GSC3281 用户手册-精简版

2013 年 8 月

版本号：3.2

北京神州龙芯集成电路设计有限公司

BLX IC Design Co., Ltd

目录

图目录

表目录

1 产品概览

1.1 产品简介

GSC3281 芯片是一款主要面向工业终端类应用的 SOC 芯片，采用 0.13um 标准 CMOS 制造工艺，主频 200~300MHZ。GSC3281 以 32 位龙芯处理器作为主控处理器，并在片内集成了丰富的功能模块与外围设备，包括 10/100Mbps 以太网 MAC 控制器、USB2.0 OTG 控制器、DDR2 控制器、NAND Flash 控制器、I2C、I2S、UART、SPI、PWM、旋转编码器鉴相器、定时器等等。丰富的片内集成设备提高了整体性能，降低了系统成本，并可以满足更多的应用需求。

GSC3281 芯片的一个显著特征是采用了 32 位龙芯处理器作为主控处理器。32 位龙芯处理器是一款 7 级流水乱序执行 RISC 处理器，具有独立的 16KB 指令 cache 与 16KB 数据 cache，支持 MMU，可以运行 Linux、WinCE、VxWorks 等主流操作系统。

GSC3281 内部总线架构采用了主流的 AXI、AHB、APB 总线，并针对片内设备的特点与应用需求进行了一系列设计结构优化，提高了系统性能与数据吞吐率。同时，GSC3281 芯片采用了一系列的低功耗优化措施，包括休眠模式、关闭无用时钟、低频运行等，软硬件的配合可以显著降低整个芯片的功耗。针对具体应用，通过软硬件优化措施，GSC3281 芯片可在性能、功耗、灵活性等方面达到一个最佳的平衡。

GSC3281 芯片具有较好的通用性，除了云计算、工业控制、税控终端、智能电网集中器等终端类应用之外，还可以拓展应用于不同的相关领域，例如信息安全领域。

GSC3281 芯片集成了丰富的片上功能，本手册后续部分将分别对此进行详细的介绍。GSC3281 芯片主要集成了如下的片上功能：

- 32 位龙芯处理器作为主控处理
- 支持 NAND Flash 与 SPI 两种启动模式
- DDR2 控制器，最高支持 16x533Mbps 数据传输速率
- NAND Flash 控制器，支持 8 位 SLC/MLC NAND Flash 颗粒
- 外部静态存储器接口（EMI 接口）
- 10/100Mbps 自适应以太网 MAC 控制器
- USB2.0 OTG 控制器
- 多通道 DMA 控制器
- 可编程中断控制器
- 1 路旋转编码器接口与 3 通道 PWM 接口，支持电机控制 PWM 与普通 PWM
- 8 个 UART 接口，支持包括 2/3/8 线以及 232/485 等不同类型的串口
- 2 个 SPI 主机接口
- 4x4 矩阵键盘接口
- 2 个 7816 主机接口
- 1 个 I2C 主机接口
- 1 个 I2S 主机接口
- 2 个 PS2 主机接口
- 4 路 ADC 接口
- 4 个可编程定时器

- 1 个看门狗定时器
- 87 个可编程 GPIO 引脚与 31 个外部中断源
- 片上可编程 PLL 时钟发生器

1.2 功能特性

龙芯处理器

- 32 位 RISC 体系结构
- 7 级乱序执行流水线
- 包含硬件乘法器与除法器
- 16KB 指令 cache 与 16KB 数据 cache
- 关键字优先与非阻塞 cache
- 支持 MMU 并包含 32 项 TLB
- 支持 EJTAG 片上调试功能
- 采用 AXI 接口
- 支持 Linux、WinCE 等主流操作系统

两种启动模式

- NAND Flash 启动
- SPI Flash 启动
- 通过引脚配置选择启动模式

DDR2 内存控制器

- 最高数据传输速率为 16x600Mbps
- 16 位数据位宽
- 最大支持 256MB 存储容量
- 支持 1 个 rank
- 软件可配置 PHY 时序

NAND Flash 控制器

- 支持 8 位 SLC/MLC NAND Flash 颗粒
- 支持 2KB/4KB 页大小
- 支持硬件 BCH ECC 校验码
- 支持省电模式
- 读写时序参数可配置
- 具有启动功能
- 内置 DMA 引擎

外部静态存储器接口（EMI 接口）

- 支持采用异步 SRAM 时序的接口设备
- 支持 8 位数据位宽
- 最大支持 3 个片外设备
- 对于非 SRAM 类型的设备可支持 READY 握手信号

10/100Mbps 以太网 MAC 控制器

- 支持 IEEE 802.3 协议
- 支持标准 RMII 接口
- 支持 10/100Mbps 传输速率
- 支持全双工和半双工操作模式
- 内置接收和发送 DMA
- 自动丢弃错误帧
- 支持对特殊 MAC 地址的检测
- Hash 表支持对单播和多播地址的匹配
- 支持混杂模式，即可接收 LAN 中所有帧
- 支持 VLAN 帧的识别
- 支持 IP 报文头部 checksum 字段检验
- 支持 TCP/IP 报文中 checksum 字段插入

USB2.0 OTG 控制器

- 支持主机与设备模式
- 支持非点对点模式（即支持 HUB）
- 内置 DMA 引擎
- 主机模式下共有 8 个 channel
- 设备模式下共有 5 个 endpoints
- 内部 FIFO 大小为 1024x35bit

多通道 DMA 控制器

- 4 通道 DMA
- 支持存储器到存储器、存储器到外设、外设到存储器、外设到外设等传输类型
- 支持 single-block 与 multi-block 传输
- 支持软件握手与硬件握手的 DMA 请求
- 支持 16 个硬件握手请求

可编程中断控制器

- 支持 27 个中断源
- 高电平触发中断
- 每一个中断可分别进行使能与屏蔽
- 所有中断源具有相同的中断优先级
- 软件可强制某一个中断源产生中断

- 可在时钟关闭的情况下接收中断并向 CPU 发出中断请求

SPI 主机接口

- 2 个 SPI 接口，均为主机模式
- SPI1 接口支持启动功能
- 支持查询、中断和 DMA 传输模式
- 支持 256 种波特率
- 支持 Byte Sleep
- 数据帧长度可配置为 2-17 位
- 支持 MSB 优先或 LSB 优先
- 支持全双工通信
- 支持全部四种 SPI 模式

4x4 矩阵键盘接口

- 支持最大 4x4 矩阵键盘
- 待机模式下可通过按键产生唤醒中断
- 支持单键与同时按下任意双键的情况

7816 主机接口

- 两个 7816 主机接口
- 支持 ISO7816-3 协议
- 异步半双工模式
- 支持 T=0 协议
- 支持可编程波特率
- 支持正向模式和反向模式
- 支持奇偶校验。
- 支持自动重传

3 通道 PWM 与旋转编码器接口

- 支持 3 个独立的 PWM 通道
- 支持一个增量式旋转编码器
- PWM 支持两种工作模式：普通 PWM 模式与电机控制 PWM 模式
- PWM 模式下可以产生 6 个单边沿输出、3 个双边沿输出或者混合输出
- PWM 模式下未用通道可用作定时器
- 电机 PWM 模式下每个通道产生两个极性相反的输出
- 支持 3 个捕获输入
- 支持 1 个快速终止输入

UART

- 8 个 UART，均兼容 16550a
- 支持 5~8 位数据位
- 支持 1/1.5/2 位停止位
- UART0/1/2/7 支持 2 线 232 连接
- UART3/4/5 支持 3 线 485 连接
- UART6 支持 8 线全功能串口
- UART3/4/5/6 支持 DMA 传输
- UART7 支持红外接口
- 232 连接支持最大波特率 3.7Mbps
- 485 连接支持最大波特率 12Mbps

I2C 主机接口

- 支持主机模式
- 支持标准、快速与高速三种传输速率
- 支持 7/10 位寻址方式
- 支持查询、中断与 DMA 传输方式

I2S 主机接口

- 支持主机模式
- 1 个接收通道与 1 个发送通道
- 支持 12/16/20/24/32 位采样宽度
- 支持 DMA 传输模式

PS2 主机接口

- 支持两个 PS2 主机接口
- 可用于连接鼠标和键盘
- 11 位数据帧格式
- 独立的发送与接收模块
- 支持查询和中断传输模式

ADC 接口

- 4 通道 SAR 型 ADC
- 12 位精度
- 支持最高采样率为 120Ksps
- 支持低功耗模式

可编程定时器

- 4 个 32 位定时器
- 每个定时器时钟独立可配置
- 支持循环定时与单次定时两种工作模式

看门狗定时器

- 32 位看门狗定时器
- 可配置看门狗定时器计数时钟

- 发生超时的时候，允许直接产生系统复位信号，也允许先产生中断再产生系统复位信号

可编程 GPIO 引脚

- 87 个 GPIO 引脚，每个引脚独立可配置
- 其中 31 个 GPIO 可用作外部中断源
- 支持高电平、低电平、上升沿、下降沿等 4 种中断触发模式
- 具有内部消抖电路可用于对外部中断源输入进行消抖

片上 PLL

- 输出频率范围 62.5MHZ~1500MHZ
- 外部引脚可配置 8 种 PLL 输出频率
- 运行时软件可灵活配置 PLL 输出频率

工作电压

- 核心电压：1.2V
- IO 电压：3.3V
- DDR2 接口电压：1.8V
- USB2.0 OTG 数字电压：1.2V
- USB2.0 OTG 模拟电压：3.3V
- PLL 模拟电压：1.2V
- PLL 数字电压：1.2V
- ADC 模拟电压：3.3V
- ADC 数字电压：1.2V

温度范围

- -40℃~85℃

工作频率

- DDR2 PHY 最高工作频率 600MHZ
- CPU 最高工作频率 300MHZ
- AXI 总线最高工作频率 300MHZ
- AHB 总线最高工作频率 200MHZ
- APB 总线最高工作频率 66.7MHZ

封装

- LFBGA256 封装

1.3 结构框图

GSC3281 芯片以龙芯处理器为核心，集成了丰富的片上设备，所有的功能模块通过 AMBA 总线进行连接，包括 AXI 总线、AHB 总线以及 APB 总线，构成了一个独立的片上系统。GSC3281 芯片的整体结构框图如图 1-1 所示。

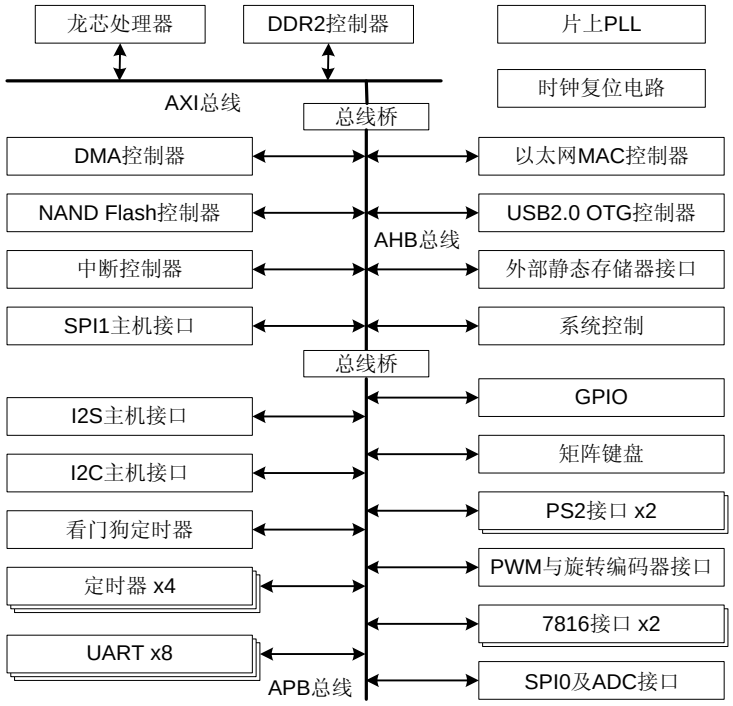


图 1-1 GSC3281 结构框图

2 存储空间

2.1 地址映射

GSC3281 芯片以 32 位龙芯处理器作为中央处理器，具有 4GB 的存储器逻辑地址空间，所有设备与存储器采用统一编址的方式；对于设备内部的控制与状态寄存器，由于映射到存储器空间，因此可以采用与访问存储器一样的访问方式。

在 32 位龙芯处理器的体系结构中，将逻辑地址空间划分为 5 个地址段，分别为 kuseg、kseg0、kseg1、ksseg 以及 kseg3，如**错误！未找到引用源。**所示；其中 kuseg、ksseg、kseg3 可以映射到物理地址中的任意位置，而 kseg0、kseg1 则固定映射到物理地址的最低 512MB 空间。对于低 512MB 物理地址空间，由于可以从不同的逻辑地址空间映射过来，因此可以通过不同的方式访问相同的物理地址。

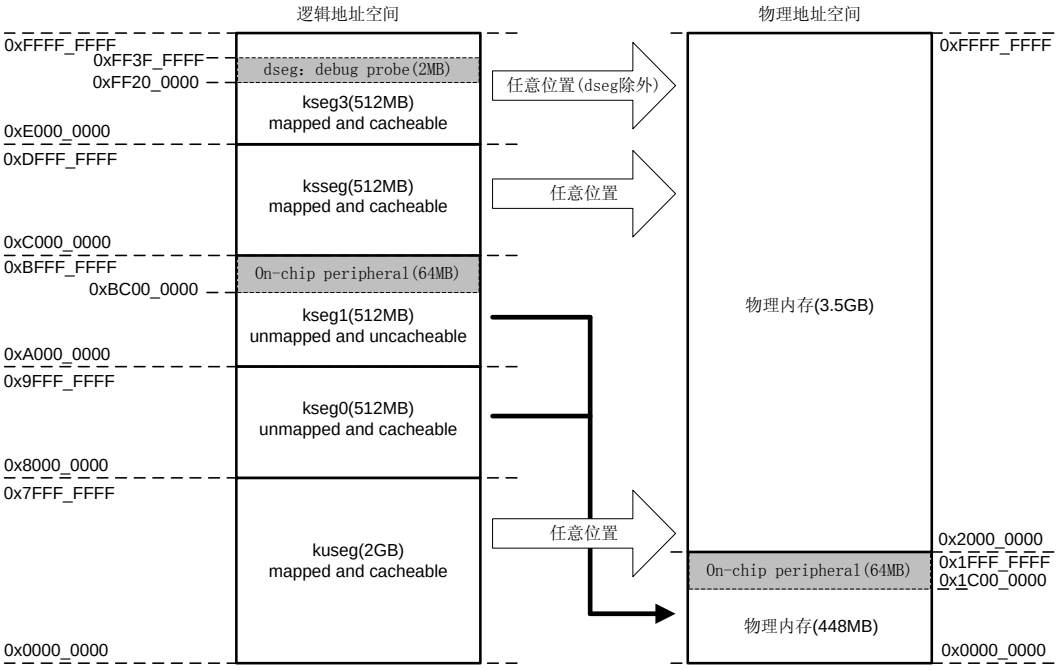


图 2-1 龙芯处理器逻辑地址空间与物理地址空间的映射关系

GSC3281 芯片中所有设备与存储器采用统一编址的方式，所有片内设备的逻辑地址映射到 kseg1 中从 0xBC00_0000 开始的 64MB 空间，对应于物理地址中从 0x1C00_0000 开始的 64MB 空间。通过访问相应的存储器逻辑地址，在芯片内部自动转换为对应的物理地址，就可以访问到设备中对应的控制与状态寄存器，从而控制片内设备实现各种期望的功能。

在分配给片内设备单元的 64MB 地址空间中，针对不同的功能单元分别划分了 4KB、8KB 以及 256KB 不等大小的空间，如表 2-1 所示。表 2-1 给出了片内设备在 kseg1 段的逻辑地址空间，软件可以通过读写相应的 kseg1 段逻辑地址来访问与控制这些片内功能设备；如果必要，软件也可以通过其他地址段的逻辑地址来访问这些片内设备。

表 2-1 GSC3281 芯片的存储器地址映射表

物理基地址	逻辑基地址(kseg1)	大小/Byte	功能单元
0x00000000	0xA0000000	448M	物理内存
0x1C000000	0xBC000000	256K	USB2.0 OTG 控制器
0x1C040000	0xBC040000	8K	保留
0x1C042000	0xBC042000	8K	DMA 控制器
0x1C044000	0xBC044000	8K	NAND Flash 控制器
0x1C046000	0xBC046000	8K	保留
0x1C048000	0xBC048000	8K	中断控制器
0x1C04A000	0xBC04A000	8K	系统控制
0x1C04C000	0xBC04C000	8K	SPI1 主机接口
0x1C04E000	0xBC04E000	8K	外部静态存储器接口控制器
0x1C055000	0xBC055000	-	保留
0x1C100000	0xBC100000	4K	保留
0x1C101000	0xBC101000	4K	SPI0 主机接口
0x1C102000	0xBC102000	4K	矩阵键盘
0x1C103000	0xBC103000	4K	7816-0 主机接口
0x1C104000	0xBC104000	4K	7816-1 主机接口
0x1C105000	0xBC105000	4K	旋转编码器与 PWM
0x1C106000	0xBC106000	4K	PS/2-0 接口
0x1C107000	0xBC107000	4K	PS/2-1 接口
0x1C108000	0xBC108000	4K	UART0
0x1C109000	0xBC109000	4K	UART1
0x1C10A000	0xBC10A000	4K	UART2
0x1C10B000	0xBC10B000	4K	UART3
0x1C10C000	0xBC10C000	4K	UART4
0x1C10D000	0xBC10D000	4K	UART5
0x1C10E000	0xBC10E000	4K	UART6
0x1C10F000	0xBC10F000	4K	UART7
0x1C110000	0xBC110000	4K	可编程输入输出接口
0x1C111000	0xBC111000	4K	I2C 主机接口
0x1C112000	0xBC112000	4K	I2S 主机接口
0x1C113000	0xBC113000	4K	DDR2 控制器
0x1C114000	0xBC114000	4K	可编程定时器
0x1C115000	0xBC115000	4K	看门狗定时器
0x1C116000	0xBC116000	4K	DDR2 PHY
0x1C117000	0xBC117000	8K	以太网 MAC 控制器
0x1C119000	0xBC119000	-	保留
0x1DC00000	0xBDC00000	1.5K	EMI 设备空间
0x1E000000	0xBE000000	-	保留
0x1FC00000	0xBFC00000	4M	4MB 启动空间
0x20000000	-	3.5G	物理内存

2.2 片内存储器

GSC3281 芯片所有片内设备映射到从 0x1C000000 开始的 64MB 物理地址空间，对应于 kseg1 段从 0xBC000000 开始的 64MB 逻辑地址空间。在这 64MB 空间中，除了未实际使用的空间，其他绝大多数地址都用于各种控制与状态寄存器，只有 4KB 空间用于片内存储器。这 4KB 片内存储器是 NAND Flash 控制器的片内缓冲区，用于读写 NAND Flash 颗粒时对一个 4KB 大小的页进行缓冲。该片内存储器分别映射到两个地址空间，分别为 0xBC044000 开始的 4KB 空间，用于 NAND Flash 控制器的数据缓冲，以及 0xBFC00000 开始的 4KB 地址空间，用于 NAND Flash 启动。当 NAND Flash 控制器不工作时，可以将这 4KB 片内存储器用作普通的数据存储区域，通过 0xBC044000 开始的 4KB 逻辑地址空间即可以访问该片内存储器。

2.3 片外存储器

GSC3281 芯片支持多种常用的片外存储器，为不同的应用提供了多样的选择，这些存储器类型包括有：

- DDR2 存储器；
- NAND Flash 存储器；
- SPI 接口 NOR Flash 存储器；

2.3.1 DDR2 存储器

GSC3281 芯片集成了 DDR2 控制器，可支持最大 256MB 的 16 位 DDR2 存储器，最高数据传输速率可达 16x533Mbps。DDR2 控制器有 AXI 和 AHB 两个主机端口，可区分不同接口类型和不同优先级的片内主设备，例如 CPU 和 DMA 请求。DDR2 控制器提供了先进的配置特性，例如可配置端口带宽和读操作预取长度等，根据需要可由软件进行微调以优化带宽和延迟。

2.3.2 NAND Flash 存储器

GSC3281 芯片集成了 ONFI 1.0 协议 NAND Flash 控制器，支持 8 位 NAND Flash 颗粒芯片，页大小可配置为 2KB 或者 4KB，采用的硬件校验算法为 BCH8，支持 NAND Flash 启动。

2.3.3 SPI Flash 存储器

GSC3281 芯片集成了两个 SPI 主机接口，两个 SPI 接口都可以外接任意大小的 SPI Flash。SPI1 接口支持系统从 SPI Flash 里启动，系统启动空间大小为 4MB，SPI0 接口不支持系统启动。在启动完成后，软件可通过正常的访问方式继续读写 SPI1 接口的 SPI Flash 芯片。

3 龙芯处理器

3.1 概述

GSC3281 芯片以 32 位龙芯处理器作为中央处理器，32 位龙芯处理器是一款由我国自主研发的 RISC 处理器，具有独立的 16KB 指令 cache 与 16KB 数据 cache，支持包含 32 项 TLB 的 MMU，并采用了一系列先进的体系结构设计技术，包括七级流水、动态调度、乱序执行、非阻塞 cache 等，具有出色的性能，可以运行 ucLinux、VxWorks、Linux、WinCE 等主流操作系统。

3.2 功能特性

32 位龙芯处理器具有如下的关键功能特性：

- RISC 体系结构
- 32 位地址与数据格式，采用小尾端次序
- 乱序执行内核，7 级基本流水线
- 独立的 16KB 指令 cache 与 16KB 数据 cache，分别为四路组相联结构
- 关键字优先与非阻塞 cache
- 支持 MMU，包含 32 项 TLB
- 硬件乘法器与除法器
- AMBA AXI 接口
- 支持 EJTAG 片上调试功能

3.3 结构框图

32 位龙芯处理器的结构框图如图 3-1 所示，主要组成部件包括有取指译码部件、调度内核、运算部件、MMU、CP0 协处理器、指令 cache、数据 cache、AXI 总线接口以及 EJTAG 调试单元等。

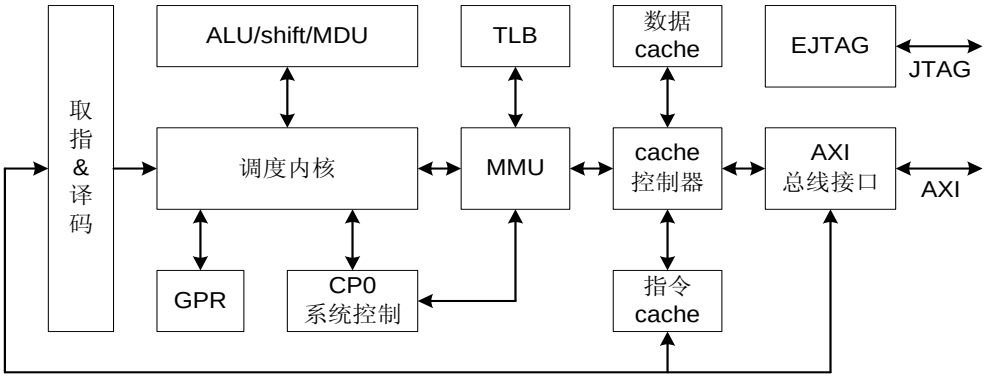


图 3-1 32 位龙芯结构框图

3.4 流水线

32 位龙芯处理器内部具有七级基本流水线，采用了单发射乱序执行内核，通过寄存器重命名、动态调度以及转移预测等先进的体系结构技术，使得指令在处理器中可以不按照原始顺序执行，减缓或者消除了数据相关与控制相关带来了的流水线停顿，提高了流水线的执行效率与处理器的整体性能。

32 位龙芯处理器的七级基本流水线包括取指、译码、重命名、发射、执行、写回与提交共七个流水阶段；在取指阶段，CPU 访问 TLB 与指令 cache，取回需要的指令，并根据不同指令进行部分预先译码操作；在译码阶段，对取回的指令进行译码并转换为统一的内部编码；在重命名阶段，根据流水线中正在执行的各条指令的运行情况，对当前指令的逻辑寄存器进行动态重命名以解决相关问题；在发射阶段，当前指令读取通用寄存器堆的数据并发射到各执行部件中；在执行阶段，各执行部件从操作数都准备好的指令中选取最老的指令进行执行；在写回阶段，各运算部件将运算结果写回到指令重排队列中；在提交阶段，将指令重排队列中的指令运算结果写回到通用寄存器堆中，并更新相应的处理器状态。在没有流水线停顿的理想情况下，每一个流水节拍可以完成一条指令；而当出现数据相关、cache 不命中等情况时，则将引起流水线停顿，使得 CPU 无法一个节拍完成一条指令。

由于 32 位龙芯处理器的流水线是动态流水线，因此执行不同指令时所需要的流水级是不同的；对于大多数指令而言，通过七级基本流水线就可以完成执行；但某些复杂指令则需要更多的流水级节拍，例如访存指令流水线为八级，定点乘法与定点除法指令的流水线级数则根据操作数的不同而存在变化。

3.5 寄存器

32 位龙芯处理器提供了三类软件可见的寄存器：32 个 32 位的通用寄存器、两个 32 位的整数乘法与除法运算结果寄存器、若干 32 位系统控制寄存器。

对于 32 个通用寄存器而言，r0 与 r31 是两个特殊的寄存器。其中 r0 寄存器硬连线为 0，可以用作任何需要零值的指令的源寄存器；r0 也可以用作任何指令的目标寄存器，但 r0 寄存器的零值不会被更改，即该指令的运算结果将被丢弃。r31 寄存器除了用作普通的通用寄存器以外，还用于转移并链接指令的链接寄存器，即用于存储将来要返回的指令 PC。

定点乘法与除法结果寄存器 HI 与 LO 专门用于存储定点乘法的乘积与定点除法的商及余数，其中 HI 寄存器用于存储乘法结果的高 32 位或者除法结果的商，LO 寄存器用于存储乘法结果的低 32 位或者除法结果的余数。为了访问 HI 与 LO 寄存器，32 位龙芯处理器提供了 MTHI、MTLO、MFHI、MFLO 等四条专门的指令。

系统控制寄存器主要用于存储管理、例外处理等系统控制功能，只能在内核状态下访问该类寄存器，有关系统控制寄存器的更多内容请看“CP0 协处理器”一节。

3.6 存储管理

3.6.1 工作模式

32 位龙芯处理器支持四种工作模式：

- 用户模式(user mode)

- 监管模式(supervisor mode)
- 内核模式(kernel mode)
- 调试模式(debug mode)

32 位龙芯处理器支持最大 4GB 的逻辑地址空间，4GB 逻辑地址空间划分为不同的地址段，当处理器处于不同的工作模式时，对不同地址段具有不同的访问权限，虚实地址转换过程也会存在不同。

3.6.1.1 逻辑地址空间

32 位龙芯处理器最大支持 4GB 逻辑地址空间，当处理器处于不同工作模式时，可以访问 4GB 逻辑地址空间的不同地址段，如图 3-2 所示。

在复位之后，或者发生例外的时候，龙芯处理器将进入到内核模式；在内核模式下，软件可以访问整个 4GB 逻辑地址空间以及 CP0 寄存器。

某些分层操作系统可由内核模式进入到监管模式，在监管模式下，软件只能访问 4GB 逻辑地址空间中的 suseg 段与 sseg 段；如果软件试图访问 suseg 段与 sseg 段之外的地址空间，则将触发一个地址错误例外；监管模式下不能访问 CP0 寄存器。

在用户模式下，软件只能访问 4GB 逻辑地址空间中的 useg 段；如果软件试图访问 useg 段之外的地址空间，则将触发一个地址错例外；用户模式下软件不能访问 CP0 寄存器。

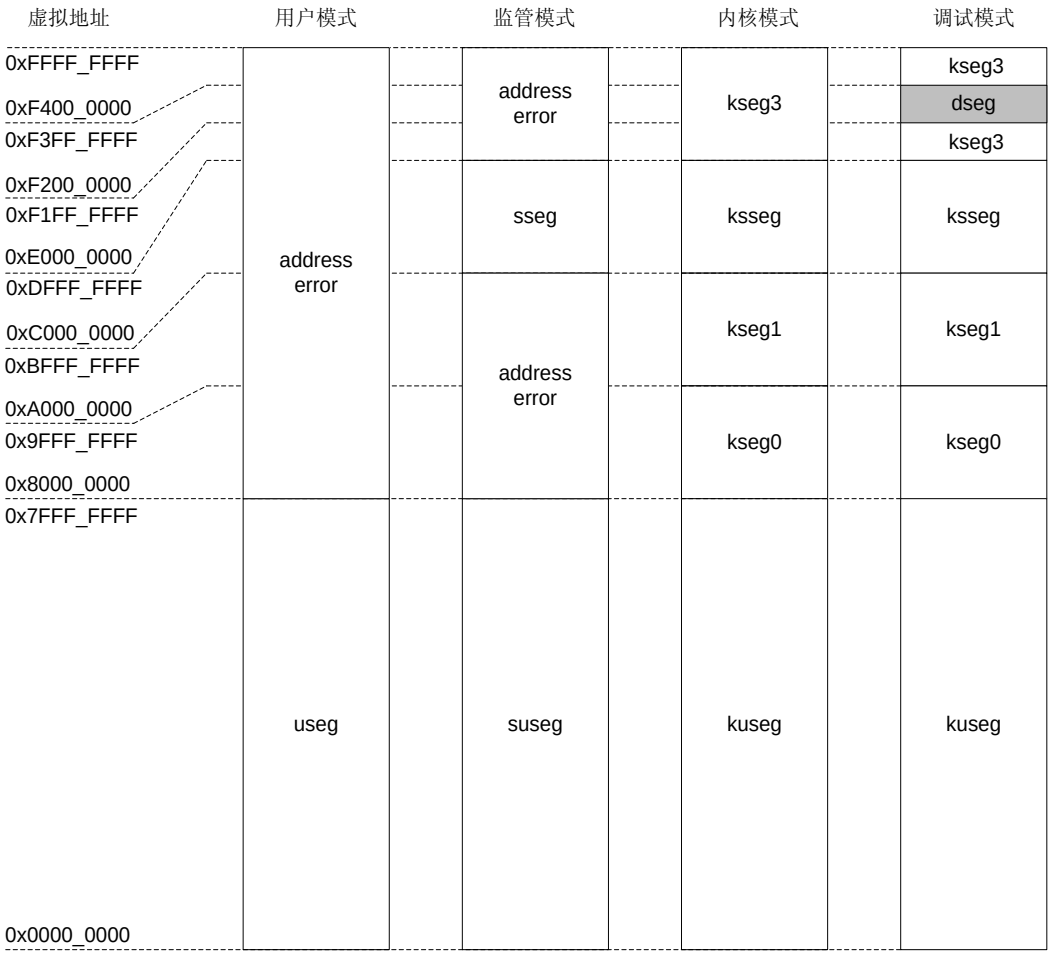


图 3-2 32 位龙芯处理器的逻辑地址空间与地址段划分

当发生调试例外时，龙芯处理器将进入到调试模式，在调试模式下，软件除了具有和内

核模式一样的访问权限，额外还可以访问调试地址段 dseg；调试地址段 dseg 与内核模式下的 kseg3 地址段部分重叠，但 dseg 段的访问可以由软件打开或者关闭，因此在必要时调试模式下软件也可以访问完整的 kseg3 地址段。

图 3-2 中的每一个地址段或者是具有 mapped 属性或者具有 unmapped 属性，具体请看后续各工作模式的详细描述；在虚实地址转换时，unmapped 地址段的虚拟地址不会使用 TLB，而是采用一种直接映射方式；而 mapped 地址段的虚拟地址则通过 TLB 进行虚实地址映射，转换的单位是页，转换的信息包括物理页号、是否 cacheable 以及保护属性等。

3.6.1.2 用户模式

在用户模式下，软件只能访问 2GB 大小的逻辑地址空间，该地址空间称为 useg（用户段，user segment），起始于地址 0，结束于地址 0x7FFF_FFFF，如图 3-3 所示。任何对 useg 空间以外逻辑地址的访问都将引起一个地址错误例外。

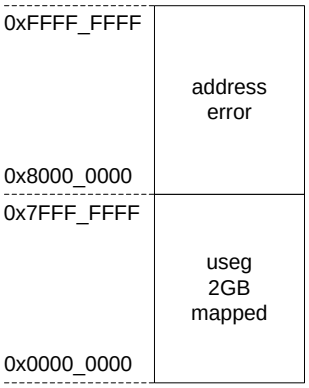


图 3-3 用户模式下的地址空间

当 CP0 寄存器中的 Status 寄存器同时包含如下状态位时，龙芯处理器将处于用户模式：

- KSU=10_2
- EXL=0
- ERL=0

在用户模式下，除了上述 Status 寄存器的三个状态位，Debug 寄存器中的 DM 位也必须为 0.

useg 地址段的属性如表 3-1 所示，表明用户模式下软件可访问的虚拟地址的最高位必须为 0，即只能访问整个 4GB 逻辑地址空间中的低 2GB 空间，任何对高 2GB 地址空间的访问都将引起一个地址错误例外。对 useg 空间的访问将通过 TLB 进行虚实地址转换，在转换之前，虚拟地址将与 8 位的 ASID 域合并形成一个唯一的地址，该地址在经过 TLB 进行虚实地址转换之后产生对应的物理地址。在进行虚实地址转换时，TLB 表项中除了对应的物理页号信息，还包含了当前页是否使用 cache 的信息，即当前访问是直接访问外部存储器空间还是访问 cache。

表 3-1 用户模式下地址段属性

地址位	Status 寄存器			Debug 寄存器	地址段名称	地址范围	地址段大小
	EXL	ERL	KSU	DM			
A(31)=0	0	0	10_2	0	useg	0x000_0000 ~0x7FFF_FFFF	2GB

3.6.1.3 监管模式

在分层操作系统中，操作系统内核运行于内核模式下，而操作系统其余部分则运行于监管模式下。当 CP0 寄存器中的 Status 寄存器同时包含如下状态位时，龙芯处理器将处于监管模式：

- KSU=01_2
- EXL=0
- ERL=0

在监管模式下，除了上述 Status 寄存器的三个状态位，Debug 寄存器中的 DM 位也必须为 0.

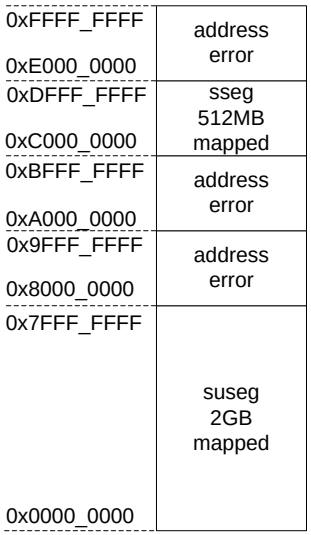


图 3-4 监管模式下的地址空间

监管模式下软件可访问的逻辑地址空间如图 3-4 所示，包括 suseg 与 sseg 两个地址段，地址段属性如表 3-2 所示。监管模式下的 suseg 地址段对应于用户模式下的 useg 地址段，地址范围为 0x0-0x7FFF_FFFF，即整个 4GB 逻辑地址空间中的低 2GB 空间。监管模式下的 sseg 地址段大小为 512MB，即 32 位虚拟地址中最高三位地址为 110_2。对 suseg 与 sseg 空间的访问都将通过 TLB 进行虚实地址转换，在转换之前，虚拟地址将与 8 位的 ASID 域合并形成一个唯一的地址，该地址在经过 TLB 进行虚实地址转换之后产生对应的物理地址。在进行虚实地址转换时，TLB 表项中除了对应的物理页号信息，还包含了当前页是否使用 cache 的信息，即当前访问是直接访问外部存储器空间还是访问 cache。

表 3-2 监管模式下地址段属性

地址位	Status 寄存器			Debug 寄存器	地址段名称	地址范围	地址段大小
	EXL	ERL	KSU	DM			
A(31)=0	0	0	01_2	0	suseg	0x000_0000 ~0x7FFF_FFFF	2GB
A(31:29) = 110_2					sseg	0xC000_0000 ~0xDFFF_FFFF	512MB

3.6.1.4 内核模式

当 CP0 寄存器中的 Status 寄存器包含如下任意一个或者多个状态位时，龙芯处理器将处于内核模式：

- KSU=00_2
- EXL=1
- ERL=1

在内核模式下，除了上述三个状态位，Debug 寄存器中的 DM 位也必须为 0.

