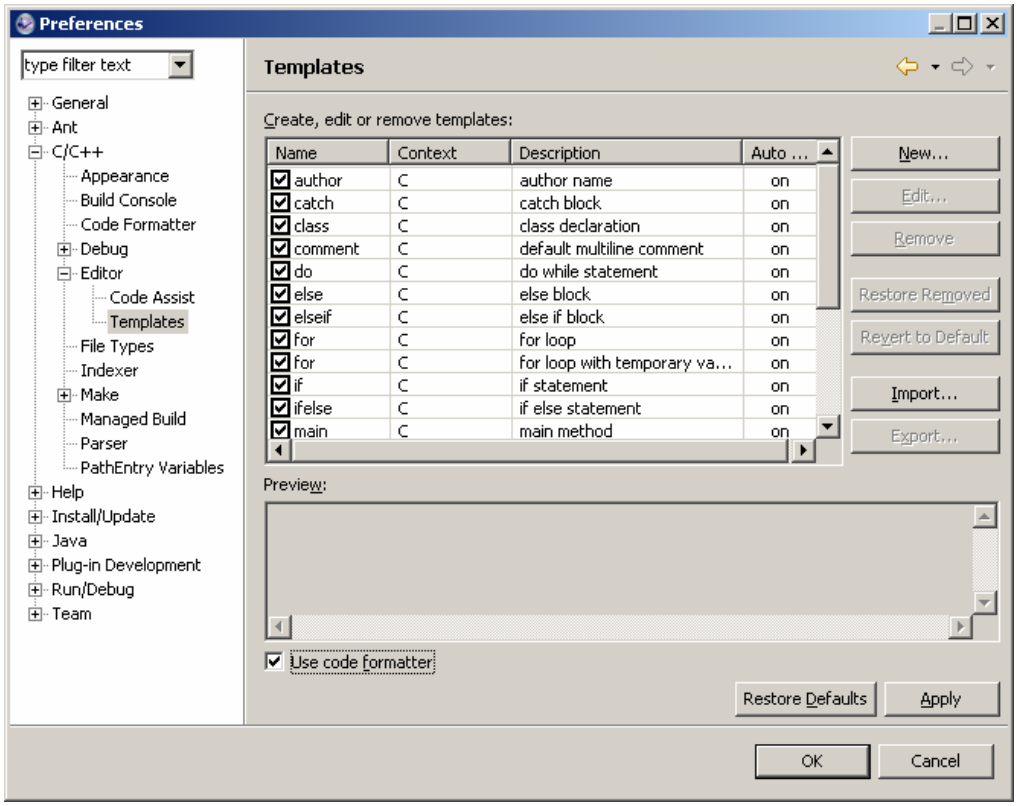


图 8-3 设置代码协助模板

程序编译

Orion5.0 提供两个编译器 sparc-rtems-gcc 编译器和 sparc-elf-gcc 编译器给用户选择。正确配置编译选项是成功通过编译的前提。

Rtems c 工程一般选择 sparc-rtems-gcc 编译器，在选择工程类型时默认选择。eCos c 工程一般选择 sparc-elf-gcc 编译器，在选择工程类型时默认选择。bare c 工程两种编译器都可以使用，默认是 sparc-rtems-gcc 编译器。

编译结果信息在控制台输出。

程序调试

调试是嵌入式软件开发中的重要环节，Orion5.0 强大的调试功能允许您对程序进行单步跟踪，设置断点，观察变量，察看堆栈等等。

Orion5.0 支持以下的调试方式：

- 仿真调试模式（S698、TSC695F 平台适用）；

- 在线调试模式（S698 平台适用）；
- 调试监听调试模式（TSC695F 平台适用）。

1. 仿真调试模式：

调用软件模拟器调试程序，这种方式无需硬件平台，调试方便、简捷，但要注意大多数涉及到硬件处理的程序都无法通过模拟器调试。Orion5.0 开发环境中，针对 S698 系列处理器的模拟器为 Sim-698，针对 TSC695F 处理器的模拟器为 Sim-695。

2. 在线调试模式：

用于 S698 平台的在线硬件调试。搭建宿主机/目标板调试环境，主机通过串口与目标板连接，利用 S698 系列处理器的内部 DSU 调试单元，下载程序并运行，以完成交叉调试。这种方式采用真实的运行环境，能及时准确的定位问题。调试时仅需一根串口线，无须外接仿真器，这也正是 S698 系列处理器的一大优势。

针对 S698P4-II 并行处理器，Orion5.0 将默认调用硬件调试器 GRMON；其他 S698 系列处理器，用户可以选用硬件调试器 V8mon.exe。

图 8-4 宿主机/目标板调试环境

3. 调试监听调试模式：

用于 TSC695F 平台的硬件调试。因为 TSC695F 处理器内部无 DSU 调试单元，需要事先在开发板的 ROM 中烧写监听程序，调试时监听负责与主机通讯，完成主机交付的各种调试指令。采用这种模式，主机与目标板之间需要两根串口线连接，一个负责调试通讯，一个负责打印输出。

ROM 映像文件

代码调试无误后，就可以生成 ROM 映像文件烧制到目标系统的 ROM 或

者 Flash 芯片中。烧写完成后，重新上电，程序可自动运行。

从编译生成的可执行文件到 ROM 映像文件还需要一些过程，Orion5.0 具备自动生成 ROM 映像文件和在线烧写的功能。

使用 Orion5.0 生成和烧写 ROM 映像文件可以有图形界面和命令行两种方式。下文的操作步骤中假设使用的硬件系统为 Orbita SPARC V8 开发系统，编译完成的目标文件为 test.exe。