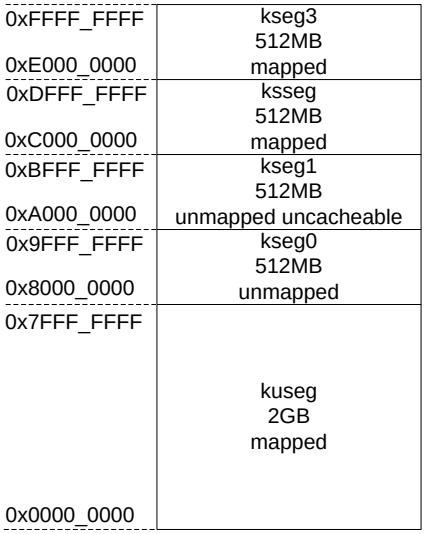


图 3-5 内核模式下的地址空间

当龙芯处理器发生一个例外并且该例外不是调试例外的时候，Status 寄存器的 EXL 位或者 ERL 位将会被置 1，处理器进入到内核状态开始进行例外处理；当完成例外处理的时候，可执行一条例外返回指令 ERET，处理器将跳转到此前发生例外的指令重新开始执行正常程序流程，同时将 EXL 位或者 ERL 位清零，此时处理器返回到发生例外之前的工作模式。

龙芯处理器在内核模式下可以访问整个 4GB 逻辑地址空间，如图 3-5 所示；4GB 逻辑地址空间划分为不同的地址段，包括 kuseg、kseg0、kseg1、ksseg 以及 kseg3，不同地址段的属性如表 3-3 所示。

表 3-3 内核模式下地址段属性

地址位	Status 寄存器			Debug 寄存器	地址段名称	地址范围	地址段大小
	EXL	ERL	KSU	DM			
A(31)=0	EXL=1 或者 ERL=1 或者 KSU=00_2			0	kuseg	0x000_0000 ~0x7FFF_FFFF	2GB
A(31:29) = 100_2					kseg0	0x8000_0000 ~0x9FFF_FFFF	512MB
A(31:29) = 101_2					kseg1	0xA000_0000 ~0xBFFF_FFFF	512MB
A(31:29) = 110_2					ksseg	0xC000_0000 ~0xDFFF_FFFF	512MB
A(31:29) = 111_2					kseg3	0xE000_0000 ~0xFFFF_FFFF	512MB

内核模式下 kuseg 地址段对应于用户模式下的 useg 地址段与监管模式下的 suseg 地址段，地址范围为 0x0-0x7FFF_FFFF，即整个 4GB 逻辑地址空间中的低 2GB 空间。正常情况下，内核模式下对 kuseg 段的访问将通过 TLB 进行虚实地址转换，在转换之前，虚拟地址将与 8 位的 ASID 域合并形成一个唯一的地址，该地址在经过 TLB 进行虚实地址转换之后产生对应的物理地址。在进行虚实地址转换时，TLB 表项中除了对应的物理页号信息，还包含了当前页是否使用 cache 的信息，即当前访问是直接访问外部存储器空间还是访问 cache。

内核模式下 kseg0 地址段的范围为 0x8000_0000-0x9FFF_FFFF，大小为 512MB，对应于 32 位虚拟地址中最高三位地址为 100_2；kseg0 地址段具有 unmapped 属性，因此对 kseg0 段的访问不会经过 TLB 进行虚实地址转换，而是从虚拟地址直接减去 0x8000_0000 作为物理地址。Config 寄存器中的 K0 域控制 kseg0 段是否使用 cache，即当前访问是直接访问外部存储器空间还是访问 cache。

内核模式下 kseg1 地址段的范围为 0xA000_0000-0xBFFF_FFFF，大小为 512MB，对应于 32 位虚拟地址中最高三位为 101_2；kseg1 地址段具有 unmapped 和 uncacheable 属性，因此对 kseg1 段的访问不会经过 TLB 进行虚实地址转换，而是从虚拟地址直接减去 0xA000_0000 作为物理地址，该物理地址将用于直接访问外部存储器空间。

内核模式下 ksseg 地址段的范围为 0xC000_0000-0xDFFF_FFFF，大小为 512MB，对应于 32 位虚拟地址中最高三位为 110_2；kseg3 地址段的范围为 0xE000_0000-0xFFFF_FFFF，大小为 512MB，对应于 32 位虚拟地址空间中最高三位为 111_2；对 ksseg 段与 kseg3 段的访问将通过 TLB 进行虚实地址转换，在转换之前，虚拟地址将与 8 位的 ASID 域合并形成一个唯一的地址，该地址在经过 TLB 进行虚实地址转换之后产生对应的物理地址。在进行虚实地址转换时，TLB 表项中除了对应的物理页号信息，还包含了当前页是否使用 cache 的信息，即当前访问是直接访问外部存储器空间还是访问 cache。

3.6.1.5 调试模式

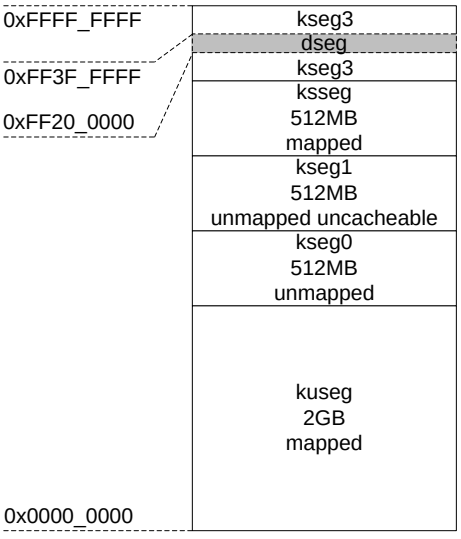


图 3-6 调试模式下的地址空间

当发生调试例外时，Debug 寄存器中的 DM 位为 1，龙芯处理器进入到调试模式。除了额外可以访问调试地址段 dseg 之外，调试模式与内核模式具有相同的逻辑地址空间及访问权限；调试地址段 dseg 的地址范围为 0xFF20_0000-0xFF3F_FFFF，大小为 2MB，与 kseg3 地址段部分重叠，如图 3-6 所示。

dseg 地址段进一步划分为 dmseg 与 drseg 两个子地址段，如表 3-4 所示；dmseg 地址段的范围是 0xFF20_0000-0xFF2F_FFFF，用于调试程序的程序数据存储空间；drseg 地址段的范围是 0xFF30_0000-0xFF3F_FFFF，调试寄存器将映射到 drseg 地址段，通过访问 drseg 地址段空间就可以访问调试寄存器。

表 3-4 dseg 地址段的划分与属性

地址段	子地址段	逻辑地址空间	物理地址空间	cache 属性
dseg	dmseg	0xFF20_0000~ 0xFF2F_FFFF	dmseg 映射到调试主机 EJTAG 调试空间，drseg 映射为断点寄存器	uncacheable
	drseg	0xFF30_0000~ 0xFF3F_FFFF		

dseg 地址段与 kseg3 地址段部分重叠，根据软件配置的不同，当访问这一段地址空间时，硬件上既可以访问 dseg 段地址空间，也可以访问 kseg3 段地址空间。处理器访问 drseg 地址段的条件组合如表 3-5 所示，映射到 drseg 地址段的寄存器包括调试控制寄存器(DCR, Debug control register)与硬件断点寄存器，只允许软件以字的方式访问这些映射寄存器；如果访问 drseg 地址段中未映射到寄存器的空闲地址，则结果不可预知，软件应该避免这种情况；有关 DCR 寄存器以及硬件断点寄存器的更多信息，请参考 EJTAG 规范。处理器访问 dmseg 地址段的条件组合如表 3-6 所示，其中应当尽量避免 ProbEN 控制位为 0 时访问 dmseg 地址段的情况，如果出现这种情况，则对 dmseg 地址段的任何访问都不能正常完成，处理器将处于停顿状态，直到 ProbEN 控制位重新变为 1。

表 3-5 drseg 地址段的访问

操作	Debug 寄存器的 LSNW 位	访问目标
Load/Store	1	kseg3
Fetch	Don’t care	drseg
Load/Store	0	

表 3-6 dmseg 地址段的访问

操作	DCR 寄存器中的 ProbEN 位	Debug 寄存器中的 LSNW 位	访问目标
Load/Store	Don’t care	1	kseg3
Fetch	1	Don’t care	dmseg
Load/Store	1	0	
Fetch	0	Don’t care	尽量避免
Load/Store	0	0	

3.6.2 TLB

32 位龙芯处理器支持全功能 MMU 并通过片内 TLB 进行虚实地址转换，其中片内 TLB 包括 32 个表项，采用全相联映射结构，取指请求与数据访问均通过这 32 项 TLB 进行虚实地址转换。每一个 TLB 表项对应于奇数页与偶数页两个子表项，因此 32 项 TLB 一共可以实现 64 个操作系统页的虚实地址转换。在一次虚实地址转换过程中，虚拟地址（即逻辑地址，下同）与 EntryHI 寄存器中的 ASID 域进行合并扩展，合并扩展后的值用于并行查询 TLB 中的所有表项，如果与某一个 TLB 表项匹配，则表明 TLB 命中，该 TLB 表项中的物理地址即为虚拟地址对应的物理地址。

32 位龙芯处理器可以支持不同页大小的虚实地址转换，页的大小可以从 4KB 到 16MB，每一个数值都是前一个数值的 4 倍，因此页大小包括 4KB、16KB、64KB、256KB、1MB、4MB 与 16MB 等情况。

当一个虚拟地址在 TLB 中找到一个匹配项时，即 TLB 命中，此时从 TLB 中读出对应的物理页号，并与偏移地址合并形成最终的物理地址。当一个虚拟地址在 TLB 中不能找到匹配项时，即发生 TLB 缺失，此时产生一个 TLB 缺失例外，软件将从内存中把页表填写到 TLB 中。在软件写入 TLB 表项时，既可以随机写入到一个表项位置，也可以通过硬件机制写入到一个特定的 TLB 表项位置。

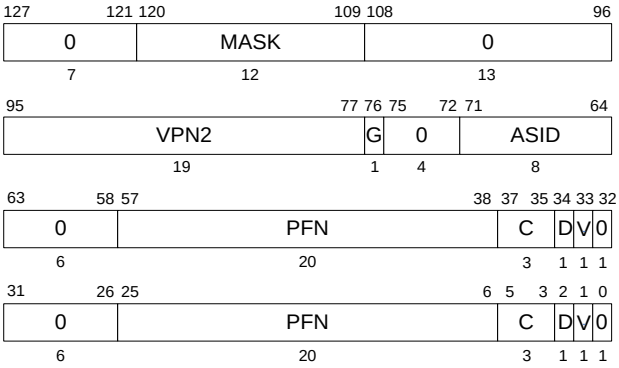


图 3-7 TLB 表项格式

表 3-7 TLB 表项位域说明

位域	位宽	功能描述
MASK	12	页屏蔽位；在虚实地址转换过程中，根据页的大小，页屏蔽位用于屏蔽 VPN2 域中的若干位，使之不参加判断 TLB 命中与否的地址比较，页屏蔽位与页大小的对应关系如下： 页屏蔽位 | 页大小 0000_0000_0000 | 4KB 0000_0000_0011 | 16KB 0000_0000_1111 | 64KB 0000_0011_1111 | 256KB 0000_1111_1111 | 1MB 0011_1111_1111 | 4MB 1111_1111_1111 | 16MB
VPN2	19	虚拟页号除以 2 之后的值；由于在 TLB 表项中一个虚拟页对应于一对奇偶物理页，因此虚拟页号需要除以 2；VPN2[18:10]总是会参加判断 TLB 命中与否的地址比较，而 VPN2[11:0]则根据 MASK 域中指定的页大小来决定是否参加地址比较。
G	1	全局位；当 G 域被设置为 1 时，表明当天 TLB 表项适用于所有的进程，因而在判断 TLB 是否命中时 ASID 域不再参与地址比较。
ASID	8	地址空间标志符；用于标志当前 TLB 表项所属的进程。
PFN	20	物理页号；定义了物理地址的[31:12]位，当页大小大于 4KB 时，该域中实际只有部分位会被使用到。
C	3	cache 属性；C 域用于标志当前页的 cache 属性，在 32 位龙芯处理器中，C 域值为 011_2 表示当前页具有 cacheable 属性，即访问当前页将使用 cache，

位域	位宽	功能描述
		所有其他值均表示具有 uncacheable 属性，即不使用 cache。
D	1	Dirty 位；D 域表示当前页是否可写，为 1 时允许对当前页进行写操作，为 0 时不允许对当前页进行写操作，否则将引起一个 TLB modified 例外。
V	1	Valid 位；V 域表示当前 TLB 表项是否有效，当为 1 时，访问当前页可正常进行虚实地址转换，当为 0 时，访问当前页将引起一个 TLB invalid 例外。

TLB 表项的格式如图 3-7 所示，表项中每个域的说明如表 3-7 所示。软件通过 TLBWI 或者 TLBWR 指令向 TLB 中写入一个表项，在写入之前，需要先配置若干 CP0 寄存器以更新要写入的信息，这些 CP0 寄存器及其位域包括：

- PageMask 寄存器中的 MASK 域
- EntryHi 寄存器中的 VPN2 与 ASID 域
- EntryLo0 寄存器中的 PFN、C、D、V 和 G 域
- EntryLo1 寄存器中的 PFN、C、D、V 和 G 域

在上述 CP0 寄存器位域中，G 域同时存在于 EntryLo0 寄存器与 EntryLo1 寄存器中，写入到 TLB 表项中的 G 域值是两个寄存器中的 G 域值相“与”的结果。ASID 域用于减少上下文切换时引起的 TLB 抖动；在虚实地址转换时，EntryHi 寄存器中的 ASID 域与 TLB 表项中的 ASID 域将进行比较；通过 ASID 域，使得不同的进程信息可以共存于 TLB 与 cache 中。

3.6.3 虚实地址转换

在虚实地址转换过程中，如果虚拟地址的虚拟页号与 TLB 表项的 VPN2 域互相匹配，并且满足如下两个条件之一，则称为一次 TLB 命中：

- TLB 表项中奇数页与偶数页的 G 域均为 1；
- EntryHi 寄存器中的 ASID 域与 TLB 表项的 ASID 域相等；

如果不满足以上匹配条件，则龙芯处理器发生一次 TLB 缺失例外，软件将从内存中读出对应的页表项写入 TLB。

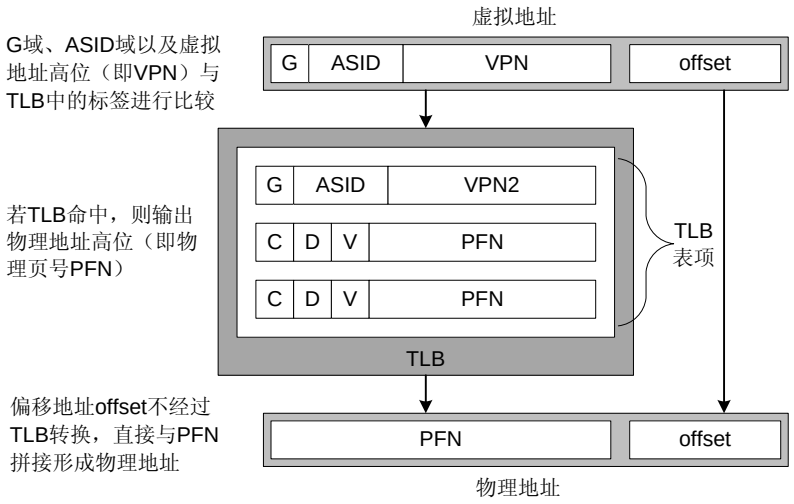


图 3-8 虚实地址转换过程

虚实地址转换过程如图 3-8 所示，虚拟地址与 8 位 ASID 值进行合并扩展，扩展之后的值用于判断是否 TLB 命中；8 位 ASID 是当前进程的标志信息，存储于 EntryHi 寄存器中.如果一个虚拟地址在 TLB 中命中，则读出对应 TLB 表项中的物理页号 PFN，并与虚拟地址中的偏

移地址拼接形成物理地址；偏移地址代表了当前请求在一页中的具体位置，偏移地址不需要进行虚实地址转换。

图 3-9 中表示了龙芯处理器在页大小为 4KB 与 16MB 时的虚实地址转换过程。偏移地址的长度由页大小决定，页大小为 4KB 时偏移地址为 12 位，虚拟地址中剩余的 20 位地址为虚拟页号（VPN）；页大小为 16MB 时偏移地址为 24 位，虚拟页号为 8 位。

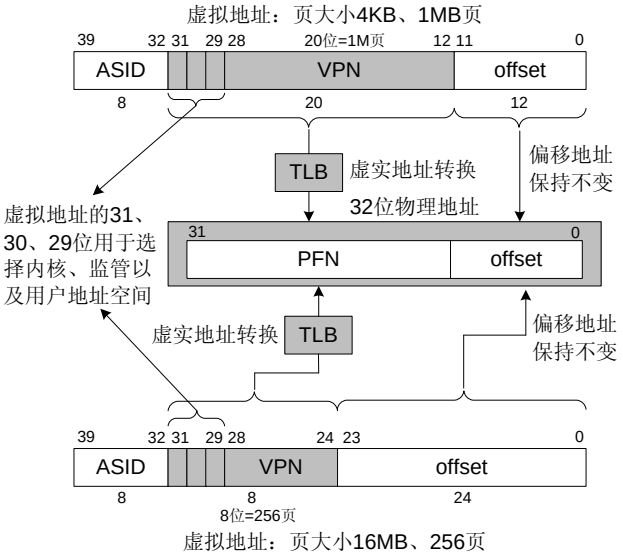


图 3-9 龙芯处理器 32 位虚实地址转换过程图示

对于一个虚拟地址访问请求，如果在 TLB 中命中一个有效的 TLB 表项，则转换产生有效的物理地址；如果在 TLB 中不命中，则产生一个 TLB 缺失例外；如果 TLB 命中但对应的 TLB 表项无效，则产生一个 TLB 无效例外；图 3-10 给出了虚实地址转换及产生例外的流程。

在软件向 TLB 中写入一个新的 TLB 表项时，既可以通过 Index 寄存器将 TLB 表项写入到一个确定的位置，也可以通过 Random 寄存器将 TLB 表项写入到一个随机的位置；对于随机写入的方式，还可以通过 Wired 寄存器使得某些 TLB 表项不会被替换。与此同时，根据实际情况的需要，软件还可以通过 PageMask 寄存器设置 TLB 表项中页的大小，页大小的范围为 4KB 到 16MB。有关与存储管理相关 CP0 寄存器的详细描述，请看“CP0 协处理器”一章。

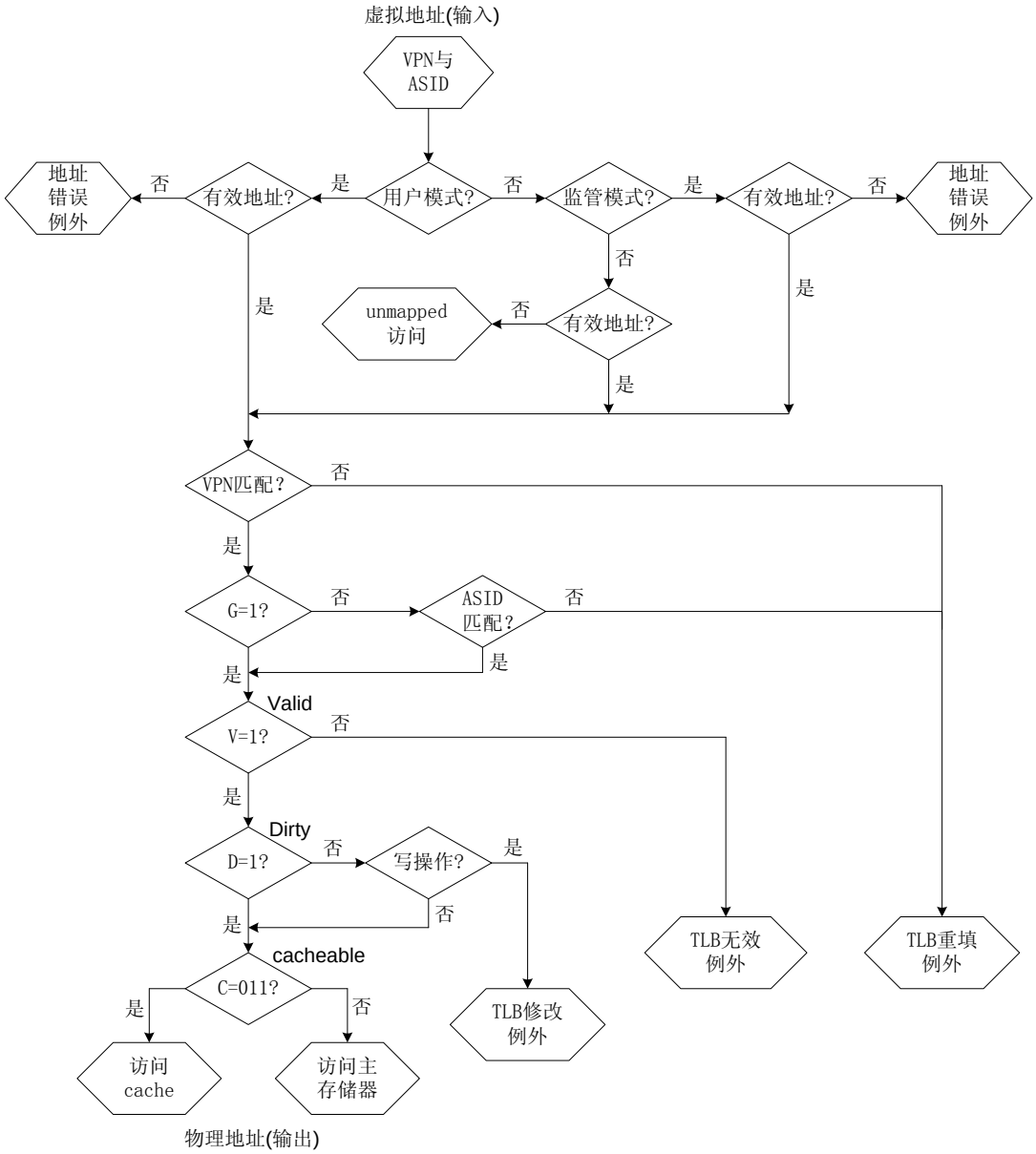


图 3-10 TLB 虚实地址转换流程

3.7 高速缓存

3.7.1 概述

32 位龙芯处理器具有独立的 16KB 片上 L1 指令 cache 与 16KB 片上 L1 数据 cache，分别组织为四路组相联结构，每一路 cache 大小为 4KB，由 128 个 cache 块（通常也称为 cache 行或者 cacheline）组成，每个 cache 块大小为 32 个字节。龙芯处理器采用了虚拟地址作为 cache 索引与物理地址作为 cache 标签，因而可以并行进行虚实地址转换与 cache 访问，提高了 cache 的效率。为了尽可能提高流水线的效率，龙芯处理器采用了非阻塞（non-blocking）cache 与关键字优先等高效 cache 设计技术。

在运行过程中，如果取指请求与数据请求访问的是 uncacheable 存储器区域，则该请求

将旁路 cache，并通过龙芯处理器的 AXI 接口向外发出请求；如果访问的是 cacheable 区域，则直接访问 cache 进行读写操作，并在 cache 不命中时通过 AXI 接口向主存储器发出一个长度为 8 个字的突发访问请求。

3.7.2 组织结构

32 位龙芯处理器的指令 cache 与数据 cache 大小均为 16KB，分别采用四路组相联结构，如图 3-11 所示，其中每一路 cache 包括 128 个索引项，每个索引项对应于一个大小为 8 个字的 cache 块。在取指请求与数据请求的访问过程中，根据索引（Index）读取相应的标签（Tag）和数据（Data），读出的标签值用于和虚实地址转换产生的物理地址进行比较，从而确定哪一路 cache 包含了需要的数据。

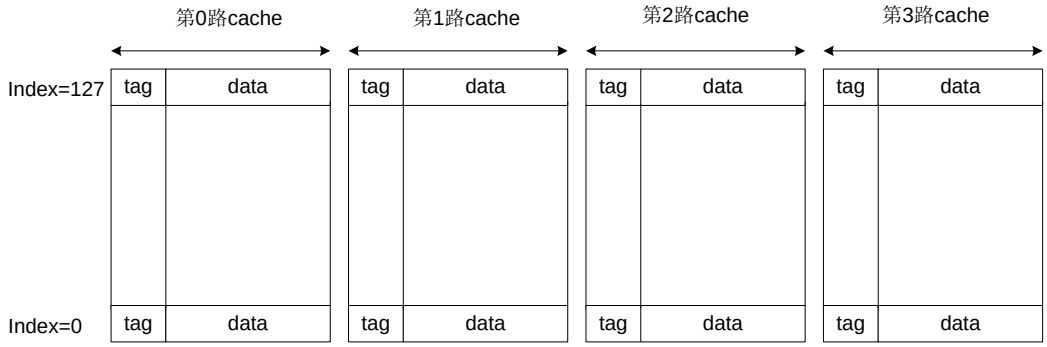


图 3-11 四路组相联 cache 组织结构

如图 3-11 所示，指令 cache 与数据 cache 具有相同的组织结构，每一路 cache 包括 128 个索引项，每一个索引项包含一个标签项与一个 8 个字的数据项，但指令 cache 与数据 cache 的标签项内容有所不同，如图 3-12 与图 3-13 所示，指令 cache 的标签项包含有 20 位的高位物理地址和 cache 块状态位，数据 cache 的标签项包含有 20 位的高位物理地址、cache 块状态位以及 cache 块是否为“脏”的 W 标志位。

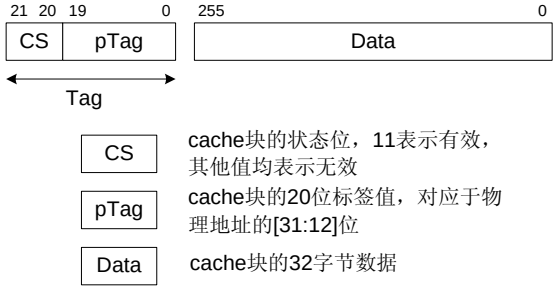


图 3-12 指令 cache 块的组成

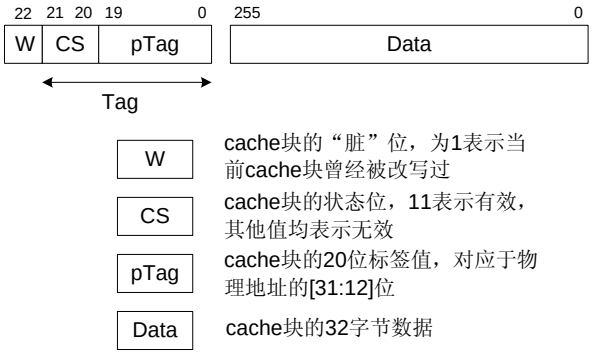


图 3-13 数据 cache 块的组成

在一次 cache 访问过程中，虚拟地址的低 12 位用作 cache 的索引，其中[11:5]位用于索引每一路 cache 中的 128 个 cache 行，[4:2]位用于索引每一个 cache 块中的 8 个字，如图 3-14 所示；虚拟地址的高位用于通过 MMU 进行虚实地址转换，转换得到的物理地址与 cache 标签中的 pTag 进行比较，如果存在某一路的 pTag 与其匹配，并且 cache 标签的状态位有效，则称为一次“cache 命中（cache hit）”，从数据项中得到需要的数据，如果四路 cache 中任何一路的 pTag 都不与其匹配，则称为一次“cache 缺失（cache miss）”，将向内存发起一个 8 个字的突发请求。

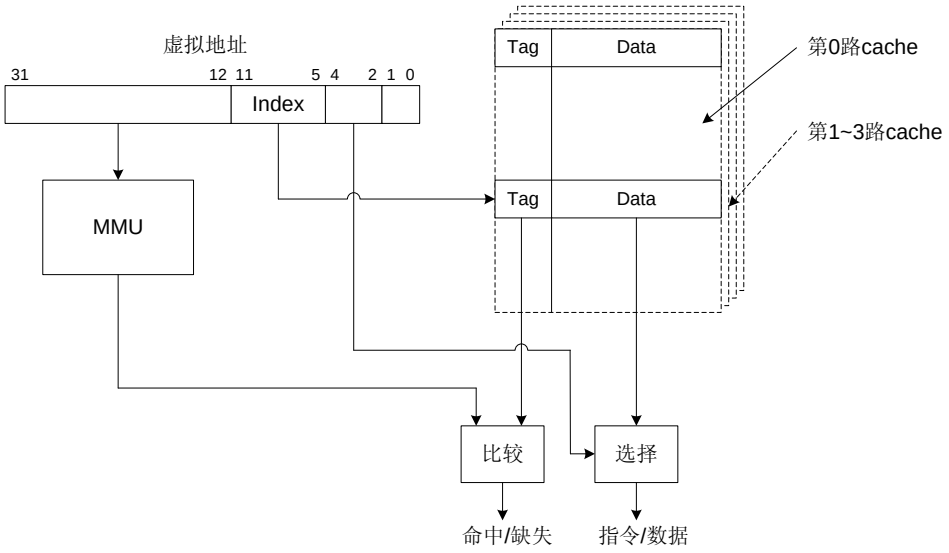


图 3-14 cache 访问过程

3.7.3 替换策略

龙芯处理器的 cache 采用了随机替换策略，当一个请求在 cache 中不命中（cache 缺失）时，将从四路 cache 中随机选择一路进行替换，并把从内存读入的新 cache 块写入到被替换的位置。

龙芯处理器的 cache 是 write-back cache，因此不管是取指请求、读数据请求还是写数据请求引起 cache 缺失，都将首先从内存中读入对应的 cache 块到 cache 中，然后再进行取指或者数据读写操作。

在数据 cache 发生替换时，如果被替换的 cache 块状态有效（即 CS=0x3）并且 W 位为 1，则表明该 cache 块的数据曾经被改写过，当前 cache 块的数据是最新的数据，因此需要将当前被替换的 cache 块数据写回到内存中，否则直接丢弃即可。

3.7.4 cache 操作

32 位龙芯处理器提供了六条 cache 指令对 cache 进行各种细粒度的操作，包括使某个 cache 块无效、读出 cache 标签、写入 cache 标签以及使命中的 cache 行无效等。指令 cache 与数据 cache 均为四路组相联结构，使用访问地址的最低两位来区分当前 cache 指令是对哪一路 cache 进行操作。在六条 cache 中，一条用于指令 cache，五条用于数据 cache，具体情况如下表 3-8 所示。

表 3-8 龙芯处理器的 cache 指令操作

cache 指令的[20:16]位	cache 操作	目标 cache
0x0	Index Invalidate	指令 cache
0x1	Index Writeback Invalidate	数据 cache
0x5	Index Load Tag	数据 cache
0x9	Index Store Tag	数据 cache
0x11	Hit Invalidate	数据 cache
0x15	Hit Writeback Invalidate	数据 cache

3.8 例外处理

32 位龙芯处理器具有多个例外来源，包括 TLB 缺失、整数溢出、IO 中断以及系统调用等，当处理器检测到任意一个例外时，正常的指令执行流程将被打断，处理器进入内核模式进行例外处理。

处理器进入内核模式之后，将自动禁止中断，并强制跳转到一个固定的入口地址开始执行例外处理程序；例外处理程序首先保存发生例外时处理器的上下文，包括 PC 值、当前工作模式、中断状态（使能或者禁止）等，当完成例外处理时，处理器将恢复为已保存的上下文现场。

当发生一个例外时，CP0 寄存器中的 EPC(Exception Program Counter)寄存器值将被更新为例外处理之后继续执行程序的第一条指令 PC 值。如果是一条普通指令引起了例外，则 EPC 保存的是该指令的 PC 值；如果是一条延时槽指令引起了例外，则 EPC 保存的是延时槽指令之前的转移指令 PC 值。为了区分这两种情况，例外处理程序可以读取 Cause 寄存器中的 BD 位进行判断。

3.8.1 精确例外

当发生一个例外时，当前指令以及流水线中所有后续指令都将被取消，不会产生有效的执行结果并影响到处理器状态，相应地，所有与当前指令有关的流水线停顿以及潜在例外都不会再产生作用。

在一条指令从取指到提交的流水线执行过程中，不管在哪个流水阶段发生了例外，该指令以及后续指令都将会被取消，并且在该指令到达提交阶段之后，该指令的例外信息将更新例外处理有关的 CP0 寄存器，包括例外原因、当前指令 PC 值等，此后处理器跳转到例外入口地址开始执行例外处理程序。

通过以上实现方式，当一条指令发生例外时，该指令之前的所有指令都可以完成执行，而该指令之后的所有指令都不会完成执行，因此 EPC 寄存器（或者 ErrorEPC 寄存器及 DEPC 寄存器）中保存的信息可以在例外处理之后使得程序恢复正常执行。当多条指令发生例外时，

这种方式也保证了发生例外的顺序与指令执行的顺序一致，流水线中后面的指令不会先于前面的指令发生例外。根据以上描述，由于龙芯处理器中例外处理精确对应到发生例外的指令，因此是一种精确的例外处理方式。

3.8.2 例外优先级

在龙芯处理器中执行指令的过程中，根据上下文情况，可能发生多种例外，表 3-9 中根据优先级列举了龙芯处理器中所有可能的例外；当同时发生多个例外时，具有最高优先级的例外将最先得到响应。

表 3-9 龙芯处理器例外优先级

优先级	例外助记符	例外描述
高	Reset	系统复位
	DSS	片上调试例外：单步执行
	DINT	片上调试例外：调试中断
	Int	中断
	DIB	片上调试例外：指令断点
	AdEL	取指地址错误
	TLBL	取指 TLB 缺失或取指 TLB 无效
	DBp	片上调试例外：软件调试断点
	Sys	系统调用
	Bp	软件断点
	CpU	协处理器不可用
	RI	非法指令
	Ov	整数溢出
	Tr	系统自陷
	DDBL/DDBS	片上调试例外：不带有数据比较的数据硬件断点，或者带有数据比较的 store 指令数据硬件断点
	AdEL	load 指令数据访问地址错误
	AdES	store 指令数据访问地址错误
	TLBL	load 操作引起的 TLB 缺失或者 TLB 无效
	TLBS	store 操作引起的 TLB 缺失或者 TLB 无效
	TLBM	TLB 修改错误
低	DDBL	片上调试例外：带有数据比较的 load 指令数据硬件断点

3.8.3 例外处理入口

在龙芯处理器中，复位作为一种例外进行处理，并且例外处理的入口地址固定为 0xBFC0_0000。对于调试例外，根据 ECR 寄存器中 ProbTrap 位是 0 还是 1，例外处理的入口地址分别固定为 0xBFC0_0480 或者 0xFF20_0200。除此之外，所有其他例外的入口地址由一个向量基址与一个向量偏移值相加组合而得，如表 3-10 与表 3-11 所示；其中对于向量基址值，根据 Status 寄存器中 BEV 位的不同，该值也会存在不同。

表 3-10 龙芯处理器例外入口的向量基址

例外		BEV	
		0	1
Reset		0xBFC0_0000	
调试例外	ProbTrap=0	0xBFC0_0480	
	ProbTrap=1	0xFF20_0200	
其他		0x8000_0000	0xBFC0_0200

表 3-11 龙芯处理器例外入口的向量偏移量

例外	向量偏移量
Reset	无
TLB 重填	0x000
其他	0x180

3.8.4 通用例外处理

除了复位与调试例外，龙芯处理器的其他例外均具有相同的基本处理流程，主要包括如下几个方面：

- 如果 Status 寄存器的 EXL 位为 0，则 EPC 寄存器和 Cause 寄存器中的 BD 位根据当前指令是否为转移指令的延时槽指令进行相应更新；如果当前指令不是延时槽指令，则 EPC 寄存器更新为当前指令的 PC 值，BD 位更新为 0；如果当前指令是延时槽指令，则 EPC 寄存器更新为 PC-4，BD 位更新为 1。如果 Status 寄存器的 EXL 位为 1，则 EPC 寄存器与 Cause 寄存器的 BD 位均不会更新；
- 根据具体的例外类型，Cause 寄存器中的 CE 位与 ExcCode 位更新为相应的值；
- Status 寄存器的 EXL 位更新为 1；
- 处理器从适当的例外入口地址开始执行例外处理程序。

EPC 寄存器中的值代表了完成例外处理之后处理器恢复执行正常程序的第一条指令 PC，正常情况下例外处理程序不需要修改该寄存器的值。除非例外处理程序需要知道发生例外的指令是否是一条转移指令的延时槽指令，否则并不需要查询 Cause 寄存器的 BD 位。

另外，对于特定的某个例外，可能会保存某些额外的现场信息到其他相关 CP0 寄存器中，并由例外处理程序根据这些信息进行不同的处理。

通用例外处理的操作过程可表示如下：

```
If Status_EXL=0 then
    if instruction_in_branchdelayslot then
        EPC <- PC-4
        Cause_BD <- 1
    else
        EPC <- PC
        Cause_BD <- 0
    endif
    if exception_type = TLB_refill then
        vector_offset <- 0x000
    else
```

```
        vector_offset <- 0x180
    endif
    Cause_CE <- faulting_coprocessor_number
    Cause_exccode <- exception_type
    Cause_EXL <- 1
    if Status_BEV = 1 then
        PC <- 0xBFC0_0200 + vector_offset
    else
        PC <- 0x8000_0000 + vector_offset
    endif
```

3.8.5 复位例外处理

龙芯处理器把复位作为一种特殊的不可屏蔽例外进行处理。当对龙芯处理器进行复位时，处理器立刻恢复为初始状态，其中部分为确定的状态，可确保处理器正确开始初始化操作，其余部分则为未定义状态。龙芯处理器复位后的确定状态包括：

- Random 寄存器值初始化为 31；
- Wired 寄存器初始化为 0；
- Config 寄存器初始化为默认配置状态；
- Status 寄存器的 BEV 域与 ERL 域初始化为 1；
- PC 从 0xBFC0_0000 开始取指；

  复位例外处理的操作过程可表示如下：

```
Random <- 31
Wired <- 0
Config <- default_state
Status_BEV < 1
Status_ERL < 1
Status_NMI < 0
PC <- 0xBFC0_0000
```

3.8.6 调试例外处理

所有的调试例外都具有相同的基本处理流程，包括如下：

- 根据发生调试例外的指令是否为转移指令的延时槽指令，DEPC 寄存器和 Debug 寄存器中的 BD 位进行相应更新；如果当前指令不是延时槽指令，则 DEPC 寄存器更新为当前指令的 PC 值，DBD 位更新为 0；如果当前指令是延时槽指令，则 DEPC 寄存器更新为 PC-4，BD 位更新为 1。
- 根据调试例外的具体类型，更新 Debug 寄存器中的 DSS、DBP、DDBL、DDBS、DIB 以及 DINT 位；
- Debug 寄存器的 DM 位更新为 1；
- 从调试例外入口地址开始取指；

DEPC 寄存器中的值代表了完成调试例外处理之后处理器恢复执行正常程序的第一条指令 PC，正常情况下调试例外处理程序不需要修改该寄存器的值。除非调试例外处理程序需要知道发生例外的指令是否是一条转移指令的延时槽指令，否则并不需要查询 Debug 寄存

器的 DBD 位。

调试例外处理程序通过查询 Debug 寄存器中的 DSS、DBP、DDBL、DDBS、DIB 以及 DINT 位，可以知道具体发生了哪个调试例外。

在发生调试例外的时候，其他与调试无关的 CP0 寄存器都不会被更新。

调试例外处理的操作过程可表示如下：

```
if instruction_in_branchdelayslot then
    DEPC <- PC-4
    DebugDBD <- 1
else
    DEPC <- PC
    DebugDBD <- 0
endif
Debug[5:0] <- debug_exception_type
DebugDM <- 1
if ECRProbTrap = 1 then
    PC <- 0xFF20_0200
else
    PC <- 0xBFC0_0480
endif
```

3.9 CP0 协处理器

3.9.1 概述

龙芯处理器体系结构最多可以支持 4 个协处理器，分别为 CP0、CP1、CP2 和 CP3，但 GSC3281 芯片中的 32 位龙芯处理器只支持 CP0 协处理器。CP0 协处理器是一个必选的协处理器，主要用于各类系统控制功能，包括存储管理、例外处理、工作模式切换、cache 系统管理以及其他特权管理。

3.9.2 CP0 寄存器列表

CP0 协处理器包含一系列的 CP0 寄存器，可通过这些寄存器实现系统控制功能，处理器必须在内核模式下软件才可以访问 CP0 寄存器。龙芯处理器的 CP0 寄存器列表如表 3-12 所示。

表 3-12 龙芯处理器的 CP0 寄存器列表

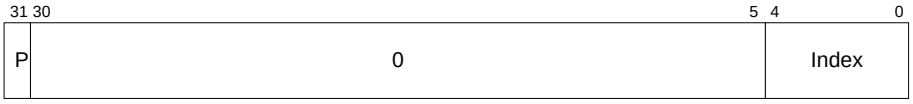
编号	寄存器名称	功能描述
0	Index	TLB 表项索引
1	Random	索引 TLB 表项的随机值
2	EntryLo0	TLB 表项的低半部分，对应于偶数号虚拟页
3	EntryLo1	TLB 表项的高半部分，对应于奇数号虚拟页
4	Context	存储器中的页表指针
5	PageMask	用于控制 TLB 表项中页大小的页屏蔽信息
6	Wired	用于控制固定 TLB 表项的多少

7	保留	保留
8	BadVaddr	最近一次发生地址相关例外的虚拟地址
9	Count	处理器内部计数器
10	EntryHi	TLB 表项的高半部分
11	Compare	比较产生时钟中断
12	Status	处理器状态与控制寄存器
13	Cause	保存最近一次例外的原因
14	EPC	保存最近一次例外的指令 PC 值
15	PRId	处理器标识符与版本号
16	Config	处理器配置寄存器
17-22	保留	保留
23	Debug	调试控制与调试例外状态寄存器
24	DEPC	保存最近一次调试例外的指令 PC 值
25-27	保留	保留
28	TagLo	cache 标签的低半部分
29	TagHi	cache 标签的高半部分，龙芯处理器中用作便签寄存器
30	ErrorEPC	保存最近一次错误的指令 PC 值，龙芯处理器中用作便签寄存器
31	DeSave	调试例外处理程序的便签寄存器

3.9.3 CP0 寄存器描述

3.9.3.1 CP0 寄存器 0：Index

Index 寄存器是 32 位可读写寄存器，包含了 TLB 探测是否成功的标志信息和指向 TLB 表项的索引值，处理器执行 TLBP 指令时将会更新探测是否成功的标志信息，执行 TLBP、TLBR 与 TLBWI 指令时将会使用到 TLB 表项索引值。由于龙芯处理器中包含了 32 项的 TLB 表项，因此 Index 寄存器中的索引位宽为 5 位。



位域	位	读写	功能描述	复位值
P	[31]	RO	探测失败标志；当 TLBP 指令不能在 TLB 中探测到一个匹配的表项时，该位将置 1。	未定义
0	[30:5]	-	未用域，读写均为 0。	0x0
Index	[4:0]	R/W	TLB 表项索引；TLBR、TLBWI 以及 TLBP 指令都将使用该域。	未定义

3.9.3.2 CP0 寄存器 1：Random

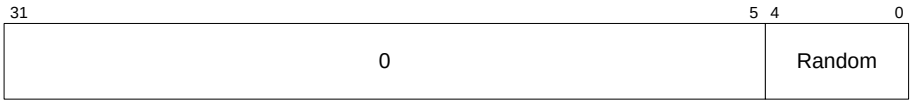
Random 寄存器是一个只读寄存器，其值在执行 TLBWR 指令时用于索引 TLB 表项。由于龙芯处理器包含了 32 项的 TLB 表项，因此 Random 寄存器位宽为 5 位。

Random 寄存器值在上限值与下限值之间随机变化，上限值与下限值分别为：

- 上限值：TLB 表项总数减 1，对于龙芯处理器而言，该值为 31。
- 下限值：操作系统专用 TLB 表项之后的第一个 TLB 表项，数值上等于 Wired 寄存器中的值。

Random 寄存器的值在每个时钟周期减 1，当减小到下限值时，则返回到上限值重新开始减 1 操作。

龙芯处理器复位时，Random 寄存器值初始化为上限值；当写 Wired 寄存器时，Random 寄存器也初始化为上限值。

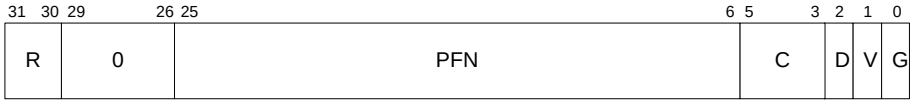


位域	位	读写	功能描述	复位值
0	[31:5]	-	未用域，读写均为 0。	0x0
Random	[4:0]	RO	TLB 表项随机索引。	未定义

3.9.3.3 CP0 寄存器 2/3：EntryLo0/1

EntryLo0 与 EntryLo1 寄存器是一对具有相同格式的可读写寄存器，在 TLB 读写时分别用于保存偶数物理页与奇数物理页的页表信息。在龙芯处理器中，TLBR 指令用于读 TLB，TLBWI 与 TLBWR 指令用于写 TLB。

在发生地址错误、TLB 无效、TLB 修改以及 TLB 重填例外之后，EntryLo0 与 EntryLo1 寄存器的值将处于未定义状态。



位域	位	读写	功能描述	复位值
Reserved	[31:30]	-	保留	0x0
0	[29:26]	R/W	这 4 位本应是 PFN 域的一部分，但由于龙芯处理器只支持 32 位物理地址，因此 PFN 域为 20 位宽，[29:26]位必须写为 0。	0x0
PFN	[25:6]	R/W	物理页号，对应于物理地址的[31:12]位。	未定义
C	5:3	R/W	cache 属性，当值为 011_2 时表示当前页具有 cacheable 属性，其余值均为 uncacheable 属性。	未定义
D	2	R/W	“脏”或者写使能位；该位为 1，表示当前页可写；该位为 0，表示当前页不可写，对当前页的写操作将引起 TLB 修改例外。	为定义

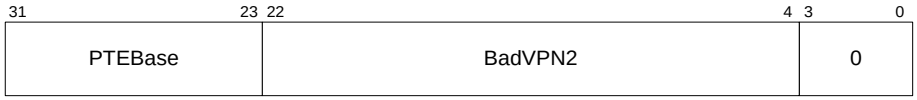
V	1	R/W	有效位，表明当前 TLB 表项是有效的；如果该位为 1，允许访问当前页；如果该位为 0，则访问当前页将引起一个 TLB 无效例外。	未定义
G	0	R/W	全局位；当 TLB 写时，EntryLo0 与 EntryLo1 寄存器中 G 位的“逻辑与”将作为 TLB 表项的 G 位值；如果 TLB 表项的 G 位为 1，则虚实地址转换时 ASID 值将不参与比较；当 TLB 读时，TLB 表项中的 G 位将同时更新到 EntryLo0 与 EntryLo1 寄存器的 G 位中。	未定义

3.9.3.4 CP0 寄存器 4：Context

Context 寄存器中包含了指向页表入口（page table entry, PTE）数组的指针，该数组是一个操作系统维护的用于虚实地址转换的数据结构。当发生 TLB 缺失例外时，操作系统从 PTE 数组中将缺失的虚实地址转换表项加载到 TLB 中。Contex 寄存器与 BadVaddr 寄存器具有部分相同的内容，但 Context 寄存器对内容进行了重新组织，使得操作系统可以直接访问存储器中的 8 字节 PTE。

当发生一个 TLB 例外时，包括 TLB 重填例外、TLB 无效例外以及 TLB 修改例外，虚拟地址中的[31:13]位将写入到 Context 寄存器中的 BadVPN2 域，而 PTEBase 域则由操作修改根据需要进行改写与使用。

在发生一个地址错误例外之后，Context 寄存器中的 BadVPN2 域将处于未定义状态。



位域	位	读写	功能描述	复位值
PTEBase	[31:23]	R/W	该域由操作系统根据需要改写与使用，通过写入合适的值，使得操作系统可以把 Context 寄存器值作为指向存储器中 PTE 数组的指针。	未定义
BadVPN2	[22:4]	RO	当发生 TLB 缺失时，将虚拟地址的[31:13]位写入到该域。	未定义
0	[3:0]	-	未用域，读写均为 0。	0x0

3.9.3.5 CP0 寄存器 5：PageMask

PageMask 寄存器是一个可读写寄存器，包含了设置页大小的页屏蔽信息，在读写 TLB 时将使用到该寄存器；PageMask 寄存器的页屏蔽信息与页大小的对应关系如下表寄存器描述所示，若使用了表中不存在的页屏蔽信息，则 TLB 行为将处于未定义状态。

位域	位	读写	功能描述	复位值

<table>
<tr><td>0</td><td>[31:25]</td><td>-</td><td>未用域，读写均为 0。</td><td>0x0</td></tr>
<tr><td>MASK</td><td>[24:13]</td><td>R/W</td><td>页屏蔽信息；当某位为 1 时，表示虚拟地址中的对应位将不参加虚实地址转换的地址比较；页屏蔽信息与页大小的对应关系如下表所示：

<table><tr><td>PageMask[24:13]</td><td>页大小</td></tr><tr><td>0000_0000_0000</td><td>4KB</td></tr><tr><td>0000_0000_0011</td><td>16KB</td></tr><tr><td>0000_0000_1111</td><td>64KB</td></tr><tr><td>0000_0011_1111</td><td>256KB</td></tr><tr><td>0000_1111_1111</td><td>1MB</td></tr><tr><td>0011_1111_1111</td><td>4MB</td></tr><tr><td>1111_1111_1111</td><td>16MB</td></tr></table></td><td>未定义</td></tr>
<tr><td>0</td><td>[12:0]</td><td>-</td><td>未用域，读写均为 0。</td><td>0x0</td></tr>
</table>

3.9.3.6 CP0 寄存器 6：Wired

Wired 寄存器指定了 TLB 中固定 TLB 表项与随机 TLB 表项之间的边界，如图 3-15 所示；与 Index 寄存器以及 Random 寄存器类似，Wired 寄存器的位宽为 5 位；编号值比 Wired 寄存器值小的 TLB 表项均属于固定 TLB 表项，不会被 TLBWR 指令改写，但可以被 TLBWI 指令改写。

龙芯处理器复位之后 Wired 寄存器值初始化为 0；写 Wired 寄存器时，Random 寄存器自动被初始化为 31。

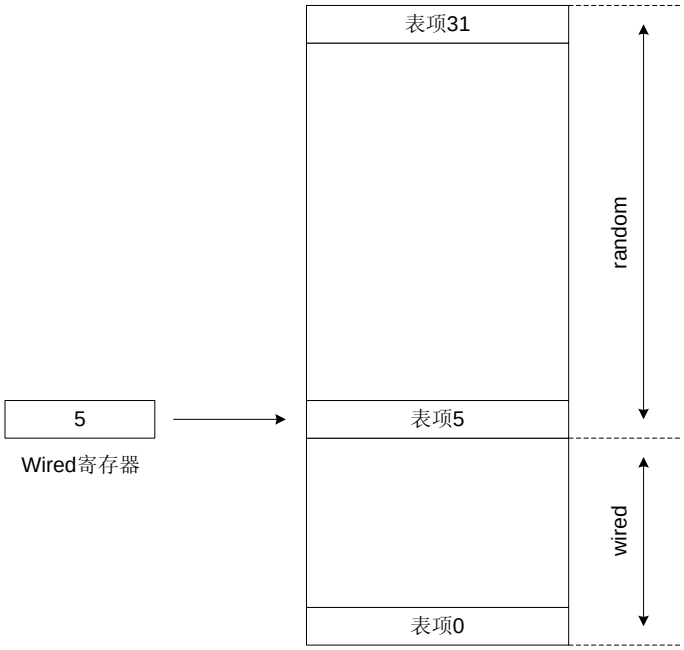
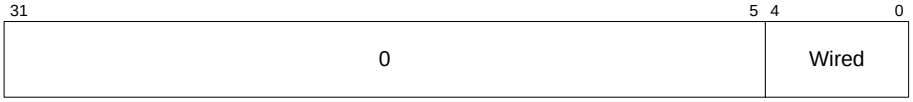


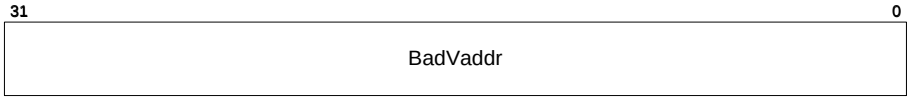
图 3-15 TLB 中的固定与随机表项



位域	位	读写	功能描述	复位值
0	[31:5]	-	未用域，读写均为 0。	0x0
Wired	[4:0]	R/W	TLB 固定表项边界值。	0x0

3.9.3.7 CP0 寄存器 7：BadVaddr

BadVaddr 寄存器是一个只读寄存器，保存了最近一次发生地址错误例外、TLB 重填例外、TLB 无效例外以及 TLB 修改例外的虚拟地址。

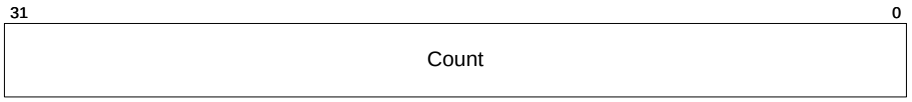


位域	位	读写	功能描述	复位值
BadVaddr	[31:0]	R/W	最近一次发生地址错误例外、TLB 重填例外、TLB 无效例外以及 TLB 修改例外的虚拟地址。	未定义

3.9.3.8 CP0 寄存器 9：Count

Count 寄存器的作用类似于一个计数器，每隔两个时钟周期进行一次加 1 操作，这种固定速率的加 1 操作不会受到指令执行过程以及流水线状态的影响。

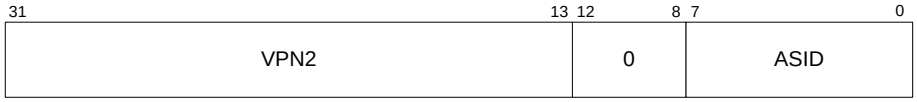
在调试模式下，Debug 寄存器中 CountDM 位的配置值决定 Count 寄存器是否继续执行加 1 操作。



位域	位	读写	功能描述	复位值
Count	[31:0]	R/W	32 位计数器，每隔两个时钟周期执行一次加 1 操作。	未定义

3.9.3.9 CP0 寄存器 10：EntryHi

EntryHi 寄存器中包含了 TLB 读、写以及访问操作中用到的虚拟地址信息。当发生一个 TLB 例外时，包括 TLB 重填例外、TLB 无效例外与 TLB 修改例外，虚拟地址中的[31:13]位将写入到 EntryHi 寄存器的 VPN2 域。软件把当前进程的地址空间标志写入到 EntryHi 寄存器中的 ASID 域，该值将在虚实地址转换过程中用于判断是否 TLB 命中。当发生地址错误例外时，EntryHi 寄存器中的 VPN2 域将处于未定义状态。

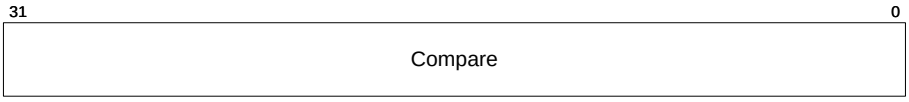


位域	位	读写	功能描述	复位值
VPN2	[31:13]	R/W	虚拟地址的[31:13]位（即虚拟页号除以 2）；VPN2 域在 TLB 例外或者 TLB 读操作时由硬件自动更新，在 TLB 写操作之前由软件写入。	未定义
0	[12:8]	-	未用域，读写均为 0。	0x0
ASID	[7:0]	R/W	地址空间标志；ASID 域在 TLB 读时由硬件自动更新，在 TLB 写操作之前由软件写入。	未定义

3.9.3.10 CP0 寄存器 11：Compare

Compare 寄存器与 Count 寄存器二者共同作用形成一个定时器，可用于产生时钟中断。Compare 寄存器值是一个恒定的值，除非软件进行改写，否则不会自行变化。

当 Count 寄存器值等于 Compare 寄存器值的时候，Cause 寄存器中的 IP[7]位被置 1，在中断使能的情况下将触发一个时钟中断。当写 Compare 寄存器时，Cause 寄存器的 IP[7]位将会自动清零。



位域	位	读写	功能描述	复位值
Compare	[31:0]	R/W	Count 寄存器的比较寄存器。	未定义

3.9.3.11 CP0 寄存器 12：Status

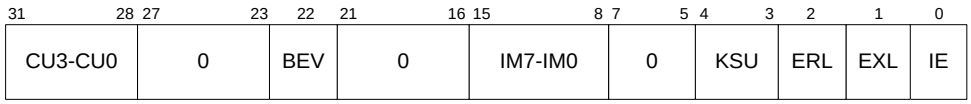
Status 寄存器是一个可读写寄存器，包含了龙芯处理器的工作模式与中断使能等状态与控制信息。

龙芯处理器支持 8 个中断，当满足如下所有条件时，中断被使能：

- Status 寄存器的中断使能位 IE=1
- Status 寄存器的例外级位 EXL=0
- Status 寄存器的错误级位 ERL=0
- Debug 寄存器的调试模式位 DM=0

在满足如上条件的情况下，当 Status 寄存器的中断屏蔽位 IM[7:0]与 Cause 寄存器的中断待定位 IP[7:0]中对应位均为 1 时，则触发一个中断。

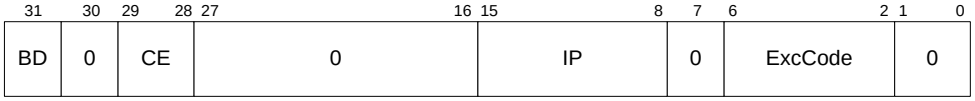
Status 寄存器的 CU3-CU0 位用于控制 4 个可能协处理器的可用性，当访问某个不可用的协处理器时，将触发一个协处理器不可用例外；不管 CU0 位是否为 1，在内核模式下 CP0 总是可用的。



位域	位	读写	功能描述	复位值
CU	[31:28]	R/W	控制 4 个可能协处理器是否可用。 0：协处理器不可用 1：协处理器可用 不管 CU0 位是否为 1，内核模式下 CP0 总是可用的；32 位龙芯处理器中未实现 CP1-CP3，因此 CU1-CU3 应当设置为 0，否则访问 CP1-3 的行为将不可预知。	未定义
0	[27:23]	-	未用域，读写均为 0。	0x0
BEV	22	R/W	控制例外入口地址。 0：正常模式入口 1：启动模式入口	0x1
0	[21:16]	-	未用域，读写均为 0。	0x0
IM	[15:8]	R/W	中断屏蔽位；控制每一个外部、内部或者软件中断是否使能；在中断被使能的情况下，如中断屏蔽位 IM[7:0]与 Cause 寄存器的中断待定位 IP[7:0]中对应位均为 1 时，则触发一个中断。 0：中断被屏蔽 1：中断未被屏蔽	未定义
0	[7:5]	-	未用域，读写均为 0。	0x0
KSU	[4:3]	R/W	处理器工作模式。 00：内核模式 01：监管模式 10：用户模式 11：未定义	0x0
ERL	[2]	R/W	错误级；当龙芯处理器复位时，该位置 1。 0：处理器未处于错误级 1：处理器处于错误级	0x1
EXL	[1]	R/W	例外级；当龙芯处理器发生除复位以外的例外时，该位置 1。 0：处理器未处于例外级 1：处理器处于例外级	未定义
IE	[0]	R/W	全局中断使能位； 0：禁止中断 1：使能中断	未定义

3.9.3.12 CP0 寄存器 13：Cause

Cause 寄存器描述了最近一次例外的例外原因，并可控制是否发生软件中断。



位域	位	读写	功能描述	复位值
BD	[31]	RO	最近一次例外是否发生在转移指令延时槽上； 0：不在转移指令延时槽上 1：在转移指令延时槽上	未定义
0	[30]	-	未用域，读写均为 0。	
CE	[29:28]	RO	发生协处理器不可用例外时的不可用协处理器编号。	未定义
0	[27:16]	-	未用域，读写均为 0。	0x0
IP	[15:10]	RO	标志是否有待处理的外部中断；1 表示有，0 表示无。 15：时钟中断 14：保留 13：保留 12：保留 11：保留 10：外部中断，中断控制器的中断输出连接到该位	未定义
IP	[9:8]	R/W	控制是否发生软件中断。1 表示发生，0 表示不发生。 9：软件中断 1 8：软件中断 0	未定义
ExcCode	[6:2]	RO	最近一次例外的例外原因，具体如表 1-10 所示。	未定义
0	[1:0]	-	未用域，读写均为 0。	0x0

表 3-13 Cause 寄存器 ExcCode 域描述

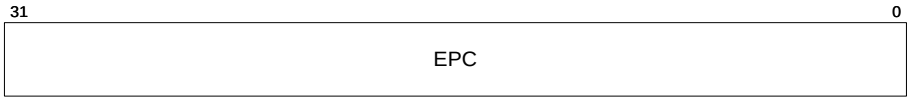
例外编号	例外助记符	例外描述
0	Int	中断
1	Mod	TLB 修改例外
2	TLBL	load 指令或者取指引起的 TLB 例外
3	TLBS	store 指令引起的 TLB 例外
4	AdEL	load 指令或者取指引起的地址错误例外
5	AdES	store 指令引起的地址错误例外
6-7	-	保留
8	Sys	系统调用例外
9	Bp	断点例外
10	RI	保留指令例外
11	CpU	协处理器不可用例外
12	Ov	整数溢出例外
13	Tr	自陷例外
14-31	-	保留

3.9.3.13 CP0 寄存器 14：EPC

EPC 寄存器是一个可读写寄存器，包含了完成例外处理之后恢复正常指令执行流程后的第一条指令的 PC 值。在发生例外的时候，根据当前指令执行情况，EPC 寄存器自动进行如下更新：

- 如果发生例外的指令不是转移指令延时槽指令，则 EPC 寄存器更新为发生例外的指令的 PC 值。
- 如果发生例外的指令是转移指令延时槽指令，则 EPC 寄存器更新为转移指令的 PC 值，同时 Cause 寄存器的 BD 位置 1。

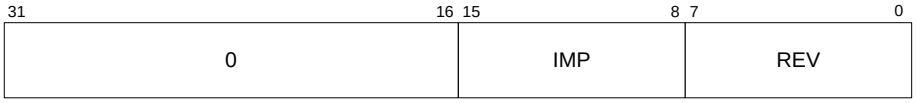
当发生例外时，如果 Status 寄存器的 EXL 位为 1，则 EPC 寄存器不会被更新，但仍然可通过 MTC0 指令写入新值。



位域	位	读写	功能描述	复位值
EPC	[31:0]	R/W	发生例外的指令 PC 值。	未定义

3.9.3.14 CP0 寄存器 15：PRId

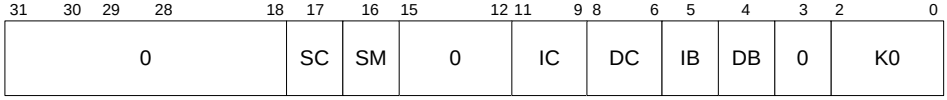
PRId 寄存器是一个 32 位的只读寄存器，包含了处理器标志与版本信息。



位域	位	读写	功能描述	复位值
0	[31:16]	-	未用域，读写均为 0。	0x0
IMP	[15:8]	RO	处理器实现标志。	0x42
REV	[7:0]	RO	处理器版本。	0x0

3.9.3.15 CP0 寄存器 16：Config

Config 寄存器包含了龙芯处理器主要的配置信息，除了 K0 域需要由软件进行配置之外，其他域都是常数只读域。



位域	位	读写	功能描述	复位值
0	[31:18]	-	未用域，读写均为 0。	0x0
SC	[17]	RO	是否存在二级 cache；32 位龙芯处理器中不存在二	0x1

			级 cache，1 表示不存在。	
SM	[16]	RO	cache 是否支持 dirty 共享状态；32 位龙芯处理器中不支持 cache 的 dirty 共享状态，1 表示不支持。	0x1
0	[15:12]	-	未用域，读写均为 0。	0x0
IC	[11:9]	RO	L1 指令 cache 的大小；IC 域值为 2，表示龙芯处理器的 L1 指令 cache 大小为 16KB。	0x2
DC	[8:6]	RO	L1 数据 cache 的大小；DC 域值为 2，表示龙芯处理器的 L1 数据 cache 大小为 16KB。	0x2
IB	[5]	RO	L1 指令 cache 的 cacheline 大小；IB 域值为 1，表示 32 字节。	0x1
DB	[4]	RO	L1 数据 cache 的 cacheline 大小；DB 域值为 1，表示 32 字节。	0x1
0	[15:12]	-	未用域，读写均为 0。	0x0
K0	[2:0]	R/W	kseg0 段的 cache 属性；当值为 011_2 时表示当前页具有 cacheable 属性，其余值均为 uncacheable 属性。	0x0

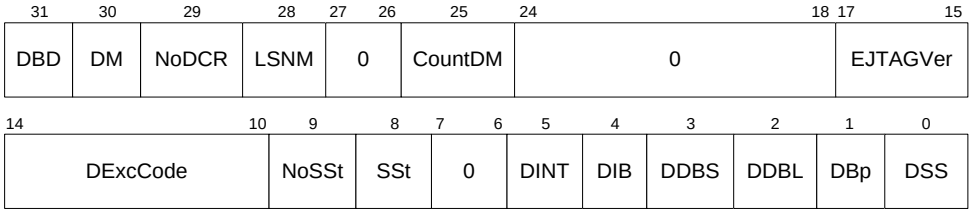
3.9.3.16 CP0 寄存器 23：Debug

Debug 寄存器用于控制调试例外的产生，并提供有关普通模式下发生调试例外及调试模式下发生普通例外的例外原因等信息。

在非调试模式下，Debug 寄存器中只有 DM 位与 EJTAGVer 位可读出有效值，读取其他位域的值则读出结果不可预测。非调试模式下不能对 Debug 寄存器进行写操作，否则操作结果不可知。

Debug 寄存器中的某些位域只有在发生调试例外或者调试模式下的例外才会进行更新，这些位域包括：

- DSS、DBp、DDBL、DDBS、DIB、DINT 在发生调试例外或者调试模式下的例外时进行更新。
- DExcCode 域在发生调试模式下的例外时进行更新，而在发生调试例外时其值不可预知；
- DBD 位在发生调试例外或者调试模式下的例外时进行更新。



位域	位	读写	功能描述	复位值
DBD	[31]	RO	标志最近一个调试例外或者调试模式下的例外是否发生在转移指令延时槽中。 0：未发生在转移指令延时槽中 1：发生在转移指令延时槽中	未定义
DM	[30]	RO	标志处理器当前是否工作在调试模式下。	0x0

位域	位	读写	功能描述	复位值
			0：处理器工作在非调试模式下 1：处理器工作在调试模式下	
NoDCR	[29]	RO	标志 dseg 段是否存在。 0：dseg 段存在 1：dseg 段不存在	0x0
LSNM	[28]	R/W	控制 load/store 指令访问 dseg 地址空间时是访问 dseg 段还是主存储器。 0：load/store 指令将访问 dseg 段 1：load/store 指令将访问主存储器	0x0
0	[27:26]	-	未用域，读写均为 0。	0x0
CountDM	[25]	R/W	控制 Count 寄存器在调试模式下是否继续计数。 0：Count 寄存器在调试模式下停止计数 1：Count 寄存器在调试模式下继续计数	0x1
0	[24:18]	-	未用域，读写均为 0。	0x0
EJTAGVer	[17:15]	RO	EJTAG 版本；龙芯处理器支持的 EJTAG 版本为 2.62。	0x2
DExcCode	[14:10]	RO	最近一次发生调试模式下例外的例外原因；该域的值与 Cause 寄存器中 ExcCode 域的值编码相同。	未定义
NoSSt	[9]	RO	是否支持单步执行功能。 0：支持单步执行调试功能 1：不支持单步执行调试功能	0x0
SSt	[8]	R/W	控制调试模式下是否使能单步执行调试功能。 0：不使能单步执行调试 1：使能单步执行调试	0x0
0	[7:6]	-	未用域，读写均为 0。	0x0
DINT	5	RO	标志是否发生了调试中断；当发生调试模式下的例外时该位自动清零。 0：未发生调试中断 1：发生了调试中断	未定义
DIB	4	RO	标志是否发生了指令硬件断点；当发生调试模式下的例外时该位自动清零。 0：未发生指令硬件断点 1：发生了指令硬件断点	未定义
DDBS	3	RO	标志是否发生了 store 指令引起的数据硬件断点；当发生调试模式下的例外时该位自动清零。 0：未发生 store 指令引起的数据硬件断点 1：发生了 store 指令引起的数据硬件断点	未定义
DDBL	2	RO	标志是否发生了 load 指令引起的数据硬件断点；当发生调试模式下的例外时该位自动清零。 0：未发生 load 指令引起的数据硬件断点 1：发生了 load 指令引起的数据硬件断点	未定义
DBp	1	RO	标志是否发生了调试指令断点；当发生调试模式下的例外时该位自动清零。 0：未发生调试指令断点	未定义

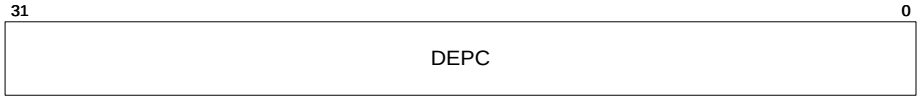
位域	位	读写	功能描述	复位值
			1：发生了调试指令断点	
DSS	0	RO	标志是否发生了单步调试例外；当发生调试模式下的例外时该位自动清零。 0：未发生单步调试例外 1：发生了单步调试例外	未定义

3.9.3.17 CP0 寄存器 24：DEPC

DEPC 寄存器是一个可读写寄存器，包含了完成调试例外处理之后恢复正常指令执行流程后的第一条指令的 PC 值。在发生例外的时候，根据当前指令执行情况，DEPC 寄存器自动进行如下更新：

- 如果发生调试例外的指令不是转移指令延时槽指令，则 DEPC 寄存器更新为发生调试例外的指令的 PC 值。
- 如果发生调试例外的指令是转移指令延时槽指令，则 DEPC 寄存器更新为转移指令的 PC 值，同时 Debug 寄存器的 DBD 位置 1。

当发生例外时，如果 Status 寄存器的 EXL 位为 1，则 DEPC 寄存器不会被更新，但仍然可通过 MTC0 指令写入新值。



位域	位	读写	功能描述	复位值
DEPC	[31:0]	R/W	发生调试例外的指令 PC 值。	未定义

3.9.3.18 CP0 寄存器 28：TagLo

TagLo 寄存器用作 cache 标签的对外接口，cache 指令通过 TagLo 寄存器可以访问或者更新 cache 的标签信息。

位域	位	读写	功能描述	复位值
0	[31:28]	-	未用域，读写均为 0。	0x0
TagLo	[27:8]	R/W	cache 行标签信息。	未定义
state	[7:6]	R/W	cache 行状态信息，值 11_2 表示 cache 行有效，其他值均表示无效。	未定义
0	[5:0]	-	未用域，读写均为 0。	0x0

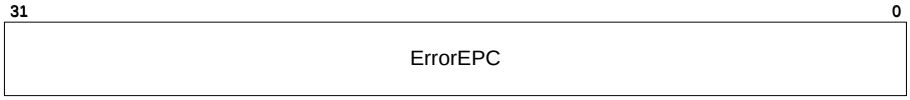
3.9.3.19 CP0 寄存器 29：TagHi

TagHi 寄存器目前没有具体的控制功能，可用于便签寄存器。

位域	位	读写	功能描述	复位值
TagHi	[31:0]	R/W	用于便签寄存器	未定义

3.9.3.20 CP0 寄存器 30：ErrorEPC

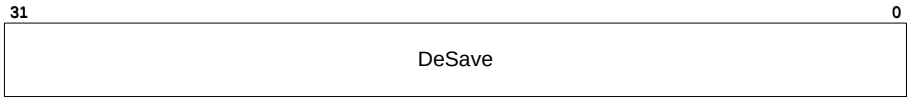
ErrorEPC 用于发生错误时保存发生例外的指令 PC 值，但 GSC3281 芯片及 32 位龙芯处理器对于各种错误情况采用了特定的处理方式，不再以一个“错误”的形式反映到 ErrorEPC 寄存器中，因此 ErrorEPC 可用作便签寄存器。



位域	位	读写	功能描述	复位值
ErrorEPC	[31:0]	R/W	用于便签寄存器	未定义

3.9.3.21 CP0 寄存器 31：DeSave

DeSave 寄存器是一个 32 位可读写寄存器，在调试例外处理程序中可用作保存现场的临时中转便签寄存器，从而使得调试软件不会破坏处理器的原有状态。



位域	位	读写	功能描述	复位值
DeSave	[31:0]	R/W	调试例外处理的便签寄存器。	未定义

3.10 片上调试特性

32 位龙芯处理器通过扩充 IEEE P1149.1 协议的 JTAG 测试访问端口（TAP），并在处理器内部增加控制模块，实现了软件调试断点、调试中断、硬件断点以及单步执行等多种片上调试功能，调试主机只需要通过一根 JTAG 调试电缆就可以访问龙芯处理器内部寄存器等各种资源，并控制龙芯处理器的运行过程，大大方便了软件开发与系统调试。在本芯片提供的软

件开发包中，将包含完善的集成开发环境与片上调试功能。

32 位龙芯处理器的片上调试功能符合 EJTAG R2.62 规范，主要包括有如下的功能：

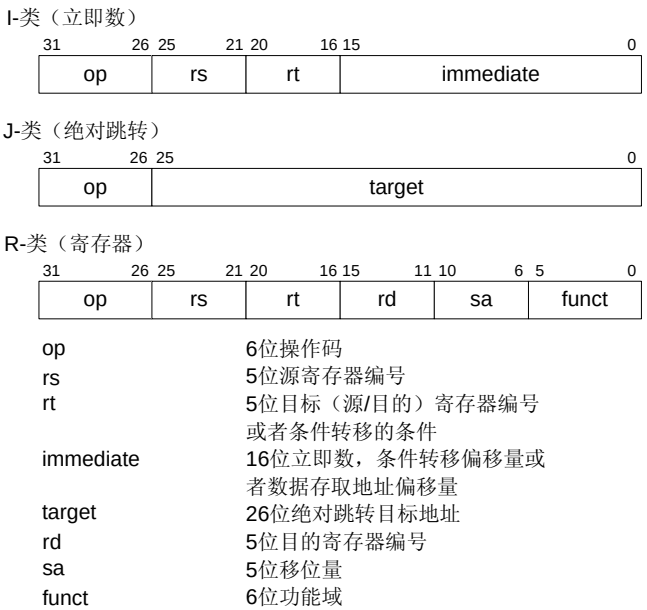
- Off-board 存储器功能：当处理器处于调试模式下，允许处理器通过 TAP 端口从系统之外的地址取指与读写数据，这部分调试专用空间被映射到处理器的逻辑地址空间中，对调试空间的访问就如同访问物理存储器一样。
- 通过 TAP 端口进行系统访问：调试主机可以通过 TAP 端口获得处理器的配置信息以及内部状态，并控制处理器的运行过程。
- 软件调试断点：通过执行软件调试断点指令，龙芯处理器可以进入到调试模式，从而允许调试主机对 RAM 区任意插入断点进行调试。
- 硬件断点：包括指令硬件断点、不带数值比较的数据硬件断点和带有数值比较的数据硬件断点，通过硬件断点，可以方便地实现对任意指令执行过程以及数据访存操作的监控。
- 单步执行：允许龙芯处理器单步执行程序中的每一条指令，并且不会额外占用存储空间。
- 调试中断：由调试主机通过 TAP 端口以异步方式使得龙芯处理器进入到调试模式进行调试处理。
- 特殊的 EJTAGBOOT 调试启动方式：允许龙芯处理器在复位之后立即进入到调试模式进行一些特殊的检查与初始化操作。

3.11 指令集

3.11.1 指令集概览

3.11.1.1 指令格式

龙芯处理器所有指令位宽均为 32 位，字边界对齐，指令格式可以分为三类：立即数类（I 类）、绝对跳转类（J 类）与寄存器类（R 类），如图 3-16 所示。



op	6位操作码
rs	5位源寄存器编号
rt	5位目标（源/目的）寄存器编号 或者条件转移的条件
immediate	16位立即数，条件转移偏移量或 者数据存取地址偏移量
target	26位绝对跳转目标地址
rd	5位目的寄存器编号
sa	5位移位量
funct	6位功能域

图 3-16 龙芯处理器指令格式

3.11.1.2 数据存取指令

龙芯处理器的数据存取指令（load/store 指令）属于 I 类指令格式，用于在存储器与通用寄存器之间搬移数据，唯一支持的寻址模式为基址寄存器加 16 位有符号立即数偏移量。龙芯处理器的数据格式采用小尾端次序，支持 8 位、16 位、32 位的对齐数据存取以及 32 位非对齐数据存取。

在后续章节有关数据存取指令的描述中，使用到了一些涉及虚拟地址、虚实地址转换等概念的描述性函数，对这些函数的说明如表 3-14 所示。

表 3-14 数据存取指令描述使用到的描述性函数

函数	说明
AddressTranslation	通过 TLB 进行虚实地址转换，如果转换成功，则得到有效的物理地址，如果转换不成功，则产生例外。
LoadMemory	根据指定的物理地址从 cache 或内存中读取数据，物理地址的最低两位以及访问类型决定了一个字的四个字节中哪些字节将被读取。
StoreMemory	根据指定的物理地址向 cache 或内存中写入数据，物理地址的最低两位以及访问类型决定了一个字的四个字节中哪些字节将被写入。

3.11.1.3 数据运算指令

龙芯处理器的数据运算指令有两种 R 类与 I 类两种指令格式，在 R 类指令格式的计算指令中，两个操作数均来自于通用寄存器，在 I 类指令格式的计算指令中，一个操作来自于通用寄存器，另一个操作数为 16 位立即数。

龙芯处理器的数据运算指令可以执行如下运算操作：

- 算术运算
- 逻辑运算
- 移位运算
- 乘法运算
- 除法运算

3.11.1.4 转移指令

龙芯处理器的转移指令包括绝对跳转指令与条件跳转指令两类，转移指令后面的一条指令称为转移指令延时槽(delay slot)指令，所有的转移指令后面都带有一条延时槽指令；当龙芯处理器执行一条转移指令时，除了 likely 条件转移指令外，所有其他转移指令的延时槽指令都必定会执行，而不管转移指令本身是否转移成功。

高级语言中的函数调用通常用绝对跳转或者绝对跳转并链接指令来实现，二者均为 J 类指令，在 J 类指令中，指令中的 26 位目标地址左移两位，并在高位与当前指令 PC 的最高 4 位进行拼接形成最终的目标转移地址。

函数返回、分支派遣以及跨页跳转等情况通常采用绝对跳转到目标寄存器指令以及绝对跳转到目标寄存器并链接指令来实现，二者均为 R 类指令，转移目标地址由通用寄存器中的 32 位数值指定。

条件转移指令的转移目标由延时槽指令 PC 与 16 位偏移量左移两位相加而得。对于 likely 条件转移指令而言，如果转移不成功，则延时槽指令不会真正执行；不管是否转移成功，所

有其他转移指令的延时槽指令都一定会执行。

3.11.1.5 控制指令

龙芯处理器的控制指令主要包括自陷指令、系统调用指令、断点指令、cache 指令以及休眠指令等。

3.11.1.6 协处理器指令

龙芯处理器只有一个系统控制协处理器（CP0），协处理器指令用于对 CP0 寄存器进行操作以实现存储管理与例外处理。

3.11.2 指令集汇总

表 3-15 以字母顺序汇总了龙芯处理器支持的全部指令。

表 3-15 龙芯处理器指令集汇总

指令名称	指令描述	指令功能
ADD	整数相加	Rd = Rs + Rt
ADDI	整数与立即数相加	Rt = Rs + Immed
ADDIU	无符号整数与无符号立即数相加	Rt = Rs + $_U$Immed
ADDU	无符号整数相加	Rd = Rs + $_U$Rt
AND	逻辑与	Rd = Rs & Rt
ANDI	与立即数逻辑与操作	Rt = Rs & (0_{16} || Immed)
BEQ	相等则转移	if (Rs == Rt) PC += (int)offset

BEQL	相等则可能转移	if (Rs == Rt)
		PC += (int)offset
		else
		ignore next instruction

BGEZ	大于等于 0 则转移	if (!Rs[31]) PC += (int)offset
BGEZAL	大于等于 0 则转移并链接	GPR[31] = PC + 8 if (!Rs[31]) PC += (int)offset

BGEZALL	大于等于 0 则可能转移并链接	GPR[31] = PC + 8
		if (!Rs[31])
		PC += (int)offset
		else
		ignore next instruction

BGEZL	大于等于 0 则可能转移	if (!Rs[31])
		PC += (int)offset
		else
		ignore next instruction

BGTZ	大于 0 则转移	if (!Rs[31] && Rs!=0) PC += (int)offset
BGTZL	大于 0 则可能转移	if (!Rs[31] && Rs!=0) PC += (int)offset

指令名称	指令描述	指令功能
		else ignore next instruction
BLEZ	小于等于 0 则转移	if (Rs[31] || Rs==0) PC += (int)offset

BLEZL	小于等于 0 则可能转移	if (Rs[31] || Rs==0)
		PC += (int)offset
		else
		ignore next instruction

BLTZ	小于 0 则转移	if (Rs[31]) PC += (int)offset
BLTZAL	小于 0 则转移并链接	GPR[31] = PC + 8 if (Rs[31]) PC += (int)offset

BLTZALL	小于 0 则可能转移并链接	GPR[31] = PC + 8
		if (Rs[31])
		PC += (int)offset
		else
		ignore next instruction

BLTZL	小于 0 则可能转移	if (Rs[31])
		PC += (int)offset
		else
		ignore next instruction

BNE	不相等则转移	if (Rs != Rt) PC += (int)offset

BNEL	不相等则可能转移	if (Rs != Rt)
		PC += (int)offset
		Else
		ignore next instruction

BREAK	断点	发生 break 例外
CACHE	cache 操作	执行 cache 操作，请参考指令描述
DERET	调试例外返回	PC = DEPC 退出调试模式
DIV	除法	LO = (int)Rs / (int)Rt HI = (int)Rs % (int)Rt
DIVU	无符号除法	LO = (uns)Rs / (uns)Rt HI = (uns)Rs / (uns)Rt

ERET	例外返回	if (SR[2])
		PC = ErrorEPC
		else
		PC = EPC
		SR[1] = 0
		SR[2] = 0
		LL = 0

J	绝对跳转	PC = PC[31:28] || (offset << 2)
JAL	绝对跳转并链接	GPR[31] = PC + 8 PC = PC[31:28] || (offset << 2)
JALR	绝对跳转到寄存器并链接	Rd = PC + 8

指令名称	指令描述	指令功能
		PC = Rs
JR	绝对跳转到寄存器	PC = Rs
LB	装载一个字节	Rt = (byte)mem[Rs + offset]
LBU	装载一个无符号字节	Rt = (ubyte)mem[Rs + offset]
LH	装载一个半字	Rt = (half)mem[Rs + offset]
LHU	装载一个无符号半字	Rt = (uhalf)mem[Rs + offset]
LL	装载一个字并链接	Rt = mem[Rs + offset] LL = 1 LLAddr = Rs + offset
LUI	装载一个立即数到寄存器高半字	Rt = immediate << 16
LW	装载一个字	Rt = mem[Rs + offset]
LWL	装载一个字的左半部分到寄存器	非对齐装载数据，请参考指令描述
LWR	装载一个字的右半部分到寄存器	非对齐装载数据，请参考指令描述
MFC0	从协处理器 0 取数	Rt = CP0[rd]
MFHI	从 HI 寄存器取数	Rd = HI
MFLO	从 LO 寄存器取数	Rd = LO
MTC0	向协处理器 0 存数	CP0[rd] = Rt
MTHI	向 HI 寄存器存数	HI = Rs
MTLO	向 LO 寄存器存数	LO = Rs
MULT	整数乘法	HI || LO = (int)Rs * (int)Rd
MULTU	无符号整数乘法	HI || LO = (uns)Rs * (uns)Rd
NOR	逻辑异非操作	Rd = ~ (Rs | Rt)
OR	逻辑或操作	Rd = Rs | Rt
ORI	与一个立即数逻辑或操作	Rt = Rs | immediate
SB	存储一个字节	(byte)mem[Rs + offset] = Rt
SC	条件存储一个字	if (LL = 1) mem[Rs + offset] = Rt Rt = LL
SDBBP	软件调试断点	发生软件调试断点例外
SH	存储一个半字	(half)mem[Rs + offset] = Rt
SLL	逻辑左移	Rd = Rt << sa
SLLV	移位量可变的逻辑左移	Rd = Rt << Rs[4:0]

SLT	小于则置位	if ((int)Rs < (int)Rt)
		  Rd = 1
		else
		  Rd = 0

SLTI	小于立即数则置位	if ((int)Rs < (int)immediate)
		  Rt = 1
		else
		  Rt = 0

SLTIU	无符号小于立即数则置位	if ((uns)Rs < (uns)immediate)
		  Rt = 1
		else

指令名称	指令描述	指令功能
		Rt = 0

SLTU	无符号小于则置位	if ((uns)Rs < (uns)Rt)
		Rd = 1
		else
		Rd = 0

SRA	算术右移	Rd = (int) Rt >> sa
SRAV	移位量可变的算术右移	Rd = (int) Rt >> Rs[4:0]
SRL	逻辑右移	Rd = (uns) Rt >> sa
SRLV	移位量可变的逻辑右移	Rd = (uns) Rt >> Rs[4:0]
SUB	整数减法	Rt = (int)Rs – (int)Rd
SUBU	无符号整数减法	Rt = (uns)Rs – (uns)Rd
SW	存储一个字	mem[Rs + offset] = Rt
SWL	存储一个字的左侧部分	非对齐存储数据，请参考指令描述
SWR	存储一个字的右侧部分	非对齐存储数据，请参考指令描述
SYNC	同步	请参考指令描述
SYSCALL	系统调用	发生系统调用例外
TEQ	相等则自陷	if ((int)Rs = (int)Rt) TrapException
TEQI	与立即数相等则自陷	if ((int)Rs = (int)immediate) TrapException
TGE	大于等于则自陷	if ((int)Rs >= (int)Rt) TrapException
TGEI	大于等于立即数则自陷	if ((int)Rs >= (int)immediate) TrapException
TGEIU	无符号大于等于立即数则自陷	if ((uns)Rs >= (uns)immediate) TrapException
TGEU	无符号大于等于则自陷	if ((uns)Rs >= (uns)Rt) TrapException
TLBP	TLB 探测	请参考指令描述
TLBR	TLB 读	请参考指令描述
TLBWI	TLB 索引写	请参考指令描述
TLBWR	TLB 随机写	请参考指令描述
TLT	小于则自陷	if ((int)Rs < (int)Rt) TrapException
TLTI	小于立即数则自陷	if ((int)Rs < (int)immediate) TrapException
TLTIU	无符号小于立即数则自陷	if ((uns)Rs < (uns)immediate) TrapException
TLTU	无符号小于则自陷	if ((uns)Rs < (uns)Rt) TrapException
TNE	不等于则自陷	if (Rs != Rt) TrapException
TNEI	不等于立即数则自陷	if (Rs != (int)immediate) TrapException
WAIT	等待	处理器进入休眠状态
XOR	逻辑异或	Rd = Rs ^ Rt
XORI	与立即数逻辑异或	Rt = Rs ^ (uns)immediate

3.11.3 指令描述

3.11.3.1 ADD

ADD Add **ADD**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	ADD 100000
6	5	5	5	5	6

格式：ADD rd, rs, rt

描述：

通用寄存器 rs 的值与通用寄存器 rt 的值相加，结果存放到通用寄存器 rd 中。如果相加结果第 30 位与第 31 位的进位输出值不同（即 2 的补码溢出），则产生一个整数溢出例外，此时目标寄存器 rd 的值不会被更新。

操作：

T: GPR[rd] ← GPR[rs] + GPR[rt]

例外：

整数溢出例外

3.11.3.2 ADDI

ADDI Add Immediate **ADDI**

31 26	25 21	20 16	15 0
ADDI 001000	rs	rt	immediate
6	5	5	16

格式：ADDI rt, rs, immediate

描述：

16 位立即数符号扩展后的值与通用寄存器 rs 的值相加，结果存放到通用寄存器 rt 中。如果相加结果第 30 位与第 31 位的进位输出值不同（即 2 的补码溢出），则产生一个整数溢出例外，此时目标寄存器 rt 的值不会被更新。

操作：

T: GPR[rt] ← GPR[rs] + $(immediate_{15})^{16}$ || $immediate_{15...0}$

例外：

整数溢出例外

3.11.3.3 ADDIU

ADDIU Add Immediate Unsigned **ADDIU**

31 26	25 21	20 16	15 0
ADDIU 001001	rs	rt	immediate
6	5	5	16

格式：ADDIU rt, rs, immediate

描述：

16 位立即数符号扩展后的值与通用寄存器 rs 的值相加，结果存放到通用寄存器 rt 中。

ADDIU 指令与 ADDI 指令的不同之处在于 ADDIU 指令不会产生整数溢出例外。

操作：

T: GPR[rt] ← GPR[rs] + $(immediate_{15})^{16}$ || $immediate_{15...0}$

例外：

无

3.11.3.4 ADDU

ADDU Add Unsigned **ADDU**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	ADDU 100001
6	5	5	5	5	6

格式：ADDU rd, rs, rt

描述：

通用寄存器 rs 的值与通用寄存器 rt 的值相加，结果存放到通用寄存器 rd 中。ADDU 指令在任何条件下都不会产生溢出例外。

ADDU 指令与 ADD 指令的不同之处在于 ADDU 指令不会产生溢出例外。

操作：

T: GPR[rd] ← GPR[rs] + GPR[rt]

例外：

无

3.11.3.5 AND

AND And **AND**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	AND 100100
6	5	5	5	5	6

格式：AND rd, rs, rt

描述：

通用寄存器 rs 的值与通用寄存器 rt 的值按位逻辑“与”运算，结果存放到通用寄存器 rd 中。

操作：

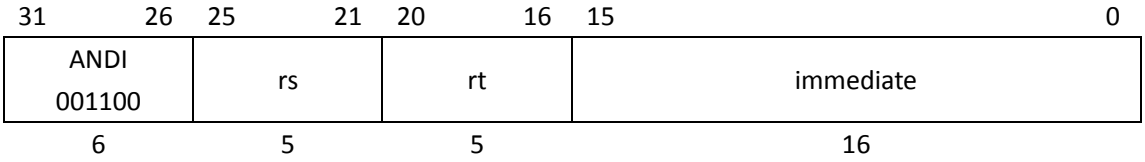
T: GPR[rd]←GPR[rs] and GPR[rt]

例外：

无

3.11.3.6 ANDI

ANDI And Immediate **ANDI**



格式：ANDI rt, rs, immediate

描述：

零扩展 16 位立即数并与通用寄存器 rs 的值按位逻辑“与”运算，结果存放到通用寄存器 rt 中。

操作：

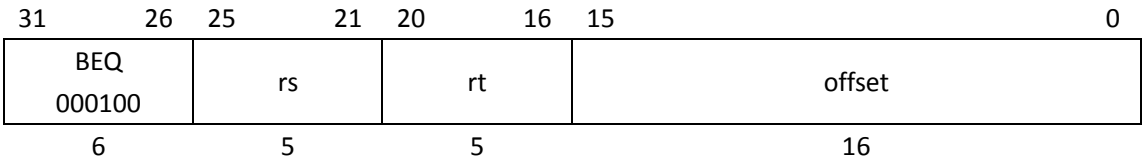
T: GPR[rt] ← 0^{16} || (immediate and GPR[rs]$_{15...0}$)

例外：

无

3.11.3.7 BEQ

BEQ Branch On Equal **BEQ**



格式：BEQ rs, rt, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得。通用寄存器 rs 的值与和通用寄存器 rt 的值进行比较，如果相等，则在延迟一条指令后转移至目的地址。

操作：

T: target ← (offset$_{15}$)^{14} || offset || 0^2

condition ← (GPR[rs] = GPR[rt])

T+1: if condition then

PC ← PC + target

endif

例外：

无

3.11.3.8 BEQL

BEQL Branch On Equal Likely **BEQL**

31 26	25 21	20 16	15 0
BEQL 010100	rs	rt	offset
6	5	5	16

格式：BEQL rs, rt, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得。通用寄存器 rs 的值和通用寄存器 rt 的值进行比较，如果相等，则在延迟一条指令后转移至目的地址。如果转移未发生，则使延时槽指令无效。

操作：

```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs] = GPR[rt])
T+1: if condition then
                  PC ← PC + target
     else
                  NullifyCurrentInstruction
     endif
```

例外：

无

3.11.3.9 BGEZ

BGEZ Branch On Greater Than Or Equal To Zero **BGEZ**

31 26	25 21	20 16	15 0
REGIMM 000001	rs	BGEZ 00001	offset
6	5	5	16

格式：BGEZ rs, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得。如果通用寄存器 rs 的符号位等于“零”，则在延迟一条指令后转移至目的地址。

操作：

```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs]31 = 0])
T+1: if condition then
                  PC ← PC + target
     endif
```

例外：

无

3.11.3.10 BGEZAL

BGEZAL Branch On Greater Than Or Equal To Zero And Link **BGEZAL**

31 26	25 21	20 16	15 0
REGIMM 000001	rs	BGEZAL 10001	offset
6	5	5	16

格式：BGEZAL rs, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得，同时延时槽指令的下一条指令地址无条件地被保存到链接寄存器 r31 中。如果通用寄存器 rs 的符号位等于“零”，则在延迟一条指令后转移至目的地址。通用寄存器 rs 不能是 r31 寄存器，否则再次执行该指令将出现不被期望的结果。另一方面，如果出现这种情况，系统却无法检测到，因此应当避免出现这种情况。

操作：

```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs]31 = 0])
      GPR[31] ← PC + 8
T+1: if condition then
                  PC ← PC + target
      endif
```

例外：

无

3.11.3.11 BGEZALL

BGEZALL Branch On Greater Than Or Equal To Zero And Link Likely **BGEZALL**

31 26	25 21	20 16	15 0
REGIMM 000001	rs	BGEZALL 10011	offset
6	5	5	16

格式：BGEZALL rs, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得，同时延时槽指令的下一条指令地址无条件地被保存到链接寄存器 r31 中。如果通用寄存器 rs 的符号位等于“零”，则在延迟一条指令后转移至目的地址。通用寄存器 rs 不能是 r31 寄存器，否则再次执行该指令将出现不被期望的结果。另一方面，如果出现这种情况，系统却无法检测到，因此应当避免出现这种情况。如果转移未发生，则使延时槽指令无效。

操作：

```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs]31 = 0])
      GPR[31] ← PC + 8
T+1: if condition then
                  PC ← PC + target
      else
                  NullifyCurrentInstruction
      endif
```

例外：

无

3.11.3.12 BGEZL

BGEZL Branch On Greater Than Or Equal To Zero Likely **BGEZL**

31 26	25 21	20 16	15 0
REGIMM 000001	rs	BGEZL 00011	offset
6	5	5	16

格式：BGEZL rs, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得。如果通用寄存器 rs 的符号位等于“零”，则在延迟一条指令后转移至目的地址。如果转移未发生，则使延时槽指令无效。

操作：

```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs]31 = 0)
T+1: if condition then
                  PC ← PC + target
      else
                  NullifyCurrentInstruction
      endif
```

例外：

无

3.11.3.13 BGTZ

BGTZ Branch On Greater Than Zero **BGTZ**

31 26	25 21	20 16	15 0
BGTZ 000111	rs	0 00000	offset

6	5	5	16

格式：BGTZ rs, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得。通用寄存器 rs 的值与“零”比较，如果 rs 符号位等于“零”而其余位不等于“零”，则在延迟一条指令后转移至目的地址。

操作：

```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs]31 = 0) and (GPR[rs] ≠ 0^32)
T+1: if condition then
                  PC ← PC + target
      endif
```

例外：

无

3.11.3.14 BGTZL

BGTZL Branch On Greater Than Zero Likely **BGTZL**

31 26	25 21	20 16	15 0
BGTZL 010111	rs	0 00000	offset
6	5	5	16

格式：BGTZL rs, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得。通用寄存器 rs 的值与“零”比较，如果 rs 符号位等于“零”而其余位不等于“零”，则在延迟一条指令后转移至目的地址。如果转移未发生，则使延时槽指令无效。

操作：

```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs]31 = 0) and   (GPR[rs] ≠ 0^32)
T+1: if condition then
                  PC ← PC + target
      else
                  NullifyCurrentInstruction
      endif
```

例外：

无

3.11.3.15 BLEZ

BLEZ Branch On Less Than Or Equal To Zero **BLEZ**

31 26	25 21	20 16	15 0
BLEZ 000110	rs	0 00000	offset
6	5	5	16

格式：BLEZ rs, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得。通用寄存器 rs 的值与“零”比较，如果 rs 符号位为“一”或 rs 寄存器的值为“零”，则在延迟一条指令后转移至目的地址。

操作：

```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs]31 = 1) or (GPR[rs] = 0^32)
T+1: if condition then
                  PC ← PC + target
      endif
```

例外：

无

3.11.3.16 BLEZL

BLEZL Branch on Less Than Or Equal To Zero Likely **BLEZL**

31 26	25 21	20 16	15 0
BLEZL 010110	rs	0 00000	offset
6	5	5	16

格式：BLEZL rs, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得。通用寄存器 rs 的值与“零”比较，如果 rs 符号位为“一”或 rs 寄存器的值为“零”，则在延迟一条指令后转移至目的地址。如果转移未发生，则使延时槽指令无效。

操作：

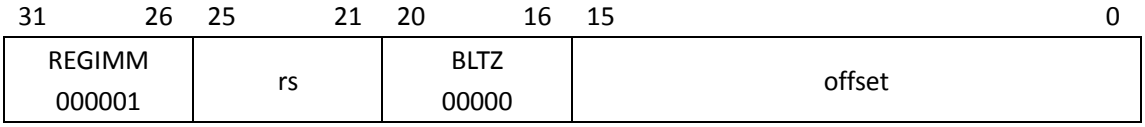
```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs]31 = 1) or (GPR[rs] = 0^32)
T+1: if condition then
                  PC ← PC + target
      else
                  NullifyCurrentInstruction
      endif
```

例外：

无

3.11.3.17 BLTZ

BLTZ Branch On Less Than Zero **BLTZ**



格式：BLTZ rs, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得。如果通用寄存器 rs 的符号位为“一”，则在延迟一条指令后转移至目的地址。

操作：

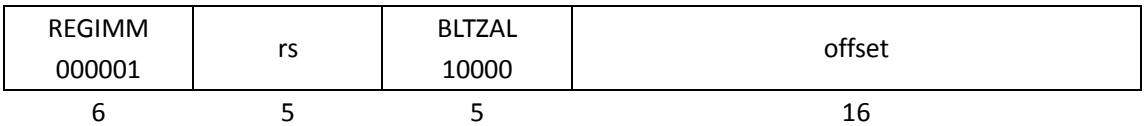
```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs]31 = 1)
T+1: if condition then
                  PC ← PC + target
      endif
```

例外：

无

3.11.3.18 BLTZAL

BLTZAL Branch On Less Than Zero And Link **BLTZAL**



格式：BLTZAL rs, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得，同时延时槽指令的下一条指令地址无条件地被保存到链接寄存器 r31 中。如果通用寄存器 rs 的符号位为“一”，则在延迟一条指令后转移至目的地址。通用寄存器 rs 不能是 r31 寄存器，否则再次执行该指令将出现不被期望的结果。另一方面，如果出现这种情况，系统却无法检测到，因此应当避免出现这种情况。

操作：

```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs]31 = 1)
      GPR[31] ← PC + 8
T+1: if condition then
                  PC ← PC + target
      endif
```

例外：

无

3.11.3.19 BLTZALL

BLTZALL Branch On Less Than Zero And Link Likely **BLTZALL**

31 26	25 21	20 16	15 0
REGIMM 000001	rs	BLTZALL 10010	offset
6	5	5	16

格式：BLTZALL rs, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得，同时延时槽指令的下一条指令地址无条件地被保存到链接寄存器 r31 中。如果通用寄存器 rs 的符号位为“一”，则在延迟一条指令后转移至目的地址。通用寄存器 rs 不能是 r31 寄存器，否则再次执行该指令将出现不被期望的结果。另一方面，如果出现这种情况，系统却无法检测到，因此应当避免出现这种情况。如果转移未发生，则使延时槽指令无效。

操作：

```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs]31 = 1)
      GPR[31] ← PC + 8
T+1: if condition then
                  PC ← PC + target
      else
                  NullifyCurrentInstruction
      endif
```

例外：

无

3.11.3.20 BLTZL

BLTZL Branch On Less Than Zero Likely **BLTZL**

31 26	25 21	20 16	15 0
REGIMM 000001	rs	BLTZL 00010	offset
6	5	5	16

格式：BLTZL rs, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得。如果通用寄存器 rs 的符号位为“一”，则在延迟一条指令后转移至目的地址。如果转移未发生，则使延时槽指令无效。

操作：

T: target ← $(offset_{15})^{14}$ || offset || 0^2

condition ← $(GPR[rs]_{31} = 1)$

T+1: if condition then

PC ← PC + target

else

NullifyCurrentInstruction

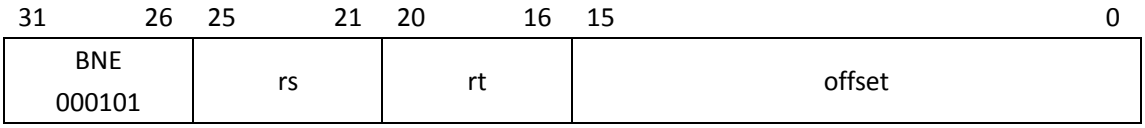
endif

例外：

无

3.11.3.21 BNE

BNE Branch On Not Equal **BNE**



格式：BNE rs, rt, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得。通用寄存器 rs 的值与通用寄存器 rt 的值进行比较，如果二者不相等，则延迟一条指令后转移至目的地址。

操作：

T: target ← $(offset_{15})^{14}$ || offset || 0^2

condition ← (GPR[rs] ≠ GPR[rt])

T+1: if condition then

PC ← PC + target

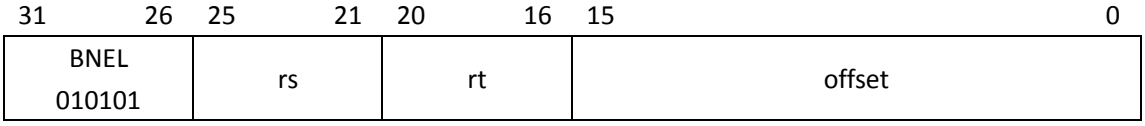
endif

例外：

无

3.11.3.22 BNEL

BNEL Branch On Not Equal Likely **BNEL**



格式：BNEL rs, rt, offset

描述：

转移目的地址由延时槽指令地址与 16 位偏移量左移 2 位并符号扩展后的值相加而得。通用寄存器 rs 的值与通用寄存器 rt 的值进行比较，如果二者不相等，则延迟一条指令后转移至目的地址。如果转移未发生，则使延时槽指令无效。

操作：

```
T:    target ← (offset15)^14 || offset || 0^2
      condition ← (GPR[rs] ≠ GPR[rt])
T+1: if condition then
                  PC ← PC + target
      else
                  NullifyCurrentInstruction
      endif
```

例外：

无

3.11.3.23 BREAK

BREAK Breakpoint **BREAK**

31 26	25 6	5 0
SPECIAL 000000	code	BREAK 001101
6	20	6

格式：BREAK

描述：

执行 BREAK 指令将发生断点例外，并立即无条件转移到例外处理程序。

code 域可用于软件传递参数，在例外处理程序中只有读取该指令的值才能提取 code 域。

操作：

```
T: BreakpointException
```

例外：

断点例外

3.11.3.24 CACHE

CACHE Cache **CACHE**

31 26	25 21	20 16	15 0
CACHE 101111	base	op	offset
6	5	5	16

格式：CACHE op, offset(base)

描述：

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成虚拟地址，经虚实地址转换后产生有效的物理地址，在该物理地址上执行五位 op 域指定的 cache 操作。

在用户模式或者监管模式下，如果 Status 寄存器的 CP0 使能位为“零”，则执行 CACHE 指令将引起一个协处理器不可用例外。在 32 位龙芯处理器中，如果 CACHE 指令对下面二表中未列举到的组合情况执行操作，则操作结果不可预知。

32 位龙芯处理器使用虚拟地址的[11:5]位对 cache 进行索引，指定对一级 cache 中具体哪个 cache 块进行操作。

如果 CACHE 指令访问的地址在 cache 中命中，则执行指定的 cache 操作；如果在 cache 中不命中，在不执行任何操作。

当 CACHE 指令执行一个写回操作时，由于 32 位龙芯处理器不包含二级 cache，因此一级 cache 的数据将直接写回到内存中。

CACHE 指令在执行 cache 操作时可能引起 TLB 重填例外或者 TLB 无效例外。对于物理地址不需要与 cache 标签进行比较的索引类 cache 操作，可以使用 unmapped 地址访问 cache，从而避免产生 TLB 例外。此外，CACHE 指令不会引起 TLB 修改例外。

CACHE 指令的[17:16]位用于指定对处理器中哪一个 cache 进行操作，如下表所示。

CACHE 指令字的[17:16]位	名称	指定的 cache
0	I	一级指令 cache
1	D	一级数据 cache
2	Reserved	-
3	Reserved	-

CACHE 指令的[20:18]位用于指定执行什么样的 cache 操作，如下表所示。

CACHE 指令的[20:18]位	目标 cache	操作名称	操作描述
0	I	Index Invalidate	将指定 cache 块的状态位置为无效。
0	D	Index Writeback Invalidate	根据虚拟地址索引访问一级数据 cache，检查指定 cache 块的状态位以及写回标志位（W 位），如果 cache 块的状态位为有效并且 W 位为“一”，则将 cache 块的数据写回到内存中，写回地址来源于 cache 标签（tag）。同时，当前 cache 块的状态位置为无效，W 位保持不变。
1	D	Index Load Tag	根据虚拟地址索引访问一级数据 cache，将指定 cache 块的标签（tag）值读入到 TagLo 寄存器中。
2	D	Index Store Tag	根据虚拟地址索引访问一级数据 cache，将 TagLo 寄存器的值写入到指定 cache 块的标签（tag）中。
4	D	Hit Invalidate	根据虚拟地址索引访问一级数据 cache，如果命中，则将指定 cache 块的状态位置为无效。
5	D	Hit Writeback Invalidate	根据虚拟地址索引访问一级数据 cache，如果命中，并且指定 cache 块的 W 位为“一”，则将 cache 块的数据写回到内存中，同时状态位置为无效。

操作：

T: vAddr ← $((offset_{15})^{16}$ || $offset_{15...0})$ + GPR[base]
(pAddr, uncached) ← AddressTranslation(vAddr, DATA)
CacheOp(op, vAddr, pAddr)

例外：

协处理器不可用例外

3.11.3.25 DERET

DERET Debug Exception Return **DERET**

31 26	25 24	6 5 0
COP0 010000	CO 1	0 000 0000 0000 0000 0000	DERET 011111
6		19	6

格式：DERET

描述：

DERET 指令用以从调试例外处理程序中返回，同时处理器工作模式恢复为发生调试例外之前的非调试模式。与条件转移以及绝对转移指令相比，ERET 指令的下一条指令不会被执行。

在 LL 指令和 SC 指令之间执行 DERET 指令不会使得 SC 指令执行失败。

操作：

```
T: if DebugDM = 1 then
        DebugDM ← 0
        PC ← DEPC
   else
        UNDEFINED
```

例外：

协处理器不可用例外

3.11.3.26 DIV

DIV Divide **DIV**

31 26	25 21	20 16	15 6	5 0
SPECIAL 000000	rs	rt	0 00 0000 0000	DIV 011010
6	5	5	10	6

格式：DIV rs, rt

描述：

通用寄存器 rs 和 rt 做 2 的补码除法运算，其中 rs 的值为被除数。在任何情况下该指令都不会产生溢出例外，如果除数等于“零”，则结果不可预知。

在该指令之后，通常跟随一些额外的指令用以检查除数是否为“零”以及是否发生溢出。

该指令运算完成之后，商保存到特殊寄存器 LO 中，余数保存到特殊寄存器 HI 中。

操作：

```
T: LO ← GPR[rs] div GPR[rt]
   HI ← GPR[rs] mod GPR[rt]
```

例外：

无

3.11.3.27 DIVU

DIVU Divide Unsigned **DIVU**

31 26	25 21	20 16	15 6	5 0
SPECIAL 000000	rs	rt	0 00 0000 0000	DIVU 011011
6	5	5	10	6

格式：DIVU rs, rt

描述：

通用寄存器 rs 和 rt 的做无符号数除法运算，其中 rs 的值为被除数。在任何情况下该指令都不会产生溢出例外，如果除数等于“零”，则结果不可以预知。

在该指令之后，通常跟随一些额外的指令用以检查除数是否为“零”以及是否发生溢出。

该指令运算完成之后，商保存到特殊寄存器 LO 中，余数保存到特殊寄存器 HI 中。

操作：

T: LO ← (0 || GPR[rs]) div (0 || GPR[rt])
HI ← (0 || GPR[rs]) mod (0 || GPR[rt])

例外：

无

3.11.3.28 ERET

ERET Exception Return **ERET**

31 26	25	24 6	5 0
COP0 010000	CO 1	0 000 0000 0000 0000 0000	ERET 011000
6		19	6

格式：ERET

描述：

ERET 用以从中断、例外或错误的处理程序中返回。与条件转移以及绝对转移指令相比，ERET 指令的下一条指令不会被执行。

ERET 指令本身不能被放置在转移指令延时槽中。

如果处理器是从错误处理程序中返回（SR_2 = 1），则 PC 更新为 ErrorEPC 寄存器的值，并且清零 Status 寄存器的 ERL 位（SR_2）；否则（SR_2=0）PC 更新为 EPC 寄存器的值，并且清零 Status 寄存器的 ERL 位（SR_1）。

在 LL 指令和 SC 指令之间执行 ERET 指令会使得 SC 指令执行失败。

操作：

T: if SR_2 = 1 then
PC ← ErrorEPC

SR ← $SR_{31...3}$ || 0 || $SR_{1...0}$

else

PC ← EPC

SR ← $SR_{31...2}$ || 0 || SR_0

endif

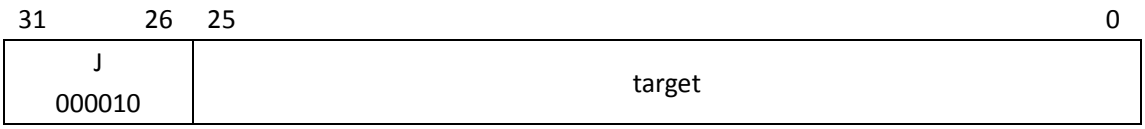
LLbit ← 0

例外：

协处理器不可用例外

3.11.3.29 J

J Jump **J**



格式：J target

描述：

26 位目标地址左移两位，并与延时槽指令地址的高位地址进行拼接产生 32 位目标地址。程序在延迟一条指令后无条件转移到目标地址处。

操作：

T: temp ← target

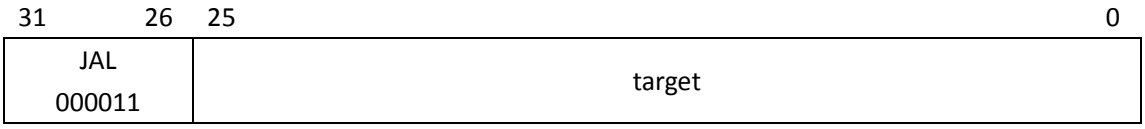
T+1: PC ← $PC_{31...28}$ || temp || 0^2

例外：

无

3.11.3.30 JAL

JAL Jump And Link **JAL**



格式：JAL target

描述：

26 位目标地址左移两位，并与延时槽指令地址的高位地址进行拼接产生 32 位目标地址。程序在延迟一条指令后无条件转移到目标地址处，同时延时槽指令的下一条指令地址无条件地被保存到链接寄存器 r31 中。

操作：

T: temp ← target

GPR[31] ← PC + 8

T+1: PC ← $PC_{31...28}$ || temp || 0^2

例外：

无

3.11.3.31 JALR

JALR Jump And Link Register **JALR**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	00000	rd	00000	JALR 001001
6	5	5	5	5	6

格式：JALR rs

JALR rd, rs

描述：

程序在延迟一条指令后无条件转移到通用寄存器 rs 指向的地址，同时延时槽指令的下一条指令地址被保存到通用寄存器 rd 中。在汇编程序中如果不指定 rd 寄存器，则默认值为 r31。

寄存器描述符 rs、rd 不能指向相同的寄存器，否则再次执行该指令时，执行结果将出现不同。另一方面，如果出现这种情况，系统却无法检测到，因此应当避免出现这种情况。

由于指令必须字对齐，因此 JALR 指令中目标寄存器 rs 的值的低两位必须为“零”，否则从目标地址取指时将发生地址错误例外。

操作：

T: temp ← GPR [rs]

GPR[rd] ← PC + 8

T+1: PC ← temp

例外：

无

3.11.3.32 JR

JR Jump Register **JR**

31 26	25 21	20 6	5 0
SPECIAL 000000	rs	0 0000 0000 0000 0000	JR 001000
6	5	15	6

格式：JR rs

描述：

程序在延迟一条指令后无条件转移到通用寄存器 rs 指向的地址。

由于指令必须字对齐，因此 JR 指令中目标寄存器 rs 的值的低两位必须为“零”，否则从目标地址取指时将发生地址错误例外。

操作：

T: temp ← GPR [rs]

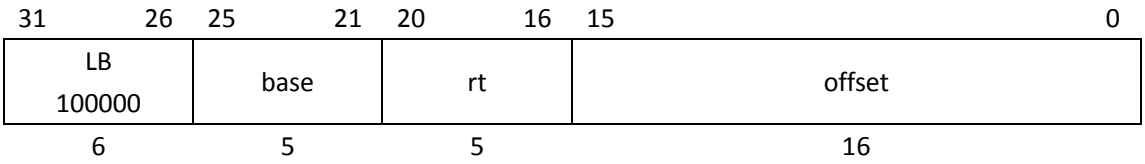
T+1: PC ← temp

例外：

无

3.11.3.33 LB

LB Load Byte **LB**



格式：LB rt, offset(base)

描述：

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成虚拟地址。该地址经转换后产生有效的物理地址，根据物理地址从存储器中读取一个字节，并对字节进行符号扩展后装载到通用寄存器 rt 中。

操作：

T: vAddr ← $((offset_{15})^{16}$ || $offset_{15...0})$ + GPR[base]

(pAddr, uncached) ← AddressTranslation (vAddr, DATA)

mem ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)

byte ← $vAddr_{1...0}$

GPR[rt] ← $(mem_{7+8*byte})^{24}$ || $mem_{7+8*byte...8*byte}$

例外：

TLB 重填例外

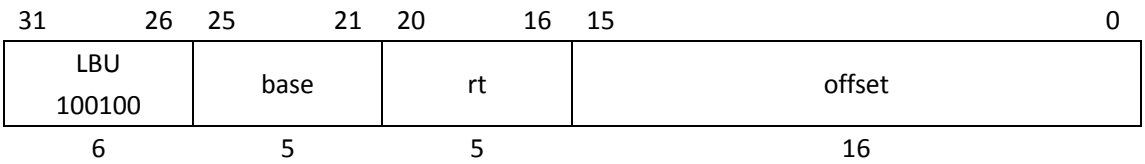
TLB 无效例外

总线错误例外

地址错误例外

3.11.3.34 LBU

LBU Load Byte Unsigned **LBU**



格式：LBU rt, offset(base)

描述：

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成虚拟地址。该地址经转换后产生有效的物理地址，根据物理地址从存储器中读取一个字节，并对字节进行零扩展后装载到通用寄存器 rt 中。

操作：

T: vAddr ← ((offset$_{15}$)^{16} || offset$_{15...0}$) + GPR[base]

(pAddr, uncached) ← AddressTranslation (vAddr, DATA)

mem ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)

byte ← vAddr$_{1...0}$

GPR[rt] ← 0^{24} || mem$_{7+8*byte...8*byte}$

例外：

TLB 重填例外

TLB 无效例外

总线错误例外

地址错误例外

3.11.3.35 LH

LH Load Halfword **LH**

31 26	25 21	20 16	15 0
LH 100001	base	rt	offset
6	5	5	16

格式：LH rt, offset(base)

描述：

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成虚拟地址。该地址经转换后产生有效的物理地址，根据物理地址从存储器中读取半字，并对半字进行符号扩展后装载到通用寄存器 rt 中。

如果有效物理地址的最低位不为“零”，则产生一个地址错误例外。

操作：

T: vAddr ← ((offset$_{15}$)^{16} || offset$_{15...0}$) + GPR[base]

(pAddr, uncached) ← AddressTranslation (vAddr, DATA)

mem ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)

byte ← vAddr$_{1...0}$

GPR[rt] ← (mem$_{15+8*byte}$)^{16} || mem$_{15+8*byte...8*byte}$

例外：

TLB 重填例外

TLB 无效例外

总线错误例外

地址错误例外

3.11.3.36 LHU

LHU Load Halfword Unsigned **LHU**

31 26	25 21	20 16	15 0

LHU 100101	base	rt	offset
6	5	5	16

格式：LHU rt, offset(base)

描述：

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成虚拟地址。该地址经转换后产生有效的物理地址，根据物理地址从存储器中读取半字，并对半字进行零扩展后装载到通用寄存器 rt 中。

如果有效物理地址的最低位不为“零”，则产生一个地址错误例外。

操作：

T: vAddr ← $((offset_{15})^{16} || offset_{15...0}) + GPR[base]$

(pAddr, uncached) ← AddressTranslation (vAddr, DATA)

mem ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)

byte ← $vAddr_{1...0}$

GPR[rt] ← $0^{16} || mem_{15+8*byte...8*byte}$

例外：

TLB 重填例外

TLB 无效例外

总线错误例外

地址错误例外

3.11.3.37 LL

LL Load Linked **LL**

31 26	25 21	20 16	15 0
LL 110000	base	rt	offset
6	5	5	16

格式：LL rt, offset(base)

描述：

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成虚拟地址。该地址经转换后产生有效的物理地址，根据物理地址从存储器中读取一个字装载到通用寄存器 rt 中。

通过 LL 与 SC 指令，处理器可以检查访问的字是否被其他处理器或设备修改过，从而实现对存储器中某个存储位置的原子操作，具体过程如下：

```
L1:
    LL T1, (T0)
    ADD T2, T1, 1
    SC T2, (T0)
    BEQ T2, 0, L1
    NOP
```

上述指令序列实现对 T0 指向的字进行原子加“一”操作，将 ADD 指令修改为 OR 指令则可以实现原子置位操作。该指令可在用户模式下使用，不需要 CP0 使能。

对 uncacheable 地址进行 LL 操作，或者在多处理器系统中对 noncoherent 地址进行 LL

操作，则操作结果将会不可预知。LL 和 SC 指令之间的 cache 缺失可能会引起 SC 操作失败，因此在 LL 和 SC 指令之间不应该有 load/store 操作，否则 SC 操作可能会永远都不能成功。例外也会引起 SC 操作失败，因此应当避免持续不断发生例外的情况。如果有效物理地址的最低两位不为“零”，则产生地址错误例外。

操作：

T: vAddr ← $((offset_{15})^{16}$ || $offset_{15...0})$ + GPR[base]
(pAddr, uncached) ← AddressTranslation (vAddr, DATA)
mem ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
GPR[rt] ← $mem_{31...0}$
LLbit ← 1

例外：

TLB 重填例外
TLB 无效例外
总线错误例外
地址错误例外

3.11.3.38 LUI

LUI Load Upper Immediate **LUI**

31 26	25 21	20 16	15 0
LUI 001111	0 00000	rt	immediate
6	5	5	16

格式：LUI rt, immediate

描述：

16 位的立即数左移十六位，低位补“零”后存入通用寄存器 rt。

操作：

T: GPR[rt] ← immediate || 0^{16}

例外：

无

3.11.3.39 LW

LW Load Word **LW**

31 26	25 21	20 16	15 0
LW 100011	base	rt	offset
6	5	5	16

格式：LW rt, offset(base)

描述：

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成虚拟地址。该地址经转换

后产生有效的物理地址，根据物理地址从存储器中一个字装载到通用寄存器 rt 中。如果有效物理地址的最低两位不为“零”，则产生地址错误例外。

操作：

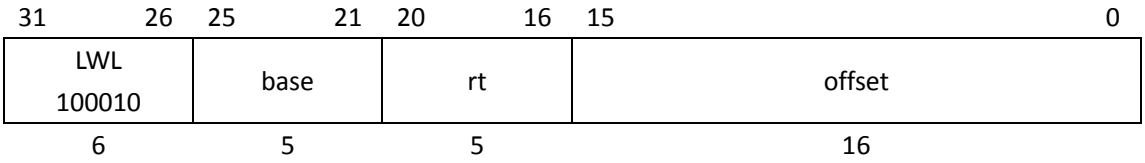
T: vAddr ← ((offset$_{15}$)^{16} || offset$_{15...0}$) + GPR[base]
(pAddr, uncached) ← AddressTranslation (vAddr, DATA)
mem ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
GPR[rt] ← mem$_{31...0}$

例外：

TLB 重填例外
TLB 无效例外
总线错误例外
地址错误例外

3.11.3.40 LWL

LWL Load Word Left **LWL**



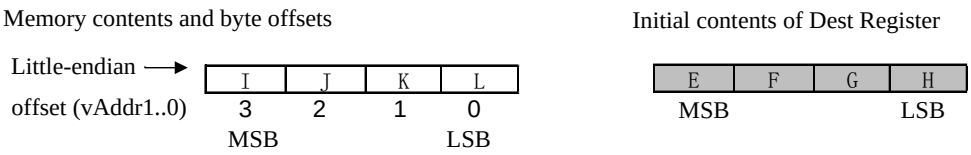
格式：LWL rt, offset(base)

描述：

该指令同 LWR 组合后可以读取内存中非字边界对齐的连续四个字节。将内存中任意地址开始的字分为高位和低位两部分，LWL 将高位部分装载到寄存器 rt 的左边部分，而 LWR 将低位部分装载到 rt 寄存器右边部分。

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成指向任意字节的虚拟地址，该地址经转换后产生有效的物理地址，根据物理地址从存储器中读取一个字范围内从指定字节开始的若干字节。根据指定的不同起始字节地址，读取的数据可能从一个到四个字节不等。

LWL 指令从存储器中指定位置开始读取数据装载到通用寄存器 rt 的高位部分，直到将存储器中一个字的最低字节读出为止，寄存器 rt 的低位部分保持不变。LWL 指令装载数据的情况如下图所示。



Destination register contents after instruction (shaded is unchanged)

vAddr1..0				
0	L	F	G	H
1	K	L	G	H
2	J	K	L	H
3	I	J	K	L

处理器内部对通用寄存器 rt 的值进行了旁路处理，因此当一条内存装载指令后立即跟随一条 LWL 指令，并且两条指令指向相同的 rt 寄存器时，不需要在两条指令中间加入 NOP

操作。LWL 指令不会因为地址不对齐而发生地址错误例外。

操作：

T: vAddr ← ((offset$_{15}$)^{16} || offset$_{15...0}$) + GPR[base]
(pAddr, uncached) ← AddressTranslation (vAddr, DATA)
pAddr ← pAddr$_{PSIZE-1...2}$ || 0^2
byte ← vAddr$_{1...0}$
mem ← LoadMemory (uncached, 0 || byte, pAddr, vAddr, DATA)
temp ← mem$_{8*byte+7...0}$ || GPR[rt]$_{23-8*byte...0}$
GPR[rt] ← temp

例外：

TLB 重填例外
TLB 无效例外
总线错误例外
地址错误例外

3.11.3.41 LWR

LWR Load Word Right **LWR**

31 26	25 21	20 16	15 0
LWR 100110	base	rt	offset
6	5	5	16

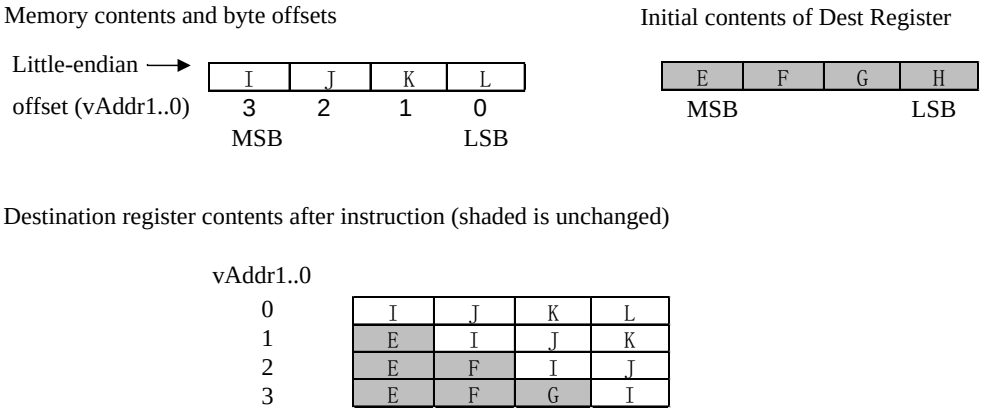
格式：LWR rt, offset(base)

描述：

该指令同 LWL 组合后可以读取内存中非字边界对齐的连续四个字节。将内存中任意地址开始的字分为高位和低位两部分，LWR 将低位部分装载到 rt 寄存器右边部分，而 LWL 将高位部分装载到寄存器 rt 的左边部分。

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成指向任意字节的虚拟地址，该地址经转换后产生有效的物理地址，根据物理地址从存储器中读取一个字范围内从指定字节开始的若干字节。根据指定的不同起始字节地址，读取的数据可能从一个到四个字节不等。

LWR 指令从存储器中指定位置开始读取数据装载到通用寄存器 rt 的低位部分，直到将存储器中一个字的最高字节读出为止，寄存器 rt 的高位部分保持不变。LWR 指令装载数据的情况如下图所示。



处理器内部对通用寄存器 rt 的值进行了旁路处理，因此当一条内存装载指令后立即跟随一条 LWR 指令，并且两条指令指向相同的 rt 寄存器时，不需要在两条指令中间加入 NOP 操作。该指令不会因为地址不对齐而发生地址错误例外。

操作：

T: vAddr ← $((offset_{15})^{16}$ || $offset_{15...0})$ + GPR[base]

(pAddr, uncached) ← AddressTranslation (vAddr, DATA)

pAddr ← $pAddr_{PSIZE-1...2}$ || 0^2

byte ← $vAddr_{1...0}$

mem ← LoadMemory (uncached, 0 || byte, pAddr, vAddr, DATA)

temp ← $GPR[rt]_{31...32-8*byte}$ || $mem_{31...8*byte}$

GPR[rt] ← temp

例外：

TLB 重填例外

TLB 无效例外

总线错误例外

地址错误例外

3.11.3.42 MFC0

MFC0 Move From System Control Coprocessor **MFC0**

31 26	25 21	20 16	15 11	10 0
COP0 010000	MF 00000	rt	rd	0 000 0000 0000
6	5	5	5	11

格式：MFC0 rt, rd

描述：

读协处理器 CP0 的 rd 寄存器值到通用寄存器 rt 内。

操作：

T: data ← CPR[0,rd]

T+1: GPR[rt] ← data

例外：

协处理器不可用例外

3.11.3.43 MFHI

MFHI Move From HI **MFHI**

31 26	25 16	15 11	10 6	5 0
SPECIAL 000000	0 00 0000 0000	rd	0 00000	MFHI 010000
6	10	5	5	6

格式：MFHI rd

描述：

读特殊寄存器 HI 的值到通用寄存器 rd 中。

操作：

T: GPR[rd] ← HI

例外：

无

3.11.3.44 MFLO

MFLO Move From Lo **MFLO**

31 26	25 16	15 11	10 6	5 0
SPECIAL 000000	0 00 0000 0000	rd	0 00000	MFLO 010010
6	10	5	5	6

格式：MFLO rd

描述：

读特殊寄存器 LO 的值到通用寄存器 rd 中。

操作：

T: GPR[rd] ← LO

例外：

无

3.11.3.45 MTC0

MTC0 Move To System Control Coprocessor **MTC0**

31 26	25 21	20 16	15 11	10 0
COP0 010000	MT 00100	rt	rd	0 000 0000 0000
6	5	5	5	11

格式：MTC0 rt, rd

描述：

将通用寄存器 rt 的值写入到协处理器 CP0 的 rd 寄存器中。

操作：

T: data ← GPR[rt]

T+1: CPR[0,rd] ← data

例外：

协处理器不可用例外

3.11.3.46 MTHI

MTHI Move To HI **MTHI**

31 26	25 21	20 6	5 0
SPECIAL 000000	rs	0 000 0000 0000 0000	MTHI 010001
6	5	15	6

格式：MTHI rs

描述：

将通用寄存器 rs 的值写入到特殊寄存器 HI 中。

操作：

T: HI ← GPR[rs]

例外：

无

3.11.3.47 MTLO

MTLO Move To LO **MTLO**

31 26	25 21	20 6	5 0
SPECIAL 000000	rs	0 000 0000 0000 0000	MTLO 010011
6	5	15	6

格式：MTLO rs

描述：

将通用寄存器 rs 的值写入到特殊寄存器 LO 中。

操作：

T: LO ← GPR[rs]

例外：

无

3.11.3.48 MULT

• **MULT** Multiply **MULT**

31 26	25 21	20 16	15 6	5 0

SPECIAL 000000	rs	rt	0 00 0000 0000	MULT 011000
6	5	5	10	6

格式：MULT rs, rt

描述：

对通用寄存器 rs 和 rt 的值进行 2 的补码乘法运算。该指令在任何情况下都不会产生整数溢出例外。乘法操作完成后，将双字结果中的高低两个字分别保存到特殊寄存器 HI 和 LO 中。

操作：

T: t ← GPR[rs] * GPR[rt]

LO ← $t_{31...0}$

HI ← $t_{63...32}$

例外：

无

3.11.3.49 MULTU

MULTU Multiply Unsigned **MULTU**

31 26	25 21	20 16	15 6	5 0
SPECIAL 000000	rs	rt	0 00 0000 0000	MULTU 011001
6	5	5	10	6

格式：MULTU rs, rt

描述：

对通用寄存器 rs 和 rt 的值进行无符号数乘法运算。该指令在任何情况下都不会产生整数溢出例外。乘法操作完成后，将双字结果中的高低两个字分别保存到特殊寄存器 HI 和 LO 中。

操作：

T: t ← (0 || GPR[rs]) * (0 || GPR[rt])

LO ← $t_{31...0}$

HI ← $t_{63...32}$

例外：

无

3.11.3.50 NOR

NOR Nor **NOR**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	NOR 100111
6	5	5	5	5	6

格式：NOR rd, rs, rt

描述：

通用寄存器 rs 的值与通用寄存器 rt 的值按位逻辑“或非”运算，结果存放到通用寄存器 rd 中。

操作：

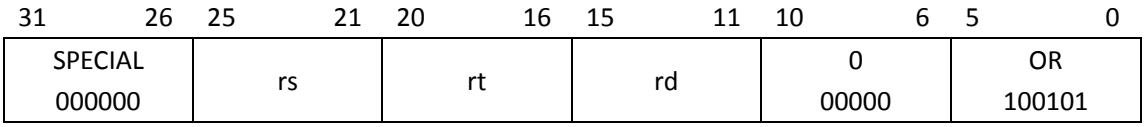
T: GPR[rd]←GPR[rs] nor GPR[rt]

例外：

无

3.11.3.51 OR

OR Or **OR**



格式：OR rd, rs, rt

描述：

通用寄存器 rs 的值与通用寄存器 rt 的值按位逻辑“或”运算，结果存放到通用寄存器 rd 中。

操作：

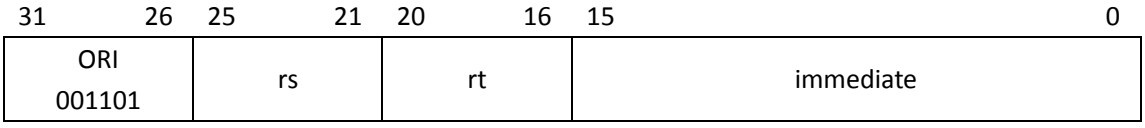
T: GPR[rd]←GPR[rs] or GPR[rt]

例外：

无

3.11.3.52 ORI

ORI Or Immediate **ORI**



格式：ORI rt, rs, immediate

描述：

零扩展 16 位立即数并与通用寄存器 rs 的值按位逻辑“或”运算，结果存放到通用寄存器 rt 中。

操作：

T: GPR[rt]←GPR[rs]$_{31...16}$|| (immediate or GPR[rs] $_{15...0}$)

例外：

无

3.11.3.53 SB

SB Store Byte **SB**

31 26	25 21	20 16	15 0
SB 101000	base	rt	offset
6	5	5	16

格式：SB rt, offset(base)

描述：

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成虚拟地址。将通用寄存器 rt 的最低字节存储到虚拟地址转换得到的有效物理地址中。

操作：

T: vAddr ← $((offset_{15})^{16}$ || $offset_{15...0})$ + GPR[base]
(pAddr, uncached) ← AddressTranslation (vAddr, DATA)
byte ← $vAddr_{1...0}$
data ← $GPR[rt]_{31-8*byte...0}$ || 0^{8*byte}
StoreMemory (uncached, BYTE, data, pAddr, vAddr, DATA)

例外：

TLB 重填例外
TLB 无效例外
TLB 修改例外
总线错误例外
地址错误例外

3.11.3.54 SC

SC Store Conditional **SC**

31 26	25 21	20 16	15 0
SC 111000	base	rt	offset
6	5	5	16

格式：SC rt, offset(base)

描述：

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成虚拟地址。将通用寄存器 rt 的值有条件地存储到虚拟地址转换得到的有效物理地址中。

如果前一个 LL 指令之后任何其他处理器或设备修改过物理地址中的值，或在 LL 和 SC 指令之间执行过 ERET 指令，则 SC 指令将执行失败，不会向物理地址中写入数据。

SC 执行成功与否可在执行完该指令后由通用寄存器 rt 的值反映出来，如果 SC 执行成功，即向物理地址中成功写入数据，则 rt 寄存器值设置为“一”，否则 rt 寄存器值设置为“零”。

如果 SC 指令的存储地址不同于最近一次 LL 指令的访问地址，则 SC 指令操作结果不可预知。

该指令可在用户模式下使用，不需要使能 CP0 协处理器。

如果有效物理地址的最低位两位非“零”，则产生地址错误例外。

执行 SC 指令时，如果同时出现执行失败与产生例外的情况，则优先产生例外。

操作：

```
T:   vAddr ← ((offset15)16 || offset15...0) + GPR[base]
     (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
     data ← GPR[rt]
     if LLbit then
          StoreMemory (uncached, WORD, data, pAddr, vAddr, DATA)
     endif
     GPR[rt] ← 031 || LLbit
```

例外：

TLB 重填例外

TLB 无效例外

TLB 修改例外

总线错误例外

地址错误例外

3.11.3.55 SDBBP

SDBBP Software Debug Breakpoint **SDBBP**

31–26	25–6	5–0
SPECIAL2 011100	code	SDBBP 111111
6	20	6

格式：SDBBP

描述：

执行 SDBBP 指令时产生调试断点例外，并立即无条件地转移到调试例外处理程序。

code 域可用于软件传递参数，在例外处理程序中只有读取该指令的值才能提取 code 域。

操作：

```
T: DebugBreakpointException;
```

例外：

调试断点例外

3.11.3.56 SH

SH Store Halfword **SH**

31–26	25–21	20–16	15–0
SH 101001	base	rt	offset
6	5	5	16

格式：SH rt, offset(base)

描述：

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成虚拟地址。将通用寄存器 rt 的低半字存储到虚拟地址转换得到的有效物理地址中。如果有效物理地址的最低位不等于“零”，则产生地址错误例外。

操作：

T: vAddr ← ((offset$_{15}$)^{16} || offset$_{15...0}$) + GPR[base]

(pAddr, uncached) ← AddressTranslation (vAddr, DATA)

byte ← vAddr$_{1...0}$

data ← GPR[rt]$_{31-8*byte...0}$ || 0^{8*byte}

StoreMemory (uncached, HALFWORD, data, pAddr, vAddr, DATA)

例外：

TLB 重填例外

TLB 无效例外

TLB 修改例外

总线错误例外

地址错误例外

3.11.3.57 SLL

SLL Shift Left Logical **SLL**

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 000000	0 00000	rt	rd	sa	SLL 000000
6	5	5	5	5	6

格式：SLL rd, rt, sa

描述：

将通用寄存器 rt 的数据左移 sa 位，并对低位补“零”。移位结果存放到通用寄存器 rd 中。

操作：

T: GPR[rd] ← GPR[rt]$_{31-sa...0}$ || 0^{sa}

例外：

无

3.11.3.58 SLLV

SLLV Shift Left Logical Variable **SLLV**

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 000000	rs	rt	rd	0 00000	SLLV 000100
6	5	5	5	5	6

格式：SLLV rd, rt, rs

描述：

由通用寄存器 rs 的低五位指定移位量对通用寄存器 rt 的值进行左移，并对低位补“零”。移位结果存放到通用寄存器 rd 中。

操作：

T: s ← GP[rs]$_{4...0}$

GPR[rd] ← GPR[rt]$_{(31-s)...0}$ || 0^{s}

例外：

无

3.11.3.59 SLT

SLT Set On Less Than **SLT**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SLT 101010
6	5	5	5	5	6

格式：SLT rd, rs, rt

描述：

通用寄存器 rs 的值减去通用寄存器 rt 的值，其中两个寄存器值均看作有符号整数。如果通用寄存器 rs 的值小于通用寄存器 rt 的值，则通用寄存器 rd 被置“一”，否则置“零”。

该指令在任何情况下都不会产生整数溢出例外，即便减法运算产生溢出，该指令的比较结果仍然有效。

操作：

T: if GPR[rs] < GPR[rt] then

GPR[rd] ← 0^{31} || 1

else

GPR[rd] ← 0^{32}

endif

例外：

无

3.11.3.60 SLTI

SLTI Set On Less Than Immediate **SLTI**

31 26	25 21	20 16	15 0
SLTI 001010	rs	rt	immediate
6	5	5	16

格式：SLTI rt, rs, immediate

描述：

通用寄存器 rs 的值减去 16 位立即数符号扩展后的值，其中两个数据均看作有符号整数。如果通用寄存器 rs 的值小于符号扩展后的立即数，则通用寄存器 rt 被置“一”，否则置“零”。

该指令在任何情况下都不会产生整数溢出例外，即便减法运算产生溢出，该指令的比较结果仍然有效。

操作：

```
T: if GPR[rs] < (immediate15)^16 || immediate15...0 then
                GPR[rd] ← 0^31 || 1
   else
                GPR[rd] ← 0^32
   endif
```

例外：

无

3.11.3.61 SLTIU

SLTIU Set On Less Than Immediate Unsigned **SLTIU**

31 26	25 21	20 16	15 0
SLTIU 001011	rs	rt	immediate
6	5	5	16

格式：SLTIU rt, rs, immediate

描述：

通用寄存器 rs 的值减去 16 位立即数符号扩展后的值，其中两个数据均看作无符号整数。如果通用寄存器 rs 的值小于符号扩展后的立即数，则通用寄存器 rt 被置“一”，否则置“零”。

该指令在任何情况下都不会产生整数溢出例外，即便减法运算产生溢出，该指令的比较结果仍然有效。

操作：

```
T: if (0 || GPR[rs]) < (immediate15)^16 || immediate15...0 then
                GPR[rd] ← 0^31 || 1
   else
                GPR[rd] ← 0^32
   endif
```

例外：

无

3.11.3.62 SLTU

SLTU Set On Less Than Unsigned **SLTU**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SLTU 101011

6	5	5	5	5	6

格式：SLTU rd, rs, rt

描述：

通用寄存器 rs 的值减去通用寄存器 rt 的值，其中两个寄存器值均看作无符号整数。如果通用寄存器 rs 的值小于通用寄存器 rt 的值，则通用寄存器 rd 被置“一”，否则置“零”。

该指令在任何情况下都不会产生整数溢出例外，即便减法运算产生溢出，该指令的比较结果仍然有效。

操作：

```
T: if (0 || GPR[rs]) < 0 || GPR[rt] then
                GPR[rd] ← 0^31 || 1
   else
                GPR[rd] ← 0^32
   endif
```

例外：

无

3.11.3.63 SRA

SRA Shift Right Arithmetic **SRA**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	0 00000	rt	rd	sa	SRA 000011
6	5	5	5	5	6

格式：SRA rd, rt, sa

描述：

将通用寄存器 rt 的值右移 sa 位，并对高位进行符号扩展。移位结果存放到通用寄存器 rd 中。

操作：

T: GPR[rd] ← $(GPR[rt]_{31})^{sa}$ || $GPR[rt]_{31...sa}$

例外：

无

3.11.3.64 SRAV

SRAV Shift Right Arithmetic Variable **SRAV**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SRAV 000111
6	5	5	5	5	6

格式：SRAV rd, rt, rs

描述：

由通用寄存器 rs 的低五位指定移位量对通用寄存器 rt 的值进行右移，并对高位进行符号扩展。移位结果存放到通用寄存器 rd 中。

操作：

T: s ← $GP[rs]_{4...0}$

$GPR[rd] \leftarrow (GPR[rt]_{31})^{s} \,||\, GPR[rt]_{31...s}$

例外：

无

3.11.3.65 SRL

SRL Shift Right Logical **SRL**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	0 00000	rt	rd	sa	SRL 000010
6	5	5	5	5	6

格式：SRL rd, rt, sa

描述：

将通用寄存器 rt 的值右移 sa 位，并对高位补“零”。运算结果存放到通用寄存器 rd 中。

操作：

T: $GPR[rd] \leftarrow 0^{sa} \,||\, GPR[rt]_{31...sa}$

例外：

无

3.11.3.66 SRLV

SRLV Shift Right Logical Variable **SRLV**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SRLV 000110
6	5	5	5	5	6

格式：SRLV rd, rt, rs

描述：

由通用寄存器 rs 的低五位指定移位量对通用寄存器 rt 的值进行右移，并对高位补“零”。移位结果存放到通用寄存器 rd 中。

操作：

T: s ← $GPR[rs]_{4...0}$

$GPR[rd] \leftarrow 0^{s} \,||\, GPR[rt]_{31...s}$

例外：

无

3.11.3.67 SUB

SUB Subtract **SUB**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SUB 100010
6	5	5	5	5	6

格式：SUB rd, rs, rt

描述：

通用寄存器 rs 的值减去通用寄存器 rt 的值，结果存放到通用寄存器 rd 中。

SUB 指令与 SUBU 指令的不同之处在于 SUBU 指令不会产生溢出例外。

对于 SUB 指令，如果相减操作后第 30 位与第 31 位的进位输出值不同，则产生一个整数溢出例外（即 2 的补码溢出），此时目标寄存器 rd 的值不会被更新。

操作：

T: GPR[rd] ← GPR[rs] – GPR[rt]

例外：

整数溢出例外

3.11.3.68 SUBU

SUBU Subtract Unsigned **SUBU**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SUBU 100011
6	5	5	5	5	6

格式：SUBU rd, rs, rt

描述：

通用寄存器 rs 的值减去通用寄存器 rt 的值，结果存放到通用寄存器 rd 中。

SUBU 指令与 SUB 指令的不同之处在于 SUBU 指令不会产生溢出例外。SUBU 指令在任何情况下都不会产生整数溢出例外。

操作：

T: GPR[rd] ← GPR[rs] – GPR[rt]

例外：

无

3.11.3.69 SW

SW Store Word **SW**

31 26	25 21	20 16	15 0
SW	base	rt	offset

101011			
6	5	5	16

格式：SW rt, offset(base)

描述：

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成虚拟地址。将通用寄存器 rt 的值存储到虚拟地址转换得到的有效物理地址中。

如果有效物理地址的最低两位不等于“零”，则产生地址错误例外。

操作：

T: vAddr ← $((offset_{15})^{16}$ || $offset_{15...0})$ + GPR[base]

(pAddr, uncached) ← AddressTranslation (vAddr, DATA)

data ← GPR[rt]

StoreMemory (uncached, WORD, data, pAddr, vAddr, DATA)

例外：

TLB 重填例外

TLB 无效例外

TLB 修改例外

总线错误例外

地址错误例外

3.11.3.70 SWL

SWL Store Word Left **SWL**

31 26	25 21	20 16	15 0
SWL 101010	base	rt	offset
6	5	5	16

格式：SWL rt, offset(base)

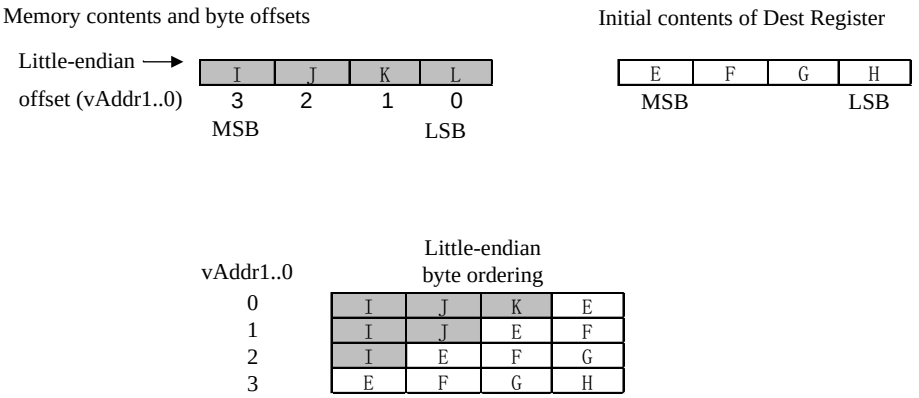
描述：

该指令与 SWR 组合后可以将寄存器值存储到内存中非字边界对齐的连续四个字节中。SWL 将寄存器的左边部分存储到内存中一个字的高位部分，SWR 将寄存器的右边部分存储到内存中一个字的低位部分。

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成指向任意字节的虚拟地址，该地址经转换后产生有效的物理地址，根据物理地址向存储器中包含该起始字节的字中写入数据。根据指定的不同起始字节地址，写入的数据可能从一个到四个字节不等。

SWL 指令从寄存器的最高字节开始读取数据写入到存储器中一个字的低位部分，直到向存储器中一个字的最低字节写入数据为止。SWL 指令存储数据的情况如下图所示。

该指令不会因为地址不对齐而发生地址错误例外。



操作：

T: vAddr ← ((offset$_{15}$)^{16} || offset $_{15...0}$) + GPR[base]

(pAddr, uncached) ← AddressTranslation (vAddr, DATA)

pAddr ← pAddr$_{31...2}$ || 0^2

byte ← vAddr$_{1...0}$

data ← 0$^{24-8*byte}$ || GPR[rt]$_{31...24-8*byte}$

Storememory (uncached, byte, data, pAddr, vAddr, DATA)

例外：

TLB 重填例外

TLB 无效例外

TLB 修改例外

总线错误例外

地址错误例外

3.11.3.71 SWR

SWR Store Word Right **SWR**

31 26	25 21	20 16	15 0
SWR 101110	base	rt	offset
6	5	5	16

格式：SWR rt, offset(base)

描述：

该指令与 SWL 组合后可以将寄存器值存储到内存中非字边界对齐的连续四个字节中。SWR 将寄存器的右边部分存储到内存中一个字的低位部分，SWL 将寄存器的左边部分存储到内存中一个字的高位部分。

符号扩展 16 位的偏移量，并与通用寄存器 base 的值相加生成指向任意字节的虚拟地址，该地址经转换后产生有效的物理地址，根据物理地址向存储器中包含该起始字节的字中写入数据。根据指定的不同起始字节地址，写入的数据可能从一个到四个字节不等。

SWR 指令从寄存器的最低字节开始读取数据写入到存储器中一个字的高位部分，直到向存储器中一个字的最高字节写入数据为止。SWR 指令存储数据的情况如下图所示。

该指令不会因为地址不对齐而发生地址错误例外。

vAddr1..0	Little-endian byte ordering			
0	E	F	G	H
1	F	G	H	L
2	G	H	K	L
3	H	J	K	L

操作：

T: vAddr ← ((offset$_{15}$)^{16} || offset $_{15...0}$) + GPR[base]

(pAddr, uncached) ← AddressTranslation (vAddr, DATA)

pAddr ← pAddr$_{31...2}$ || 0^2

byte ← vAddr$_{1...0}$

data ← GPR[rt]$_{31-8*byte...0}$ || 0^{8*byte}

StoreMemory(uncached, WORD-byte, data, pAddr, vAddr, DATA)

例外：

TLB 重填例外

TLB 无效例外

TLB 修改例外

总线错误例外

地址错误例外

3.11.3.72 SYNC

SYNC Synchronize **SYNC**

31 26	25 6	5 0
SPECIAL	0	SYNC
000000	0000 0000 0000 0000 0000	001111
6	20	6

格式：SYNC

描述：

在 32 位龙芯处理器中，SYNC 指令的作用等同与一条 NOP 指令。

操作：

T: SyncOperation()

例外：

无

3.11.3.73 SYSCALL

SYSCALL System Call **SYSCALL**

31 26	25 6	5 0
SPECIAL 000000	code	SYSCALL 001100
6	20	6

格式：SYSCALL

描述：

执行 SYSCALL 指令时产生系统调用例外，并立即无条件地转移到例外处理程序。

code 域可用于软件传递参数，在例外处理程序中只有读取该指令的值才能提取 code 域。

操作：

T: SystemCallException

例外：

系统调用例外

3.11.3.74 TEQ

TEQ Trap If Equal **TEQ**

31 26	25 21	20 16	15 6	5 0
SPECIAL 000000	rs	rt	code	TEQ 110100
6	5	5	10	6

格式：TEQ rs, rt

描述：

比较通用寄存器 rs 的值和通用寄存器 rt 的值，如果二者相等，则产生自陷例外。

code 域可用于软件传递参数，在例外处理程序中只有读取该指令的值才能提取 code 域。

操作：

```
T: if GPR[rs] = GPR[rt] then
     TrapException
   endif
```

例外：

自陷例外

3.11.3.75 TEQI

TEQI Trap If Equal Immediate **TEQI**

31 26	25 21	20 16	15 0
REGIMM 000001	rs	TEQI 01100	immediate

6	5	5	16

格式：TEQI rs, immediate

描述：

对 16 位立即数进行符号扩展，并与通用寄存器 rs 的值进行比较。如果二者相等，则产生自陷例外。

操作：

T: if GPR[rs] = $(immediate_{15})^{16}$ || $immediate_{15...0}$ then

TrapException

endif

例外：

自陷例外

3.11.3.76 TGE

TGE Trap If Greater Than Or Equal **TGE**

31 26	25 21	20 16	15 6	5 0
SPECIAL 000000	rs	rt	code	TEG 110000
6	5	5	10	6

格式：TGE rs, rt

描述：

通用寄存器 rs 的值和通用寄存器 rt 的值均看作有符号整数进行比较，如果通用寄存器 rs 的值大于或等于通用寄存器 rt 的值，则产生自陷例外。

code 域可用于软件传递参数，在例外处理程序中只有读取该指令的值才能提取 code 域。

操作：

T: if GPR[rs] ≥ GPR[rt] then

TrapException

endif

例外：

自陷例外

3.11.3.77 TGEI

TGEI Trap If Greater Than Or Equal Immediate **TGEI**

31 26	25 21	20 16	15 0
REGIMM 000001	rs	TGEI 01000	immediate
6	5	5	16

格式：TGEI rs, immediate

描述：

通用寄存器 rs 的值与 16 位立即数符号扩展后的值均看作有符号整数进行比较，如

果通用寄存器 rs 的值大于或者等于 16 位立即数符号扩展后的值，则产生自陷例外。

操作：

```
T: if GPR[rs] ≥ (immediate15)16 || immediate15...0 then
     TrapException
   endif
```

例外：

自陷例外

3.11.3.78 TGEIU

TGEIU Trap If Greater Than Or Equal Immediate Unsigned **TGEIU**

31–26	25–21	20–16	15–0
REGIMM 000001	rs	TGEIU 01001	immediate
6	5	5	16

格式：TGEIU rs, immediate

描述：

通用寄存器 rs 的值与 16 位立即数符号扩展后的值均看作无符号整数进行比较，如果通用寄存器 rs 的值大于或者等于 16 位立即数符号扩展后的值，则产生自陷例外。

操作：

```
T: if (0 || GPR[rs]) ≥ (0 || (immediate15)16 || immediate15...0) then
     TrapException
   endif
```

例外：

自陷例外

3.11.3.79 TGEU

TGEU Trap If Greater Than Or Equal Unsigned **TGEU**

31–26	25–21	20–16	15–6	5–0
SPECIAL 000000	rs	rt	code	TGEU 110001
6	5	5	10	6

格式：TGEU rs, rt

描述：

通用寄存器 rs 的值和通用寄存器 rt 的值均看作无符号整数进行比较，如果通用寄存器 rs 的值大于或等于通用寄存器 rt 的值，则产生自陷例外。

code 域可用于软件传递参数，在例外处理程序中只有读取该指令的值才能提取 code 域。

操作：

```
T: if (0 || GPR[rs]) ≥ (0 || GPR[rt]) then
     TrapException
```

endif

例外：

自陷例外

3.11.3.80 TLBP

TLBP　　Probe TLB For Matching Entry　　TLBP

31 26	25 24	6 5	0
COP0	CO	0	TLBP
010000	1	000 0000 0000 0000 0000	001000
6	1	19	6

格式：TLBP

描述：

EntryHi 寄存器的值与 TLB 表项进行比较，如果匹配某一个 TLB 表项，则将匹配的 TLB 表项地址值写入到 Index 寄存器中；如果不匹配任何 TLB 表项，则 Index 寄存器的最高位置“—”。

操作：

T: Index ← 1 || 0^{26} || undefined^{4}

for i in 0…TLBEntries-1

if ($TLB[i]_{95...77}$ = $EntryHi_{31...12}$) and ($TLB[i]_{76}$ or ($TLB[i]_{74...64}$ = $EntryHi_{7...0}$)) then

Index ← 0^{26} || $i_{4...0}$

endif

endfor

例外：

协处理器不可用例外

3.11.3.81 TLBR

TLBR　　Read Indexed TLB Entry　　TLBR

31 26	25 24	6 5	0
COP0	CO	0	TLBR
010000	1	000 0000 0000 0000 0000	000001
6	1	19	6

格式：TLBR

描述：

从 TLB 中读出 Index 寄存器指向的 TLB 表项，并将 TLB 表项的值写入到 EntryHi 寄存器与 EntryLo 寄存器中，其中 G 域的值（用于控制 ASID 域的比较）同时写入到 EntryLo0 与 EntryLo1 寄存器中。

如果 Index 寄存器的值大于处理器中 TLB 表项的总数目，则该操作无效，执行结果不可预知。

操作：

T: PageMask ← TLB[Index$_{4...0}$]$_{127...96}$

EntryHi ← TLB[Index$_{4...0}$]$_{95...64}$ and not TLB[Index$_{4...0}$]$_{127...96}$

EntryLo1 ← TLB[Index$_{4...0}$]$_{63...32}$

EntryLo0 ← TLB[Index$_{4...0}$]$_{31...0}$

例外：

协处理器不可用例外

3.11.3.82 TLBWI

TLBWI　　Write Indexed TLB Entry　　TLBWI

31 26	25 24	6 5	0
COP0	CO	0	TLBWI
010000	1	000 0000 0000 0000 0000	000010
6	1	19	6

格式：TLBWI

描述：

EntryHi 寄存器与 EntryLo 寄存器的值写入到 Index 寄存器指向的 TLB 表项中，其中 G 域的写入值由 EntryLo0 寄存器与 EntryLo1 寄存器的 G 域进行逻辑“与”操作而得。

如果 Index 寄存器的值大于处理器中 TLB 表项的总数目，则该操作无效，执行结果不可预知。

操作：

T: TLB[Index$_{4...0}$] ← PageMask || (EntryHi and not PageMask) || EntryLo1 || EntryLo0

例外：

协处理器不可用例外

3.11.3.83 TLBWR

TLBWR　　Write Random TLB Entry　　TLBWR

31 26	25 24	6 5	0
COP0	CO	0	TLBWR
010000	1	000 0000 0000 0000 0000	000110
6	1	19	6

格式：TLBWR

描述：

EntryHi 寄存器与 EntryLo 寄存器的值写入到 Random 寄存器指向的 TLB 表项中，其中 G 域的写入值由 EntryLo0 寄存器与 EntryLo1 寄存器的 G 域进行逻辑“与”操作而得。

操作：

T: TLB[Random$_{4...0}$] ← PageMask || (EntryHi and not PageMask) || EntryLo1 || EntryLo0

例外：

协处理器不可用例外

3.11.3.84 TLT

TLT Trap If Less Than **TLT**

31 26	25 21	20 16	15 6	5 0
SPECIAL 000000	rs	rt	code	TLT 110010
6	5	5	10	6

格式：TLT rs, rt

描述：

通用寄存器 rs 的值和通用寄存器 rt 的值均看作有符号整数进行比较，如果通用寄存器 rs 的值小于通用寄存器 rt 的值，则产生自陷例外。

code 域可用于软件传递参数，在例外处理程序中只有读取该指令的值才能提取 code 域。

操作：

```
T: if GPR[rs] < GPR[rt] then
       TrapException
   endif
```

例外：

自陷例外

3.11.3.85 TLTI

TLTI Trap If Less Than Immediate **TLTI**

31 26	25 21	20 16	15 0
REGIMM 000001	rs	TLTI 01010	immediate
6	5	5	16

格式：TLTI rs, immediate

描述：

通用寄存器 rs 的值与 16 位立即数符号扩展后的值均看作有符号整数进行比较，如果通用寄存器的值小于 16 位立即数扩展后的值，则产生自陷例外。

操作：

T: if GPR[rs] < $(immediate_{15})^{16}$ || $immediate_{15...0}$ then
    TrapException
  endif

例外：

自陷例外

3.11.3.86 TLTIU

TLTIU Trap If Less Than Immediate Unsigned **TLTIU**

31 26	25 21	20 16	15 0
REGIMM 000001	rs	TLTIU 01011	immediate
6	5	5	16

格式：TLTIU rs, immediate

描述：

通用寄存器 rs 的值与 16 位立即数扩展后的值均看作无符号整数进行比较，如果通用寄存器值小于 16 位立即数符号扩展后的值，则产生自陷例外。

操作：

```
T: if (0 || GPR[rs]) < (0 || (immediate15)16 || immediate15...0 ) then
       TrapException
   endif
```

例外：

自陷例外

3.11.3.87 TLTU

TLTU Trap If Less Than Unsigned **TLTU**

31 26	25 21	20 16	15 6	5 0
SPECIAL 000000	rs	rt	code	TLTU 110011
6	5	5	10	6

格式：TLTU rs, rt

描述：

通用寄存器 rs 的值和通用寄存器 rt 的值均看作无符号整数进行比较，如果通用寄存器 rs 的值小于通用寄存器 rt 的值，则产生自陷例外。

code 域可用于软件传递参数，在例外处理程序中只有读取该指令的值才能提取 code 域。

操作：

```
T: if (0 || GPR[rs]) < (0 || GPR[rt]) then
       TrapException
   endif
```

例外：

自陷例外

3.11.3.88 TNE

TNE Trap If Not Equal **TNE**

31 26	25 21	20 16	15 6	5 0
SPECIAL 000000	rs	rt	code	TNE 110110
6	5	5	10	6

格式：TNE rs, rt

描述：

通用寄存器 rs 的值和通用寄存器 rt 的值进行比较，如果二者不相等，则产生自陷例外。

code 域可用于软件传递参数，在例外处理程序中只有读取该指令的值才能提取 code 域。

操作：

```
T: if GPR[rs] ≠ GPR[rt] then
     TrapException
   endif
```

例外：

自陷例外

3.11.3.89 TNEI

TNEI Trap If Not Equal Immediate **TNEI**

31 26	25 21	20 16	15 0
REGIMM 000001	rs	TNEI 01110	immediate
6	5	5	16

格式：TNEI rs, immediate

描述：

通用寄存器 rs 的值与 16 位立即数符号扩展后的值进行比较，如果二者不相等，则产生自陷例外。

操作：

T: if GPR[rs] ≠ $(immediate_{15})^{16}$ || $immediate_{15...0}$ then
  TrapException
endif

例外：

自陷例外

3.11.3.90 WAIT

WAIT Wait **WAIT**

31 26	25	24 6	5 0
COP0 010000	CO 1	0 000 0000 0000 0000 0000	WAIT 100000
6	1	19	6

格式：WAIT

描述：

执行 WAIT 指令后，处理器将关闭时钟并进入低功耗的休眠等待状态。在处理器进入休眠等待状态之前，所有在 WAIT 之前的指令都将执行完毕，所有在 WAIT 指令之后的指令都不会被执行。在处理器进入休眠等待状态之后，如果发生中断，则处理器退出休眠等待状态

并恢复正常运行。

操作：

T: EnterWaitMode

例外：

协处理器不可用例外

3.11.3.91 XOR

XOR Exclusive Or **XOR**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	XOR 100110
6	5	5	5	5	6

格式：XOR rd, rs, rt

描述：

通用寄存器 rs 的值与通用寄存器 rt 的值按位逻辑“异或”运算，结果存放到通用寄存器 rd 中。

操作：

T: GPR[rd]←GPR[rs] xor GPR[rt]

例外：

无

3.11.3.92 XORI

XORI Exclusive OR Immediate **XORI**

31 26	25 21	20 16	15 0
XORI 001110	rs	rt	immediate
6	5	5	16

格式：XORI rt, rs, immediate

描述：

零扩展 16 位立即数并与通用寄存器 rs 的值按位逻辑“异或”运算，结果存放到通用寄存器 rt 中。

操作：

T: GPR[rt] ← GPR[rs] xor (0^{16} || immediate)

例外：

无

4 系统控制

4.1 概述

系统控制模块用于管理 GSC3281 芯片中的启动模式、时钟、复位、功耗控制、IO 复用以及模块配置等系统控制功能，是整个芯片的控制中枢。多数情况下，通过软件配置系统控制寄存器即可实现期望的功能，但某些时候硬件上也需要进行一些配套设置，例如通过芯片引脚设置启动模式。

4.2 功能特性

GSC3281 系统控制模块具备如下的功能特性：

- 支持 2 种启动模式
- PLL 输出频率可配置
- 各模块时钟可独立配置
- 所有模块均可进行软件复位
- 软件可根据需要使能或者关闭各模块时钟
- 支持 CPU 休眠模式
- 支持 IO 复用
- 支持对各模块进行系统级配置

4.3 结构框图

系统控制模块作为一个从设备连接到 GSC3281 芯片的 AHB 总线上，如图 4-1 所示，CPU 通过软件对系统控制模块进行配置，从而实现各种系统控制功能。

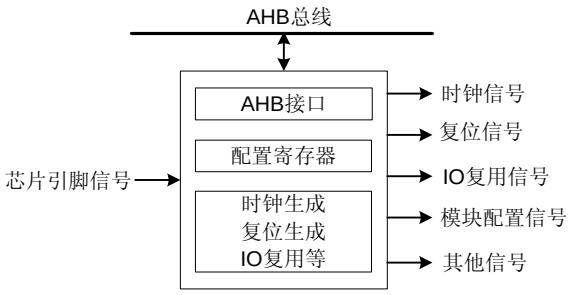


图 4-1 系统控制模块结构框图

4.4 启动模式

为了灵活适应各种应用需求，GSC3281 芯片支持 2 种启动模式：

- NAND Flash 启动
- SPI Flash 启动

通过 GSC3281 芯片的 nfcsn_bootmode0 与 pwmout3_bootmode1_psclk0 两个引脚可以配置芯片的启动模式，具体配置情况如表 4-1 所示。这两个启动模式配置引脚只在复位状态下用于配置启动模式，在 GSC3281 芯片退出复位状态之后，这两个引脚将用作普通的功能引脚。为了将芯片配置为某一种启动模式同时又不影响正常状态的功能，这两个引脚的上拉与下拉电阻必须采用弱上拉或者弱下拉。

表 4-1 GSC3281 启动模式配置

配置引脚 {bootmode1, bootmode0}[注 1]	启动模式
00	保留
01	保留
10	SPI Flash 启动
11	NAND Flash 启动

[注 1]：{bootmode1,bootmode0}={ pwmout3_bootmode1_psclk0, nfcsn_bootmode0}

4.4.1 NAND Flash 启动

通常情况下，NAND Flash 启动方式是 GSC3281 的首选启动方式；为了实现 NAND Flash 启动，在应用系统设计时，必须选用 NAND Flash 颗粒的第一块（block）确保不会出错的 NAND Flash 芯片，目前的 NAND Flash 芯片通常都可以由厂家保证这一点。为了支持 NAND Flash 启动，选用的 NAND Flash 颗粒的页（page）大小必须为 4KB。

GSC3281 芯片的 NAND Flash 控制器具有 4KB 的片内缓冲区，当配置为 NAND Flash 启动模式时，复位之后 NAND Flash 控制器自动把 NAND Flash 颗粒中第一块第一页的数据搬运到 4KB 片内缓冲区中，CPU 将从 4KB 片内缓冲区开始执行启动代码。由于 NAND Flash 控制器只有 4KB 片内缓冲区，因此启动代码必须控制在 4KB 之内。

4.4.2 SPI Flash 启动

SPI Flash 启动提供了一种可选的启动方式，允许应用系统将 GSC3281 芯片的启动代码存放于 SPI 接口的 NOR Flash 中，最大支持 4MB 启动代码；由于 SPI 接口只需要 4 个芯片接口信号，因此 SPI 启动方式可降低系统成本，适用于很多低成本方案。

SPI Flash 启动只能使用 GSC3281 芯片的 SPI1 主机接口，对应的 4 个芯片引脚为 spi1csn、spi1miso、spi1sck 与 spi1mosi；尽管 SPI1 主机接口的片选输出信号可以使用 GPIO 引脚来实现，但在 SPI1 启动时必须使用 spi1csn 引脚的片选信号。

4.5 时钟管理

4.5.1 时钟结构

按照不同的来源，GSC3281 芯片的内部时钟可以分为三类：第一类时钟由芯片外部直接输入提供给相应的模块，例如以太网 MAC 控制器的接收与发送时钟、EJTAG 调试时钟 JTCK 等；第二类时钟由模块本身的专用 PHY 直接提供，例如 USB 的 UTMI 时钟；绝大多数时钟都属于第三类时钟，来源于片内的系统 PLL，可由软件配置分频系数以及是否使能，根据应用

的不同可以进行灵活的配置以满足不同的性能或者功耗需求。GSC3281 芯片各功能模块的时钟概览情况如表 4-2 所示，如前所述，绝大多数时钟都直接或者间接来源于 PLL。

表 4-2 GSC3281 芯片时钟概览

功能	时钟	频率范围/MHZ	默认频率/MHZ	默认状态	时钟源	说明
系统控制	ext_clk	1~50	12	N/A	外部晶振	外部输入的 PLL 参考时钟
	pll_clk	62.5~1500	500	使能	PLL	PLL 输出时钟
	sysctl_clk	<178	166.7	关闭	hclk	系统控制模块工作时钟，频率等于 hclk
	pll_clkout	<100	16.67	使能	pll_clk	PLL 输出时钟的分频输出
总线	aclk	<267	250	使能	pll_clk	AXI 总线时钟，由 pll_clk 二分频得到
	hclk	<178	166.7	使能	pll_clk	AHB 总线时钟
	pclk	<59.3	55.6	使能	hlk	APB 总线时钟
CPU	cpu_clk	<267	250	使能	aclk	CPU 工作时钟
	ejtag_jtck	1~40	N/A	N/A	调试主机	EJTAG 调试时钟
DDR2	ddr2_clk	<267	250	使能	aclk	DDR2 工作时钟，频率等于 aclk
	ddr2phy_clk	<534	500	使能	pll_clk	DDR2 PHY 工作时钟，频率等于 pll_clk
DMA	dma_clk	<178	166.7	关闭	hclk	DMA 控制器工作时钟，频率等于 hclk
NFC	nfc_clk	<178	166.7	关闭	hclk	NAND Flash 控制器工作时钟，频率等于 hclk
MAC	tx_clk	25/2.5	25/2.5	N/A	rmii_clk	MAC 发送时钟
	rx_clk	25/2.5	25/2.5	N/A	rmii_clk	MAC 接收时钟
	rmii_clk	50	50	N/A	外部	MAC RMII 时钟，由外部输入
USB	utmi_clk	60	60	N/A	USB PHY	USB UTMI 接口时钟
	ohci_clk	12	12	使能	USB PHY	USB 片上振荡器产生的 OHCI 接口时钟，该时钟可用作 PLL 的参考时钟
ICTL	ictl_clk	<178	166.7	关闭	hclk	中断控制器工作时钟，频率等于 hclk
SPI1	spi1_clk	<178	5.56	关闭	hclk	SPI1 工作时钟
EMI	emi_clk	<178	166.7	关闭	hclk	EMI 控制器工作时钟，频率等于 hclk
I2C	i2c_clk	<178	16.67	关闭	hclk	I2C 工作时钟
I2S	i2s_clk	1~6.144	1.389	关闭	spll_clk	I2S 工作时钟
	i2s_clkout	1~6.144	1.389	关闭	i2s_clk	I2S 输出时钟，频率等于 i2s_clk
SPI0	spi0_clk	<59.3	5.56	关闭	pclk	SPI0 工作时钟
keypad	keypad_clk	0.002~10	1.11	关闭	pclk	矩阵键盘工作时钟
SCI0	sci0_clk	1~5	4.63	关闭	pclk	SCI0 工作时钟
	sim0_clk	1~5	4.63	关闭	sci0_clk	SCI0 对外输出时钟
SCI1	sci1_clk	1~5	4.63	关闭	pclk	SCI1 工作时钟
	sim1_clk	1~5	4.63	关闭	sci1_clk	SCI1 对外输出时钟
PWM	pwm0_clk	1~20	5.56	关闭	pclk	PWM0 工作时钟
	pwm1_clk	1~20	5.56	关闭	pclk	PWM1 工作时钟
	pwm2_clk	1~20	5.56	关闭	pclk	PWM2 工作时钟
	pwmr_clk	1~50	5.56	关闭	pclk	PWM 旋转编码器工作时钟
PS2_0	ps2_0_clk	1~10	1.11	关闭	pclk	PS2-0 工作时钟
PS2_1	ps2_1_clk	1~10	1.11	关闭	pclk	PS2-1 工作时钟

功能	时钟	频率范围/MHZ	默认频率/MHZ	默认状态	时钟源	说明
GPIO	gpio_clk	10~100	55.6	关闭	pclk	GPIO 工作时钟，频率等于 pclk
	gpiodb_clk	1~50	2.78	关闭	pclk	GPIO 消抖时钟
watchdog	wdt_clk	10~100	55.6	关闭	pclk	Watchdog 工作时钟，频率等于 pclk
timer	timer0_clk	0.01~50	2.78	关闭	pclk	Timer0 工作时钟
	timer1_clk	0.01~50	2.78	关闭	pclk	Timer1 工作时钟
	timer2_clk	0.01~50	2.78	关闭	pclk	Timer2 工作时钟
	timer3_clk	0.01~50	2.78	关闭	pclk	Timer3 工作时钟
UART0	uart0_clk	1~50	5.56	关闭	pclk	UART0 工作时钟
UART1	uart1_clk	1~50	5.56	关闭	pclk	UART1 工作时钟
UART2	uart2_clk	1~50	5.56	关闭	pclk	UART2 工作时钟
UART3	uart3_clk	1~100	16.67	关闭	hclk	UART3 工作时钟
UART4	uart4_clk	1~100	16.67	关闭	hclk	UART4 工作时钟
UART5	uart5_clk	1~100	16.67	关闭	hclk	UART5 工作时钟
UART6	uart6_clk	1~50	5.56	关闭	pclk	UART6 工作时钟
UART7	uart7_clk	1~50	5.56	关闭	pclk	UART7 工作时钟

GSC3281 芯片的时钟结构如图 4-2 所示，图中标示了除 EJTAG 调试时钟之外的所有片内时钟及其相互关系；最高频率时钟为 PLL 时钟，典型情况下为 500MHZ，该时钟直接作为 DDR2 PHY 的时钟；PLL 时钟 2 分频之后作为 AXI 总线时钟以及 CPU、DDR2 控制器的工作时钟，PLL 时钟整数分频之后作为 AHB 总线时钟及若干设备的工作时钟，AHB 总线时钟整数分频之后作为 APB 总线时钟；除 MAC 之外，其他片内设备的时钟均以 PLL 时钟、AHB 总线时钟或者 APB 总线时钟等为源时钟进行分频得到。

为了降低系统成本，PLL 的参考时钟既可以来自于外部引脚时钟，也可以来自 USB PHY 片内振荡器提供的 12MHZ 时钟；由于用于选择参考时钟的信号引脚 clksel 带有下拉电阻，因此默认情况下 GSC3281 芯片将使用 USB PHY 提供的 12MHZ 时钟作为 PLL 参考时钟。

在某些应用中，可能需要由 GSC3281 对外提供一个时钟，图 4-2 中的 pll_clkout 即为 PLL 时钟经过一定分频之后输出的时钟，该时钟一方面可用于对片外设备提供时钟，另一方面也用于分频产生 I2S 接口的工作时钟。

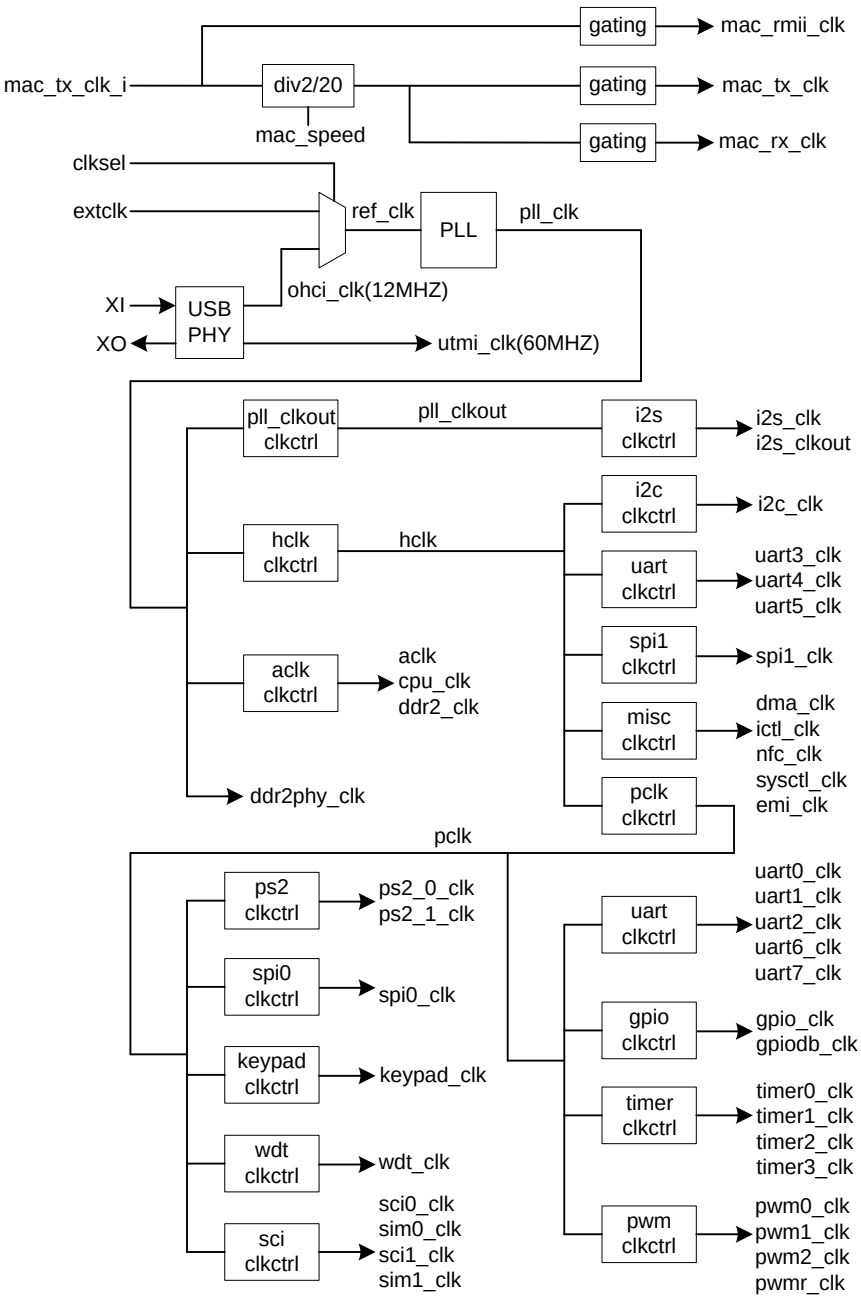


图 4-2 GSC3281 芯片时钟结构概览

USB PHY 内部包含了一个 PLL，用于产生 60MHZ 的 UTMI 时钟，USB 控制器以 UTMI 时钟为工作时钟。

GSC3281 芯片的以太网 MAC 控制器有三个工作时钟，包括发送时钟 mac_tx_clk、接收时钟 mac_rx_clk 以及 RMII 时钟 mac_rmii_clk，其中 mac_tx_clk 和 mac_rx_clk 由 mac_rmii_clk 分频得到，具体请看以太网 MAC 控制器一章的描述。

4.5.2 PLL 配置

GSC3281 芯片采用了高精度片内 PLL，可以为整个芯片提供稳定可靠的时钟来源。PLL 的输出频率范围为 62.5MHZ~1500MHZ，如图 4-3 所示，配置输入信号 M、N、OD 用于对 PLL 输出频率进行配置，bypass 则用于旁路 PLL，使得 PLL 输出时钟等于输入的参考时钟。PLL

输出频率与配置参数 M、N、OD 以及参考时钟频率之间的计算关系如下（其中 M 位宽为 7，最低位是 M_1，最高位是 M_7；N 位宽为 4，最低位是 N_0，最高位是 N_3；OD 位宽为 2，最低位是 OD_0，最高位是 OD_1）：

$$Freq_{pll\_clk} = Freq_{ref\_clk} * \left(\frac{M}{N}\right) * \left(\frac{1}{NO}\right);$$

$$M = M_7 * 128 + M_6 * 64 + M_5 * 32 + M_4 * 16 + M_3 * 8 + M_2 * 4 + M_1 * 2;$$

$$N = N_3 * 8 + N_2 * 4 + N_1 * 2 + N_0 * 1;$$

$$NO = 2^{2*OD_1+OD_0};$$

上述计算公式的各参数需要满足如下几个条件：

1) $1MHZ \leq \frac{Freq_{ref\_clk}}{N} \leq 50MHZ;$

2) $500MHZ \leq Freq_{pll\_clk} * NO \leq 1500MHZ;$

3) $M \geq 2; N \geq 2;$

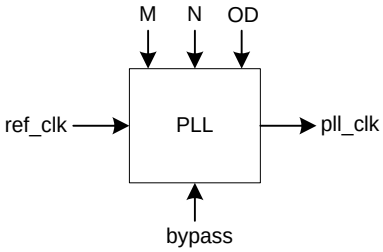


图 4-3 PLL 结构框图

GSC3281 芯片有三个时钟配置引脚可对 PLL 输出频率值进行配置，当默认采用 USB PHY 提供的 12MHZ 参考时钟时，根据不同的配置值，PLL 可以输出 600MHZ、534MHZ、500MHZ、468MHZ、400MHZ、333MHZ、267MHZ、200MHZ 等 8 种不同的频率，这三个时钟配置引脚是 nfcle_clkcfg0、nfale_clkcfg1 与 nfren_clkcfg2_emioen，它们的配置情况与输出频率以及 M/N/OD 之间的对应关系如表 4-3 所示。当采用片外参考时钟时，根据三个时钟配置引脚对应的 M/N/OD 值以及参考时钟频率值，可以计算出此时的 PLL 输出频率。PLL 的三个时钟配置引脚与功能引脚进行复用，它们的配置情况不会影响功能引脚的正常使用。

表 4-3 GSC3281 芯片时钟配置引脚对 PLL 的配置情况

配置引脚 clkcfg[2:0][注 1]	等效 M/N/OD 值	PLL 输出频率/MHZ[注 2]
000	M=200,N=2,OD=1	600
001	M=178,N=2,OD=1	534
010	M=250,N=3,OD=1	500
011	M=234,N=3,OD=1	468
100	M=200,N=3,OD=1	400
101	M=222,N=4,OD=1	333
110	M=178,N=2,OD=2	267
111	M=200,N=3,OD=2	200

[注 1]：clkcfg[2:0]即为{ nfren_clkcfg2_emioen, nfale_clkcfg1, nfcle_clkcfg0}；

[注 2]：输入的参考时钟为 12MHZ；

对于给定的参考时钟频率，除了使用三个时钟配置引脚配置产生 8 种不同的 PLL 输出频率，还可以通过软件编程 PLL 时钟配置寄存器 SYSCTL_PLL_FREQ 进行更为精细的配置，并且软件编程配置 PLL 的方式优先级高于时钟配置引脚的配置。SYSCTL_PLL_FREQ 寄存器中包含了 M、N 与 OD 参数，计算方法同样采用前述公式。当软件配置 SYSCTL_PLL_FREQ 寄存器改

变 PLL 输出频率时，PLL 将会自动进行约 0.7ms（在采用 12MHZ 参考时钟的情况下；若采用其他频率参考时钟，则根据时钟周期类推）的相位锁定过程，PLL 相位锁定之后，GSC3281 芯片将工作在新的 PLL 频率之下。

除了 PLL 时钟配置寄存器可用于配置 PLL 输出频率，软件还可以通过 PLL 控制与状态寄存器 SYSCTL_PLL_CSR 设置旁路 PLL 与查询 PLL 是否处于锁定状态。

4.5.3 时钟分频

如前所述，GSC3281 芯片内部绝大多数时钟都由 PLL 时钟进行直接或者间接分频得到，它们的分频关系如图 4-2 所示。根据模块的不同特点，有些时钟由源时钟进行整数分频得到，有些时钟由源时钟进行偶数分频得到，具体情况请查询每个分频时钟的分频系数寄存器描述。在实际应用中，综合考虑不同模块的需求以及性能、功耗等因素，可以选择配置一个合适的 PLL 频率；根据选定的 PLL 频率，针对不同模块分别选择合适的分频系数以产生期望的目标频率。除了初始化过程，在运行过程中也可以根据需要配置改变时钟分频系数，例如在模块空闲时降低模块时钟频率甚至关闭模块时钟以减小功耗。

对于给定的分频系数值 coeff，GSC3281 芯片整数分频与偶数分频后的频率值采用如下计算公式，其中整数分频不支持一分频的情况。

$$\mathrm{Freq}_{\mathrm{integral\_div}} = \frac{\mathrm{Freq}_{\mathrm{source}}}{\mathrm{coeff}+1};$$

$$\mathrm{Freq}_{\mathrm{even\_div}} = \frac{\mathrm{Freq}_{\mathrm{source}}}{2*(\mathrm{coeff}+1)};$$

GSC3281 芯片提供了对每一个功能模块的精细的使能控制，可以由软件单独关闭或者开启某一个功能模块。为了确保时钟分频电路的正常工作，当需要配置改变时钟分频系数时，必须按照如下的三个步骤进行编程：

1) 使能系统控制模块，即将寄存器位 SYSCTL_MOD_CTL1[4]置 1；
2) 关闭目标模块的使能位，即相应的使能位写 0；
3) 编程修改目标模块的时钟分频系数；
4) 开启目标模块的使能位，即相应的使能位置 1；
5) 关闭系统控制模块，即将寄存器位 SYSCTL_MOD_CTL1[4]写 0；

由于 AHB 总线在开机后始终处于工作状态，没有软件可见的使能控制位，因此以上步骤不适用于 AHB 总线时钟 hclk 的配置，软件可直接配置 hclk 分频系数。

4.6 复位管理

4.6.1 复位来源

GSC3281 芯片一共有六种复位来源，包括上电复位、芯片引脚输入的系统复位、看门狗电路超时复位、芯片级软件复位、模块级软件复位以及 EJTAG 调试复位，图 4-4 给出了 GSC3281 芯片中复位产生电路的总体方案。

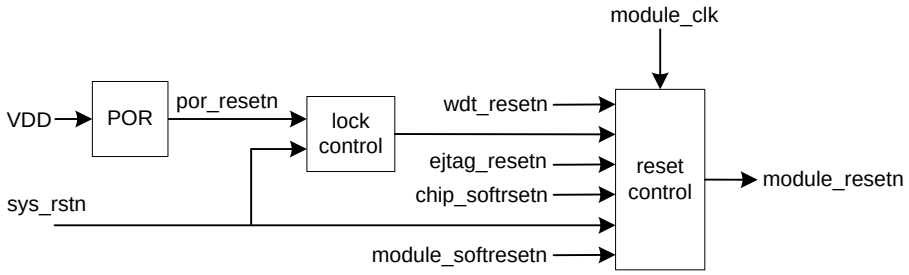


图 4-4 GSC3281 复位方案

为了降低系统成本，GSC3281 芯片在片内集成了上电复位电路。PLL 在上电之后，内部 VCO 需要一定的时间才可以锁定输出，因此整个芯片必须在 PLL 输出稳定之后才可以开始正常工作，在此之前，内部电路必须处于复位状态以避免时钟不稳定引起错误操作。为了确保上电复位之后 PLL 处于锁定状态，GSC3281 芯片在上电复位时采用了图 4-4 所示的锁定控制电路，可以在上电复位之后提供额外的延时复位，确保了复位结束之后 PLL 已经开始正常工作。

在图 4-4 的复位方案图示中，sys_rstn 指的是从芯片引脚输入的系统复位信号，可用于板级手动复位，也可以外接上电复位电路；由于 GSC3281 片内已经集成了上电复位电路，因此在成熟应用板中可以不再使用 sys_rstn 复位信号。wdt_resetn 指的是看门狗电路超时引起的复位信号，如果在某个应用没有使用看门狗电路，则 wdt_resetn 将不会起作用。EJTAG 调试复位 ejtag_resetn 来自于 EJTAG TAP 控制器，调试主机可通过 ejtag_resetn 控制复位整个芯片。芯片级软件复位与模块级软件复位则用于全芯片和某个模块的软件复位。除了模块级软件复位之外，其余五个复位源中的任何一个复位信号都可以复位整个芯片。

4.6.2 软件复位

如图 4-4 复位方案所示，GSC3281 芯片提供了芯片级与模块级两种软件复位，可进一步方便软件与应用的需要。GSC3281 芯片提供了两个软件复位寄存器 SYSCTL_MOD_SRST0 与 SYSCTL_MOD_SRST1，软件通过配置这两个寄存器可以产生期望的软件复位。

当软件配置产生芯片级软件复位时，GSC3281 整个芯片将全部复位，所有软硬件重新从头开始运行。当软件配置某个模块进行软件复位时，则只有该模块被复位，其他模块不会受到影响。

在采用 SPI flash 启动的情况下，当后续操作需要使用到 NAND flash 控制器时，为了确保 NAND flash 控制器能够稳定有效的工作，软件最好在使用之前对 NAND flash 控制器进行一次软件复位。

4.7 功耗管理

GSC3281 芯片提供了一系列的软硬件低功耗措施以降低芯片的功耗，其中软件上可以采用的低功耗措施包括关闭无用模块、降低全芯片或者功能模块的运行频率、休眠模式等；针对具体应用，通过软件上精细采用硬件提供的一系列功耗优化措施，可以显著降低整个芯片的功耗。

4.7.1 模块关闭

在 GSC3281 芯片中，除了 AXI 总线与 AHB 总线保持始终工作的状态，所有其他模块都可以由软件单独使能与关闭；当关闭一个模块之后，除漏电功耗之外，该模块将不再消耗任何能量。针对某个具体的应用，一方面可以关闭无用的模块，另一方面在运行过程中很多模块通常并不会总是处于运行状态，当某些模块处于不需要运行的阶段时，同样可以关闭这些模块，例如一个模块只是偶尔需要收发数据，则在非运行阶段关闭该模块将显著降低功耗。

4.7.2 低频运行

GSC3281 芯片内部绝大多数时钟都由 PLL 输出时钟通过一级或者多级分频得到，几乎所有时钟都可以独立地进行分频控制，同时 PLL 本身也可以由软件进行精细的频率控制，因此针对不同的应用场合，可以对 PLL 时钟以及各模块时钟进行细粒度的控制，使得各时钟运行在满足性能要求的最低频率即可，而不是简单粗放地运行在一个较高的频率之下，这样可以将 GSC3281 芯片的功耗保持在一个较低的水平。

某些功能模块可能需要一直保持在工作状态以维持正常的功能，但往往只有在进行例如数据传输等有效操作时才需要全速运行，其他多数时间将处于空闲等待状态而不需要全速运行，那么在空闲等待状态时可以降低模块的工作频率，使之最低限度维持运行状态即可，则可以最大限度降低功能模块的功耗。

4.7.3 休眠模式

当 GSC3281 的 CPU 执行休眠指令 WAIT 时，CPU 将进入休眠等待状态，CPU 时钟被关闭，不再执行任何指令，只会消耗很少的漏电功耗；在 CPU 进入休眠等待状态时，任何中断都可以将 CPU 唤醒，CPU 退出休眠等待状态并开始进行中断处理；图 4-5 给出了正常模式与休眠模式之间的状态转换图示。

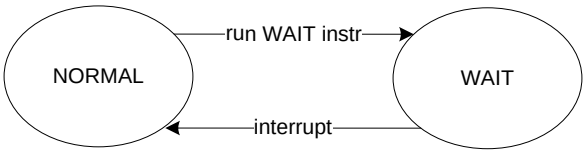


图 4-5 GSC3281 的休眠模式转换图

当执行 WAIT 指令时，在 CPU 进入休眠模式之前，所有在 WAIT 指令之前的指令都会被执行完毕，而所有在 WAIT 指令之后的指令都不会被执行，这些 WAIT 指令之后的指令将在 CPU 退出休眠状态之后继续执行。如果在执行 WAIT 指令时刚好发生中断，则 CPU 将不再进入休眠模式。

CPU 通过执行如下的 WAIT 指令序列进入休眠模式：

```
asm volatile(“.set mips3”);
asm volatile(“wait”);
asm volatile(“.set mips2”);
```

为了使 CPU 在进入休眠模式后能够被合适的中断请求触发退出休眠模式，CPU 在执行 WAIT 指令之前应该将各模块中断功能以及中断控制器配置为一个期望的状态。

GSC3281 芯片的中断控制器可以在没有时钟的状态下接收并发出中断请求，因此 CPU 进入休眠模式之前可以关闭中断控制器的时钟以进一步降低功耗。如果某个功能模块可以在

模块关闭的情况下接受外部事件并发出中断，例如矩阵键盘可以在模块关闭的情况下接收按键信息并发出唤醒中断，则极限情况下，CPU 在进入休眠模式之前可以关闭所有可被关闭的时钟，此时 GSC3281 芯片只有 PLL 时钟、AXI 总线时钟与 AHB 总线时钟等极少的时钟处于工作状态，整个芯片将处于一种极低功耗状态。当发生某个唤醒中断时，例如外部按键触发了矩阵键盘唤醒中断，则 CPU 将退出休眠模式，并根据情况恢复各模块的时钟开始正常运行。

4.8 IO 复用

GSC3281 芯片多数的数字引脚都进行了 IO 复用，即一个芯片引脚在不用的应用场合下可以通过软件配置用作不同的功能，具体复用情况请参考 GSC3281 数据手册中“引脚分布”一节。

当启动模式配置引脚选择某一种启动方式时，相应的 IO 引脚将自动选择为对应的接口功能，例如当选择 NAND Flash 启动时，所有与 NAND Flash 接口引脚复用的 IO 引脚将自动选择为 NAND Flash 接口功能。除启动相关引脚默认为启动功能以及 JTAG 相关引脚默认为 JTAG 功能之外，所有的 IO 复用数字引脚在复位之后均自动默认选择为 GPIO 引脚功能。

当需要将某些 IO 复用引脚选择为某个功能时，可通过 SYSCTL_IOMUX_CFG0 与 SYSCTL_IOMUX_CFG1 两个寄存器进行配置，例如将 SYSCTL_IOMUX_CFG0 寄存器的第 25 位设置为 1，则将 pwmout5_i2sclk、emicsn1_i2ssdi_pwmout1、sim1vccen_i2ssdo、emicsn0_i2sws 四个引脚分别使能为 I2S 的 i2s_clk、i2s_sdi、i2s_sdo、i2s_ws 功能；需要注意的是，一个引脚在任意时刻只能用作一个功能，例如在使能 I2S 功能的情况，就不能再使能 EMI、PWM、SIM 卡 1 的功能，这一点必须由软件编程保证，否则将引起不可知的错误。

GSC3281 支持两种启动方式，当采用某种启动方式时，SYSCTL_IOMUX_CFG0 寄存器与 SYSCTL_IOMUX_CFG1 寄存器中的对应寄存器位在复位后自动复位为 1，相关的复用引脚自动选择为对应的功能，例如当选择 NAND Flash 启动时，SYSCTL_IOMUX_CFG0 寄存器的 nfc_sel 寄存器位复位为 1，NAND Flash 控制器对应的 14 个复用引脚 nfwen_emiwen、nfrnb_emirdy、nfdat0_emid0 等自动选择为 NAND Flash 功能；在完成启动之后，如果启动相关引脚需要选择为其他的功能，则不仅需要在 SYSCTL_IOMUX_CFG0 寄存器与 SYSCTL_IOMUX_CFG1 寄存器中将目标功能的 IO 使能信号配置为 1，还需要同时将启动功能对应的 IO 使能信号配置为 0，即保证一个引脚在任意时刻只用作一个功能，例如系统在 NAND Flash 启动之后使用 EMI 接口，即 nfwen_emiwen、nfrnb_emirdy、nfdat0_emid0 等引脚先用作 NAND Flash 控制器引脚后用作 EMI 接口引脚，则在使用 EMI 接口之前，软件需要将 SYSCTL_IOMUX_CFG0 寄存器的 emi_wrd_sel 寄存器位以及 SYSCTL_IOMUX_CFG1 寄存器的 emi_a38_sel、emi_a02_sel、emi_csn2_sel、emi_csn1_sel、emi_csn0_sel 寄存器位配置为 1，同时将 SYSCTL_IOMUX_CFG0 的 nfc_sel 寄存器位配置为 0。

为了提高引脚复用的效率，某些功能信号可以通过多个引脚进行输入输出，例如 UART3 和 UART4 的所有信号均可通过两组引脚进行输入输出，UART5 的 txe 信号可以通过两个引脚进行输出；在实际使用时，软件可根据需要配置 SYSCTL_IOMUX_CFG0 与 SYSCTL_IOMUX_CFG1 寄存器使能其中某一组（个）引脚即可，其余组别的引脚则可配置用作其他功能。

4.9 总线错误

当 CPU 访问非法地址（即实际不存在的地址）时，GSC3281 芯片内部总线将发起一个总线错误中断，CPU 接收到该中断之后可进行必要的处理。总线错误中断的中断状态保存在系统控制模块的 SYSCTL_BUS_ERR 寄存器中，CPU 可查询或者清除其该寄存器中的总线错误

中断状态。

4.10 模块配置

GSC3281 芯片中每一个功能模块都有相应的配置寄存器，其中绝大多数配置寄存器包含在功能模块内部，但也有部分配置功能由系统控制模块的配置寄存器进行配置，在使用相应的功能模块时，需要对这些配置寄存器也进行适当的配置。这些配置功能主要包括：

- Watchdog 配置：休眠模式下是否使能 watchdog、软件暂停 watchdog 以及 watchdog 计数间隔等；
- NAND Flash 控制器配置：内部 4KB 缓冲区是否由软件控制使能位；
- DDR2 控制器配置：不同主设备请求 AHB INCR 传输的读预取长度；
- UART7 配置：是否将 UART7 配置为红外功能；
- ADC 配置：ADC 使能；
- USB PHY 配置：USB PHY 的配置；
- 时钟极性配置：SCI0/1 的输出时钟极性配置；

4.11 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

4.12 编程指导

当前版本用户手册暂不提供详细编程指导。

5 DDR2 控制器

5.1 概述

GSC3281 DDR2 控制器由控制器核心与 PHY 两大部分构成，兼容 JEDEC 关于 DDR2 SDRAM 的 JESD79-2F 规范。

控制器的主要特征如下：

- 最高工作频率为 533MHZ；
- 最大内存容量为 256MB；
- 支持 16 位位宽的内存颗粒；
- 支持一个 rank 的内存颗粒；
- 支持 Burst 长度为 4 或者 8 的内存颗粒；
- 有 AXI 和 AHB 两个主机端口；
- PHY 时序可软件调节。

5.2 引脚描述

表 5-1 DDR2 控制器引脚描述

名称	类型	上拉 下拉	功能描述
DDR2 控制器接口			
cke	O		DDR2 时钟使能
ck	O		DDR2 时钟信号
ckb	O		DDR2 反向时钟信号
odt	O		DDR2 ODT 信号
csb	O		DDR2 片选信号
rasb	O		DDR2 行激活信号
casb	O		DDR2 列激活信号
web	O		DDR2 写使能信号
dm0	O		DDR2 数据屏蔽第 0 位
dm1	O		DDR2 数据屏蔽第 1 位
addr0	O		DDR2 地址第 0 位
addr1	O		DDR2 地址第 1 位
addr2	O		DDR2 地址第 2 位
addr3	O		DDR2 地址位 3 位
addr4	O		DDR2 地址第 4 位
addr5	O		DDR2 地址第 5 位
addr6	O		DDR2 地址第 6 位
addr7	O		DDR2 地址第 7 位
addr8	O		DDR2 地址第 8 位

名称	类型	上拉 下拉	功能描述
addr9	O		DDR2 地址第 9 位
addr10	O		DDR2 地址第 10 位
addr11	O		DDR2 地址第 11 位
addr12	O		DDR2 地址第 12 位
addr13	O		DDR2 地址第 13 位
ba0	O		DDR2 BANK 地址第 0 位
ba1	O		DDR2 BANK 地址第 1 位
ba2	O		DDR2 BANK 地址第 2 位
dq0	B		DDR2 数据第 0 位
dq1	B		DDR2 数据第 1 位
dq2	B		DDR2 数据第 2 位
dq3	B		DDR2 数据第 3 位
dq4	B		DDR2 数据第 4 位
dq5	B		DDR2 数据第 5 位
dq6	B		DDR2 数据第 6 位
dq7	B		DDR2 数据第 7 位
dq8	B		DDR2 数据第 8 位
dq9	B		DDR2 数据第 9 位
dq10	B		DDR2 数据第 10 位
dq11	B		DDR2 数据第 11 位
dq12	B		DDR2 数据第 12 位
dq13	B		DDR2 数据第 13 位
dq14	B		DDR2 数据第 14 位
dq15	B		DDR2 数据第 15 位
dqs0	B		DDR2 数据选通第 0 位
dqs1	B		DDR2 数据选通第 1 位
dqsb0	B		DDR2 数据选通反相第 0 位
dqsb1	B		DDR2 数据选通反相第 1 位
vcc18	I		DDR2 数字电源(1.8V)
vss	I		DDR2 数字地

5.3 功能说明

DDR2 控制器的结构框图如图 5-1 所示，包括控制器核心和 DDR2 PHY 两部分，它们通过 DFI 接口相连接。其中，控制器核心部分负责内存时序信息控制、初始化序列的控制以及相应内存访问命令的生成；PHY 部分负责与 DDR2 芯片颗粒的交互，为数据的稳定准确采样提供保障。

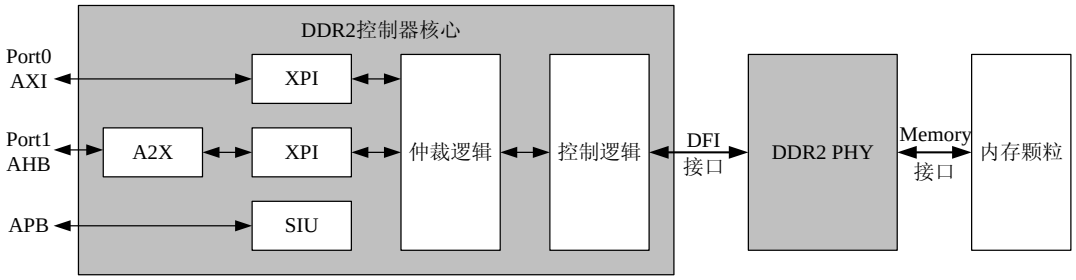


图 5-1 DDR2 控制器结构框图

5.3.1 DDR2 控制器核心

内存控制器核心有两个主机端口，分别是 Port 0 AXI 接口和 Port 1 AHB 接口，数据位宽均为 32 位。其中 AXI 接口与系统中的龙芯处理器相连接，它通过 XPI 模块将 AXI 信号转化为内部数据包；AHB 接口使用 AHB 总线与系统中其它需要使用内存的模块相连接，它包含一个 A2X 模块，用来将 AHB 信号转换成 AXI 信号，然后经过同样的 XPI 模块将 AXI 信号转化为内部数据包。内存控制器核心的编程接口采用 32 位的 APB 总线，连接到软件接口单元 SIU。

5.3.1.1 时钟描述

控制器核心共有四个时钟输入，分别是 ddr2_clk、aclk_ddr2、hclk_ddr2 和 pclk_ddr2。其中 ddr2_clk 是内部工作时钟，控制逻辑和 DFI 信号都同步于该时钟；aclk_ddr2 是 Port0 AXI 的输入时钟，aclk_ddr2 和 ddr2_clk 具有相同的时钟频率，都等于 aclk；hclk_ddr2 是 Port1 AHB 的输入时钟，频率等于 hclk；pclk_ddr2 是 APB 输入时钟，用于 APB 接口配置寄存器，频率等于 pclk。

5.3.1.2 XPI

XPI 是 AXI 端口的接口逻辑，主要实现总线协议的处理、数据缓存和地址映射，映射方法将在 5.3.1.6 节介绍。总线协议的处理包括以下几个方面：读地址的产生、写地址的产生、写数据的产生、读数据的产生和写响应的产生，在实现这些功能时分别会用到读地址队列、写地址队列、读数据队列、写数据队列和写响应队列，它们的深度如表 5-2 所示。

表 5-2 XPI 中的队列深度

AXI Port 0 读地址队列深度	4
AXI Port 0 写地址队列深度	4
AXI Port 0 读数据队列深度	10
AXI Port 0 写数据队列深度	10
AXI Port 0 写响应队列深度	10
AHB Port 1 写地址队列深度	2
AHB Port 1 写数据队列深度	10
AHB Port 1 读地址队列深度	4
AHB Port 1 读数据队列深度	64

5.3.1.3 AHB 接口的数据预取

AHB 接口具备对内存的预取功能，即在读操作的不定长 INCR 传输模式（hburst_1=1）时，会根据输入的 hincr_arlen_1 信号从内存中预取一定长度的数据，从而提高传输效率。输入信号 hincr_arlen_1 用来指示需要预取的 AHB 数据个数，是 log2 编码形式，例如 0 代表预取 1 个 AHB 数据，1 代表预取 2 个 AHB 数据，2 代表预取 4 个 AHB 数据，依次类推。可以预取的最大值不能超过 1K 字节，所以 hincr_arlen_1 最大为 log2(1024/(AHB 接口数据位宽/8))，即最大为 8。如果不定长 INCR 传输的起始地址加上预取长度跨越 1K 边界，根据 AHB 协议，AHB master 不会跨越 1K 边界传输数据，则 AHB 接口只能预取到达 1K 边界时的数据个数。

如果不定长 INCR 传输模式下需要读取的数据长度小于预取长度，传输效率反而会降低，因为这些不需要的数据还是会从内存中预取出来，再被控制器核心自动丢弃，所以 hincr_arlen_1 必须根据需要合理配置。系统中有 6 个 AHB master 连接到 DDR2 控制器的 AHB 接口，分别是 USB、MAC、NAND flash 控制器和 DMA，可以通过系统控制寄存器 SYSCTL_DDR2_CFG 来配置每个 AHB master 读操作时的 hincr_arlen_1 值。

5.3.1.4 仲裁逻辑

在介绍具体的仲裁逻辑之前首先介绍一下控制器核心的 QoS（quality of service）等级。通过寄存器 DDR2_PCFG_n.qos_class（n 为 0 或 1，代表端口号）可以设置端口的 QoS 等级，DDR2 控制器的用户可以为每个端口设置一个固定的 QoS 等级，也可以通过软件配置动态的调整各端口 QoS 等级。

控制器核心支持两级 QoS 设置，分别是 Low latency(LL)和 Best effort(BE)。LL 等级用于需要紧急响应的高优先级传输，例如 CPU cache misses。控制器核心会在每个 QoS 等级运用带宽优化，但是如果频繁在两个优先级之间切换，带宽优化效果反而会减弱，因为此时为了立刻响应高优先级请求，可能损失 DDR 内存的利用率，例如，为了响应 LL 等级的 page miss 命令而忽略 BE 等级的 page hits 命令。

仲裁逻辑的一个主要依据就是 QoS 等级，它分两个步骤对端口传来的命令进行仲裁，依次是端口仲裁和 Bank 仲裁。

- 端口仲裁

端口仲裁负责将两个端口的写通道和读通道请求进行仲裁后送到后面 bank 队列。

首先进行读写仲裁，将每个端口的读写命令通道组合为一个通道，用户可以通过寄存器 DDR2_PCFG_n.dir_grp_cnt 定义每次组合时连续的读、写命令个数。当某个端口的读、写命令同时到达时，使用 simple round-robin 算法在不同的命令类型间进行切换；如果一次只有一种类型的命令（读或写）到达，则直接传递到下一级。

然后，根据各端口的 QoS 等级将所有命令排列到两个列表中，一个是 LL 列表，一个是 BE 列表，优先处理 LL 列表中的请求。如果 BE 列表中的指令超过一定时间没有被仲裁，则会暂时停止对 LL 列表的操作而对 BE 列表进行一轮仲裁，超时长度通过寄存器 DDR2_CCFG.timeout_qos 设置。

最后，使用改进的 Deficit Round Robin 算法对当前列表里的端口命令进行仲裁，送到后面的 bank 队列。该算法对每个端口进行带宽分配，从而合理有效的仲裁每个请求，具体细节不作描述。每个端口通过寄存器 DDR2_PCFG_n.quantum 设定一个额度，代表了该端口在总带宽中占有的份额，总带宽就是每个端口的额度之和。

- Bank 仲裁

Bank 仲裁负责对各个 bank 队列进行仲裁。DDR2 内存有 8 个 bank，每个 bank 的读写各有一个队列，所以共有 16 个 Bank 队列，每个队列的深度为 4，即可以存 4 个命令。从端口仲裁模块传递过来的命令根据其 bank 地址及读写方向进入到相应的 bank 队列中。

Bank 仲裁首先根据每个命令的 QoS 进行选择，优先服务 LL 级的 bank 队列。如果一个 LL 命令进入一个 bank 队列的尾部，则这个 bank 队列都被设为 LL 级别，这说明 LL 命令会在同一 bank 队列的 BE 命令之后被执行。

在同一 QoS 级别中，bank 仲裁按照下面表格中的优先级顺序进行仲裁，其中 1 为最高优先级，数值越大，优先级越低。Page hit 指新的命令与上一个命令属于同一个 page，即行地址相同；Page miss 指新的命令与上一个命令属于不同 page，即行地址不同。

表 5-3 命令的优先级

命令类型		同一 bank	不同 bank
Read	Page hit	1	2
	Page miss	6	5
Write	Page hit	3	4
	Page miss	8	7

5.3.1.5 反压机制

控制器核心的-主机端口包含反压机制，用来限制过量传输，从而防止内部的读数据和写响应队列溢出。通过寄存器 DDR2_PCFG_n.bp_rd_en/ bp_rd_en，可以分别独立使能读、写操作的反压控制逻辑。

对于读命令，反压逻辑会跟踪读数据的存储状态，如果读数据队列没有足够的空间接收新的数据，则控制器核心不会把端口的读命令传递到内部。若没有使能反压控制逻辑，控制器核心会忽略 XPI 的 rready 信号，所以系统的 master 必须保证总是 ready 状态来接收读数据，否则出错。

对于写命令，内部逻辑是在写数据写入写数据队列后才会把下一个写命令传递到内部，同样，反压逻辑会跟踪写响应的存储状态，如果写响应队列没有足够的空间接收新的写响应，则控制器核心不会把端口的写命令传递到内部。若没有使能反压控制逻辑，控制器核心会忽略 XPI 的 bready 信号，所以系统的 master 必须保证总是 ready 状态来接收写响应，否则出错。

为了完全符合 AXI 协议，或者当端口时钟频率（aclk_ddr2 或 hclk_ddr2）低于控制器时钟（ddr2_clk）时，反压逻辑必须被使能。当 master 能够保证总是 ready 状态来接收读数据或者写响应，并且端口时钟频率大于或等于控制器时钟频率时，反压逻辑才可以不使能。对于 AHB 端口，当 master 的读、写传输有 BUSY 状态时，也必须使能读、写反压逻辑。

5.3.1.6 地址映射

目前系统所支持的内存颗粒地址位宽如表 5-4 所示，Rank 为 1 个。

表 5-4 与 JEDEC 兼容的内存颗粒地址位宽

Density	I/O	Bank width	Row width	Col width	总位宽
2Gb	x16	3	14	10	27

控制器核心将端口的系统地址映射到 SDRAM 内存的 rank、bank、row 和 column 地址位

上。通过寄存器 DDR2_DCFG.add_map 可以支持 3 种地址映射方式，由于当前支持的 Rank 数为 1，有效的地址映射有基于 row（addr_map=3’b000）和基于 bank （addr_map=3’b001）两种，具体的映射方式如表 5-5 所示。

表 5-5 系统地址到内存地址的映射

内存地址	系统地址 Addr	
	基于 row	基于 rank
Col	Addr[10:1]	Addr[10:1]
Row	Addr[24:11]	Addr[27:14]
Bank	Addr[27:25]	Addr[13:11]

5.3.1.7 操作状态机

控制器核心的控制逻辑包含了控制器的操作状态机，主要完成内存初始化序列控制和内存操作命令（包括读、写、预充电和刷新等）的生成。

控制器的操作状态机如图 5-2 所示，包括 Init_mem、Config、Access 和 Low_power 四个状态。通过寄存器 DDR2_SCTL，可以控制状态机跳转到任意状态。

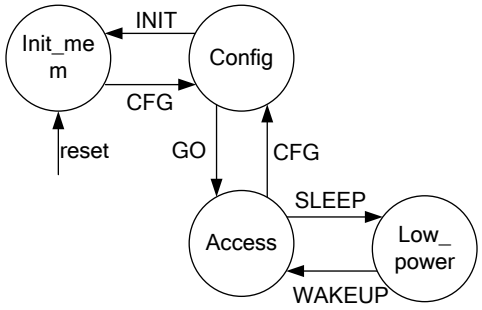


图 5-2 控制器的操作状态机

- Init_mem

系统复位后自动进入 Init_mem 状态，该状态下可以编程所有可写寄存器、完成控制器和内存的初始化。此时内存不进行刷新，所以之前写入的数据有可能丢失。Init_mem 状态同时用于系统软件复位或者需要停止控制器的某些自动控制内存功能时，例如需要停止自动控制内存 Power Down 和刷新功能时。

- Config

该状态用于暂时推迟控制器核心内部的传输操作，如果需要，软件可以重新配置控制器和内存。控制器在这个状态会保持对内存周期性的刷新操作。

- Access

该状态下，控制器核心将接口命令转化成读写内存的命令，并周期性的产生刷新命令，当检测到接口空闲时，控制器核心自动控制内存进入 Power Down 和自刷新模式，直到接口有新的传输操作。在 Access 状态，除了 DDR2_SCFG 和 DDR2_SCTL 寄存器，不能配置其它寄存器。

- Low_power

该状态下，控制器核心不需要产生刷新操作，内存处于自刷新模式。

5.3.1.8 初始化流程

控制器核心在 Init_mem 状态时完成控制器与内存的初始化，其流程是通过一系列寄存器的写操作来实现的，这些寄存器描述详见第**错误！未找到引用源。**节。具体序列如下：

1. 对控制器进行复位，如果需要，配置 DDR2_TOGCNT1U、DDR2_TOGCNT100N 和 DDR2_TINIT 寄存器；
2. 配置 DDR2_MCFG 寄存器以区分内存类型；
3. 等待 DDR2 PHY 初始化结束（寄存器 DDR2_DFISTSTAT0 的最低位为 1）；
4. 配置控制器核心的其它寄存器，设置内存时序寄存器；
5. 根据 DDR2 内存的初始化流程，配置 DDR2_MCMD 寄存器；
   1) 写 Deselect 命令到 DDR2_MCMD 寄存器，等待至少 400ns；
   2) 轮询 DDR2_MCMD.start_cmd = 0；
   3) 写 PREA 命令到 DDR2_MCMD 寄存器，轮询 DDR2_MCMD.start_cmd = 0；
   4) 写 MR2 命令到 DDR2_MCMD 寄存器，轮询 DDR2_MCMD.start_cmd = 0；
   5) 写 MR3 命令到 DDR2_MCMD 寄存器，轮询 DDR2_MCMD.start_cmd = 0；
   6) 写 MR1 命令到 DDR2_MCMD 寄存器，轮询 DDR2_MCMD.start_cmd = 0；
   7) 写 MR 命令到 DDR2_MCMD 寄存器，用于 DLL Reset，轮询 DDR2_MCMD.start_cmd = 0；
   8) 写 PREA 命令到 DDR2_MCMD 寄存器，轮询 DDR2_MCMD.start_cmd = 0；
   9) 写 REF 命令到 DDR2_MCMD 寄存器，轮询 DDR2_MCMD.start_cmd = 0；
   10) 重复 9)至少一次；
   11) 写 MR 命令到 DDR2_MCMD 寄存器，用于初始化器件操作，轮询 DDR2_MCMD.start_cmd = 0；
   12) 步骤 7)后至少 200 个 ddr2_clk 时钟周期，写 MR1 命令到 DDR2_MCMD 寄存器，用于 OCD 校准，轮询 DDR2_MCMD.start_cmd = 0；
6. 配置 DDR2_SCTL 寄存器，使控制器核心跳转到 Config 状态；
7. 根据需要进一步配置控制器核心的寄存器；
8. 配置 DDR2_SCTL 寄存器，使控制器核心跳转到 Access 状态。

5.3.2 DDR2 PHY

GSC3281 芯片的 DDR2 PHY 通过 DFI 接口与控制器核心连接，通过 Memory 接口与内存颗粒连接。它有两个工作时钟，分别是一倍时钟 dfi_clk1x 和两倍时钟 dfi_clk2x（对应的系统时钟是 ddr2_clk 和 ddr2phy_clk），支持最高频率为 533MHZ，即两倍时钟为 533MHZ。

GSC3281 芯片的 DDR2 PHY 具有如下特征：

1. 支持对 DDR2 PHY 内部时钟相位调节
2. 支持对 DQS 信号相位调节

5.3.2.1 上电复位顺序

如图 5-3 所示，DDR2 PHY 上电复位顺序是：

1. 通过 phyreg_rstn 信号复位 PHY 的寄存器；
2. 此时可以配置 DDR2 PHY 寄存器；

3. phyreg_rstn 复位之后至少两个时钟周期，然后通过 sys_rstn 复位 PHY；
4. 等待 2000 个时钟周期，dfi_init_complete 信号变高；
5. DDR2 控制器核心开始初始化和读写数据。

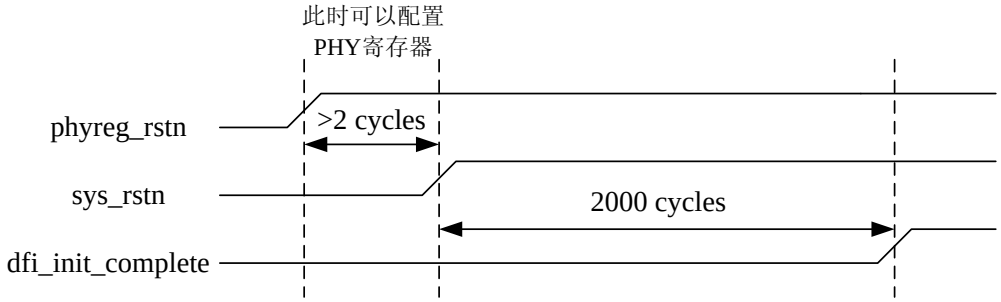


图 5-3 DDR2 PHY 上电复位序列

需要注意的是，每次重新配置 PHY 的寄存器后，都需要通过 sys_rstn 复位一次。具体步骤如下：

1. 根据需要配置 PHY 的寄存器；
2. 将寄存器位 SYSCTL_MOD_CTL1[4]置 1，使能系统控制模块；
3. 将寄存器位 SYSCTL_MOD_SRST1[9]置 1，复位 sys_rstn；
4. 依次将寄存器位 SYSCTL_MOD_SRST1[9]和 SYSCTL_MOD_CTL1[4]写 0，关闭 DDR2 PHY 软复位和系统控制模块；
5. 等待 dfi_init_complete 信号变高（寄存器 DDR2_DFISTSTAT0 的最低位为 1）。

5.3.2.2 时间参数

控制器核心与 PHY 之间的接口符合 DFI2.0 协议，从 DFI 接口到内存颗粒间的控制信号延时由参数 t_{ctrl_delay} 表示，如图 5-4 所示。

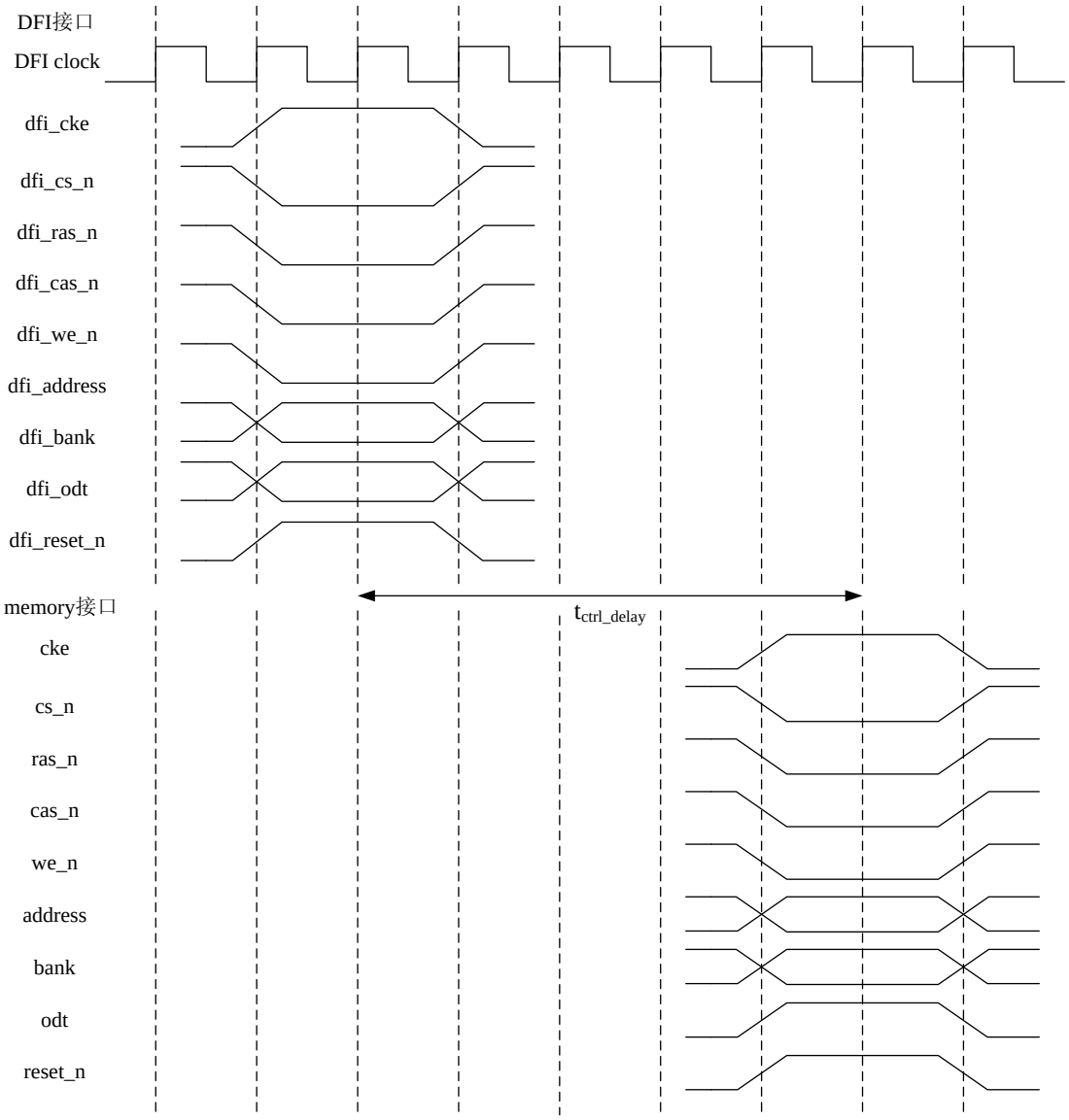


图 5-4 DFI 控制信号时间参数

对于写操作，DFI 协议包括写数据（dfi_wrdata）、写数据掩盖（dfi_wrdata_mask）和写数据使能（dfi_wrdata_en）信号，以及 t_{phy_wrlat} 参数。如图 5-5 所示，dfi_wrdata_en 信号必须在 dfi 写命令后的 t_{phy_wrlat} 个时钟周期后置位；dfi_wrdata 和 dfi_wrdata_mask 在 dfi_wrdata_en 置位后一个时钟周期以后有效。

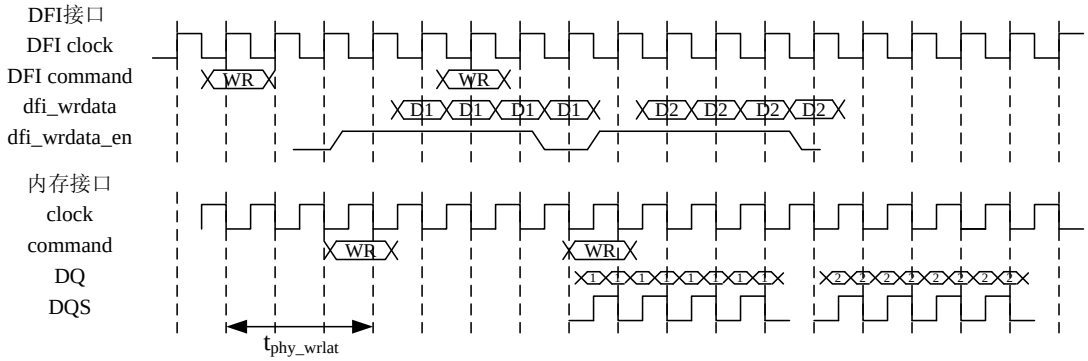


图 5-5 DFI 写操作时间参数

对于读操作，DFI 协议包括读使能（dfi_rddata_en）、读数据（dfi_rddata）和读数据有效（dfi_rddata_valid）信号，以及 t_{rddata_en} 和 t_{phy_rdlat} 参数，其中，t_{rddata_en} 是一个固定延迟值，t_{phy_rdlat} 是定义的一个最大值。如图 5-6 所示，dfi_rddata_en 信号必须在 dfi 读命令后的 t_{rddata_en} 个时钟周期后置位，对于连续的读命令，dfi_rddata_en 在第一个读命令后 t_{rddata_en} 个时钟周期后置位，然后保持整个数据流的长度有效；dfi_rddata 和 dfi_rddata_valid 必须在 dfi_rddata_en 置位后 t_{phy_rdlat} 个时钟周期以内返回。

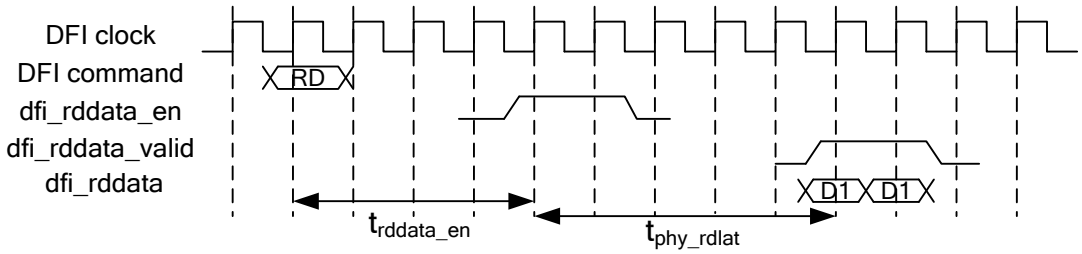


图 5-6 DFI 读操作时间参数

本系统中 DDR2 PHY 的时间参数如表 5-6 所示，其中读延迟 RL=CL+AL；写延迟 WL=RL-1；CL 和 AL 分别表示内存颗粒的 CAS 延时和附加延时，通过控制器核心的 DDR2_TCL 和 DDR2_TAL 寄存器设置。

表 5-6 DDR2 PHY 的时间参数

	t_{ctrl_delay}	t_{phy_wrlat}	t_{rddata_en}	t_{phy_rdlat}
DDR2 内存	4	WL-1	RL-2	11

5.4 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

5.5 编程指导

当前版本用户手册暂不提供详细编程指导。

6 NAND Flash 控制器

6.1 概述

目前的 NOR Flash 存储器价格较高，相对而言 NAND Flash 存储器更经济，并且采用 NAND Flash 也可以进行启动引导，因此现在大多数用户采用 NAND Flash 进行数据存储。

GSC3281 的引导代码可以存储在 NAND Flash 存储器上，为了支持 NAND Flash 启动，GSC3281 的 NAND Flash 控制器中配备了一个内置的 BUFFER。引导启动时，NAND Flash 控制器将 NAND Flash 存储器中第 0 块第 0 页数据加载到控制器内部的 BUFFER 中并且执行；执行的代码会把 NAND Flash 中的剩下引导数据读取到内存中，读取完成后，CPU 将会跳到内存中，继续执行引导代码。

6.2 引脚描述

表 6-1 NAND Flash 控制器引脚描述

名称	类型	上拉 下拉	描述
nfce	O	-	NAND Flash 片选信号，低有效
nfcle	O	-	当此信号为高时，nfdata 此时传输的是命令
nfale	O	-	当此信号为高时，nfdata 此时传输的是地址
nfre	O	-	当此信号为下降沿时，说明正从 NAND Flash 读取数据
nfwe	O	-	当此信号为上升沿时，说明正向 nfdata 写数据，地址或是命令
nfrnb	I	上拉	当此信号为低时，NAND Flash 颗粒正处于 busy 状态，颗粒正在进行擦除操作、写操作或是读操作。
nfdata[0]	B	-	数据线 0
nfdata[1]	B	-	数据线 1
nfdata[2]	B	-	数据线 2
nfdata[3]	B	-	数据线 3
nfdata[4]	B	-	数据线 4
nfdata[5]	B	-	数据线 5
nfdata[6]	B	-	数据线 6
nfdata[7]	B	-	数据线 7

6.3 功能说明

6.3.1 支持协议

支持 ONFI 1.0 协议

6.3.2 功能特点

1. 支持低功耗模式
2. 可配的读写保护块
3. 支持系统启动
4. 页大小软件可配置为 2KB 或 4KB
5. 接口为 8 位 NAND Flash 存储器接口总线
6. 硬件完成 ECC，子页内(512B)可纠正 8bit 的错误
7. 支持 DMA
8. 支持坏块映射
9. 软件可配时序参数

6.3.3 功能描述

NAND Flash 控制器的内部结构如图 6-1 所示：

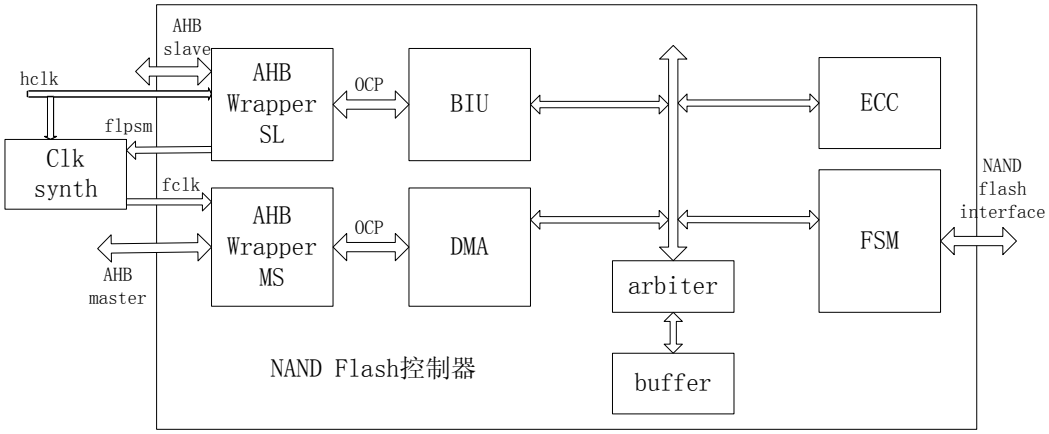


图 6-1 NAND Flash 控制器结构

AHB WRAPPER：此模块将 AHB 数据转换成内部总线数据格式。

BUS Interface Unit：这个模块将内部总线传输变成内部控制器的命令。接口允许 CPU 访问控制器内部所有的寄存器和 BUFFER 空间。内部 BUFFER 大小为 4KB，通过 AHB 接口可对 BUFFER 进行直接读写。该模块工作时钟为 hclk。

FSM Unit：这是控制器中最主要的模块。当有命令写入命令寄存器时，它控制其他模块正确的完成相应的任务。除了最主要的状态机，此模块还包含一组特殊功能寄存器（Special Function Registers）。这些特殊功能寄存器定义了 NAND Flash 颗粒的接口时序参数、传输方式（读还是写）和一些其他传输设置。

ECC：检错纠错模块。纠错是以 NAND Flash 中 512B 大小的子页为单位，最多可以纠正

8bit 的错误。从 NAND Flash 颗粒中读取一整页的数据时，NAND Flash 控制器可以自动的检查错误并纠正相应的错误比特位，如果子页中的错误比特位超过 8 位，控制器将不能进行纠正，但会通过状态寄存器告知。

DMA Unit：此模块加速数据的传输并减轻了 host 的负担。数据可以从 BUFFER 传给 MEMORY，也可以从 MEMORY 传给 BUFFER。

BUFFER Unit：内部 BUFFER 大小为 4KB。对 NAND Flash 颗粒进行整页的读操作时，数据先存放在 BUFFER 里，再由 DMA 通过 AHB 总线传送给 MEMORY 或是其他 HOST。对颗粒进行整页的写操作时，首先由 HOST 将数据写入 BUFFER，然后控制器将 BUFFER 中的数据写入外部的 NAND Flash 颗粒。

6.3.4 内部 BUFFER

为了能使 NAND Flash 控制器通过硬件完成与 NAND Flash 颗粒之间的整页读写操作，并且能够支持 DMA 功能，需要在 NAND Flash 控制器中内置 BUFFER，如图 6-2 所示。BUFFER 由四个宽度为 8bit 的单端口 RAM 组成，分别为 buffer0、buffer1、buffer2 和 buffer3，地址深度均为 1088，字地址偏移量分别为 0，4，8，……4348。CPU 可以通过 AHB 总线直接访问 BUFFER 的 1088 个地址空间。地址偏移 0-4092 这 1024 个字地址空间为 DATA 区域，大小为 4KB，地址偏移 4096-4220 这 32 个字地址空间为 ECC0 区域，大小为 128B，地址偏移 4224-4348 这 32 个字地址空间为 ECC1 区域，大小为 128B。

如果使能 NAND Flash 控制器的 ECC 功能，则硬件自动完成检验和纠错功能，此时内置 BUFFER 的 ECC0 和 ECC1 两块区域会被使用。这里以页为 4KB 的 NAND Flash 颗粒为例。向 NAND Flash 颗粒进行整页的写数据操作时，当写完前 512B 数据，NAND Flash 控制器就会计算出这 512B 数据对应的 ECC 校验码，大小为 13B，存储到 ECC0 区域的前 4 个地址空间，当写完第二组 512B 数据时，生成对应的 ECC 校验码存储到 ECC0 区域接下来的 4 个地址空间，以此类推，当写完最后一组 512B 数据时，对应的 ECC 校验码填满 ECC0 区域的最后 4 个地址空间，这样 NAND Flash 控制器再把整个 ECC0 区域中的数据写入 NAND Flash 颗粒的 OOB 区域。

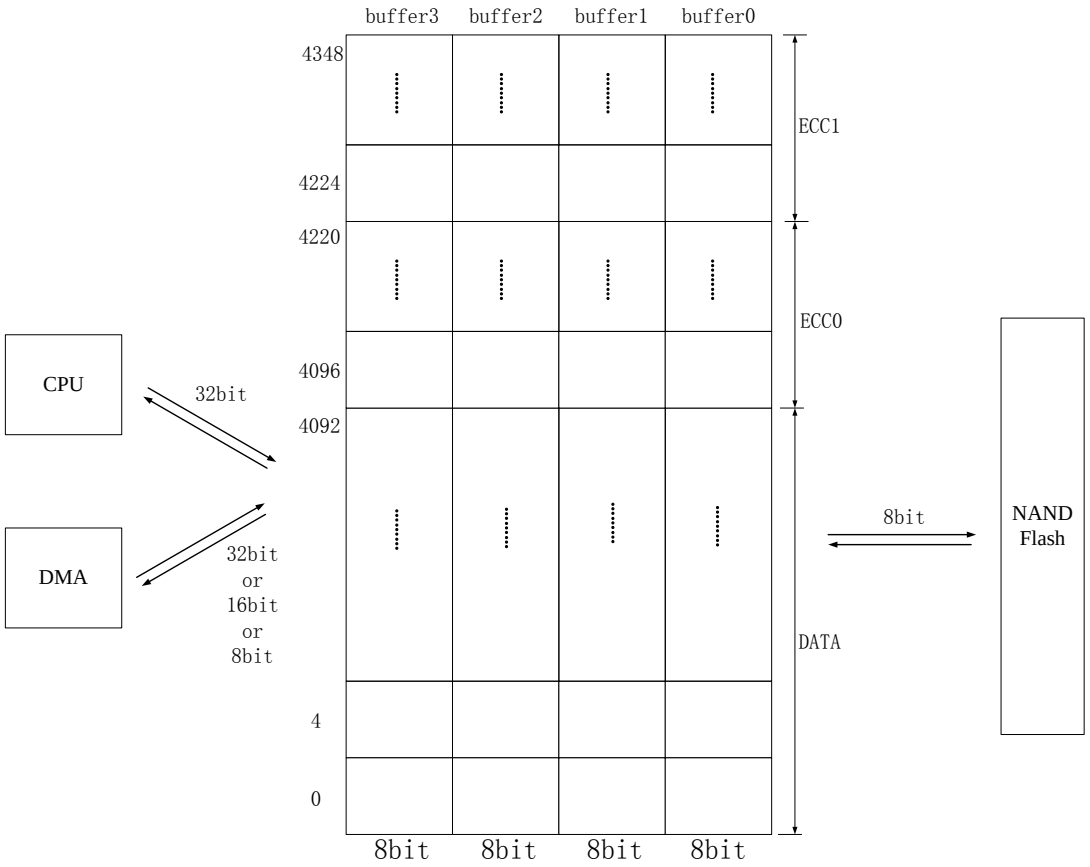


图 6-2 内置 BUFFER

使能 ECC 功能的情况下，在对 NAND Flash 颗粒进行整页的读操作时，整个 BUFFER 都会被使用。首先从 NAND Flash 颗粒读取前 512B 数据，此时生成 512B 数据对应的 ECC 校验码，存储到 ECC1 区域的前 4 个地址空间，然后读完第二组 512B 数据，生成对应的 ECC 校验码存放在 ECC1 区域中接下的 4 个地址空间，以此类推，当读完最后一组 512B 数据时，生成对应的 ECC 校验码填满整个 ECC1 区域。紧接着从 NAND Flash 颗粒读取 OOB 区域中的 ECC 校验码存放到 ECC0 区域。最后将 ECC0 和 ECC1 中的 ECC 码进行运算，可以检验数据中的错误并纠正，当每 512B 数据中的错误比特数超过 8，则不能纠错了。

如果 NAND Flash 颗粒的页大小为 2KB 时，DATA 区域只会被占用前 2KB 大小的空间，与此时同，ECC0 和 ECC1 两个区域也分别只被占用前 64B 大小的空间。

需要注意的是，当 BUFFER 与 NAND Flash 颗粒之间进行数据通信时，数据位宽是 8bit。当 BUFFER 与 CPU 间进行数据通信，数据位宽是 32bit。当 BUFFER 与 DMA 进行数据通信时，数据位宽由寄存器 NFC_DMACTRL 中的 DMA_SIZE 来设定。

6.3.5 BUFFER 与寄存器的地址映射关系

CPU 除了能直接访问 NAND Flash 控制器的内部寄存器外，还能直接访问控制器的内部 BUFFER。6.3.4 节已经对内部 BUFFER 做了详尽介绍，DATA 空间为 4KB，ECC0 和 ECC1 的空间大小均为 128B。如图 6-3 所示，BUFFER 的起始物理地址即是 NAND Flash 控制器的物理基地址 0X1C044000，寄存器的起始物理地址为 0X1C045300。

需要注意的是，BUFFER 与寄存器间有一段空出来的不可用地址。

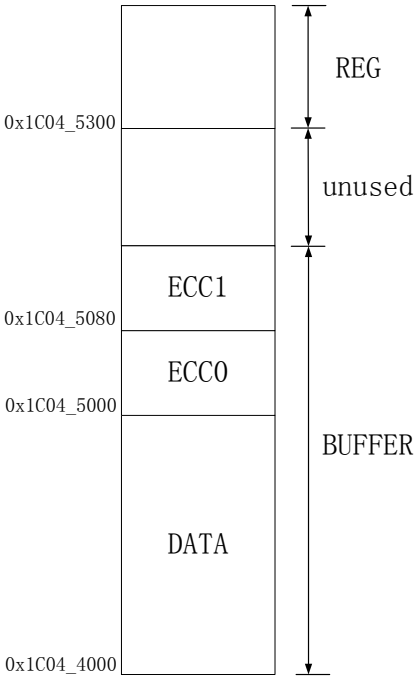


图 6-3 BUFFER 与寄存器的地址映射关系

6.3.6 DMA 操作

NAND Flash 控制器通过内置 DMA 来加快数据的传输，并减轻总线的压力。DMA 能使控制器的内部 BUFFER 与 GSC3281 中的任何 MEMORY 进行数据交互。通过 DMA 读取 BUFFER 的数据时，是从 BUFFER 的低地址顺序递增，通过 DMA 向 BUFFER 写数据时，也是从 BUFFER 的低地址顺序递增。支持的 BURST 方向软件可配，具体见寄存器描述部分。

NAND Flash 控制器与 NAND Flash 颗粒进行整页大小的数据通信时，DMA 能够加快数据传输，下面以整页写操作和整页读操作为例进行具体说明。

进行整页写操作时，首先进行 DMA 相关配置 ，使其从外部 MEMORY 搬运整页的数据到 BUFFER，然后控制器进行整页写的操作，将整页数据写入 NAND Flash 颗粒，如图 6-4 所示。关于整页写操作具体可参见 6.3.7.3 节。

如果只向 NAND Flash 颗粒写入一页数据，可以首先配置 NFC_DMAADDR、NFC_DMACTRL 和 NFC_CNTR 三个寄存器，设置好 DMA 搬运的数据长度，源地址和 BURST 方式，并将 DMA_DIR 置 0，最后将 start_flag 置 1 触发 DMA 搬运开始，通过查询 DMA_BUSY 位来确定 DMA 完成没有，完成之后向 NFC_COMM 寄存器中配置整页写命令，这样一整页数据将写入 NAND Flash 颗粒。

如果要向 NAND Flash 颗粒写入大量数据，则需要打开 DMA 的自动触发模式。首先配置 NFC_DMAADDR、NFC_DMACTRL 和 NFC_CNTR 三个寄存器，设置好 DMA 搬运的数据长度，源地址和 BURST 方式，并将 DMA_DIR 置 0，然后将 NFC_CTRL 寄存器的 DMA_trigger 位和 Trans_IE 位置 1，使 DMA 进入自动触发模式，并打开自动传输完成的中断。最后向 NFC_COMM 寄存器中配置整页写命令，这样 DMA 将开始进行搬运，搬运结束后，BUFFER 中数据将自动写入 NAND Flash 颗粒，写完之后 CPU 将接收到控制器的中断，CPU 重新配置 DMA 的相关寄存器，然后再次向 NFC_COMM 寄存器中配置整页写命令，控制器又将自动完成 DMA 的搬运并将整页数据写入 NAND Flash 颗粒，依此类推，每次向 NAND Flash 颗粒写入一页数据，

CPU 只需要干预一次。

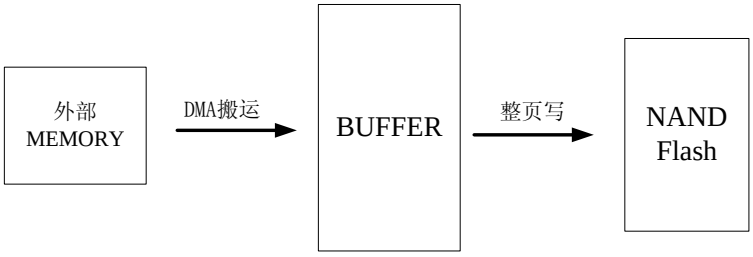


图 6-4 DMA 整页写操作

进行整页读操作时，首先控制器进行整页读操作，将 NAND Flash 颗粒中的整页数据读取到内部 BUFFER 中，然后对 DMA 进行相关配置，使其将 BUFFER 中的数据搬运内外部 MEMORY，如图 6-5 所示。关于整页读操作具体可参见 6.3.7.1 节。

如果只从 NAND Flash 颗粒读入一页数据，可以首先配置 NFC_DMAADDR、NFC_DMACTRL 和 NFC_CNTR 三个寄存器，设置好 DMA 搬运的数据长度，源地址和 BURST 方式，并将 DMA_DIR 置 1，然后向 NFC_COMM 寄存器中配置整页读命令，这样一整页数据将从 NAND Flash 颗粒读入 BUFFER 中，最后将 start_flag 置 1 触发 DMA 搬运开始，通过查询 DMA_BUSY 位来确定 DMA 完成没有。

如果要从 NAND Flash 颗粒读入大量数据，则需要打开 DMA 的自动触发模式。首先配置 NFC_DMAADDR、NFC_DMACTRL 和 NFC_CNTR 三个寄存器，设置好 DMA 搬运的数据长度，源地址和 BURST 方式，并将 DMA_DIR 置 1，然后将 NFC_CTRL 寄存器的 DMA_trigger 位和 Trans_IE 位置 1，使 DMA 进入自动触发模式，并打开自动传输完成的中断。最后向 NFC_COMM 寄存器中配置整页读命令，这样控制器将从 NAND Flash 颗粒读取一整页数据到 BUFFER，完成之后 DMA 将自动从 BUFFER 中进行搬运，搬运完成后，CPU 将接收到自动传输完成的中断，然后 CPU 重新配置 DMA 的相关寄存器，然后再次向 NFC_COMM 寄存器中配置整页读命令，控制器又将从 NAND Flash 颗粒读取一整页数据，然后 DMA 完成搬运，依此类推，每次从 NAND Flash 颗粒读入一页数据，CPU 只需要干预一次。

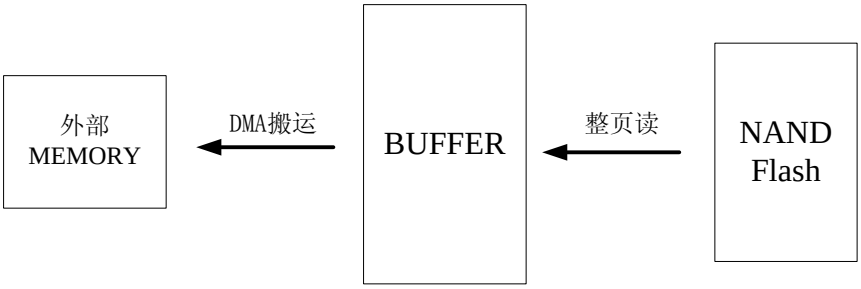


图 6-5 DMA 整页读操作

6.3.7 基本读写操作

NAND Flash 控制器的操作命令如表 6-2 所示，表中的命令值是写入 NFC_COMM 寄存器的值，控制器会将命令值解析成 ONFI 协议中通用的命令，即表 6-2 中的基本命令字节和附加命令字节，在本节中只对基本的读写操作命令进行了详细的解析说明，关于其他命令请参见 ONFI 协议，协议中对操作命令有详细说明。

表 6-2 NAND Flash 控制器的命令

序号	命令名	命令值	基本命令字节	附加命令字节	说明

1	Page Read	0x0030	0x00	0x30	从 NAND Flash 读取一整页的数据，并存放到 BUFFER 中
2	Page Read 1	0x0130	0x00	0x30	发送命令和地址到 NAND Flash
3	Read for Internal Data Move	0x0035	0x00	0x35h	发送命令和地址（4 或 5 字节）到 NAND Flash
4	Random Data Read	0x0005	0x05	0xE0h	发送命令和地址(2 字节)到 NAND Flash
5	Read Status	0x0070	0x70	-	发送命令到 NAND Flash
6	Program Page	0x0080	0x80	0x10h	将 BUFFER 中的数据都写入 NAND Flash
7	Program Page 1	0x0180	0x80	-	发送命令和地址到 NAND Flash
8	Program Page Cache	0x0580	0x80	0x15	将 BUFFER 中的数据都写入 NAND Flash
9	Write Page	0x0010	0x10h	-	发送命令到 NAND Flash
10	Write Cache	0x0015	0x15		发送命令到 NAND Flash
11	Program for Internal Data Move	0x0085	0x85	0x10	发送命令 0x85，然后接着发送地址（4 或 5 字节），最后发送命令 0x10
12	Random Data Input for Program	0x0185	0x85	-	发送命令和 2 字节的地址到 NAND Flash
13	Random Data Input for Program 1	0x0585	0x85	-	发送命令和 5 字节的地址到 NAND Flash
14	Block Erase	0x0060	0x60	0xD0	发送命令和地址到 NAND Flash
15	Reset	0x00FF	0xFF	-	发送命令到 NAND Flash
16	Read ID	0x0090	0x90	-	发送命令和地址到 NAND Flash

除支持上述通用命令外，本控制器还支持主流 NAND Flash 的特殊命令，如表 6-3 所示，此处不做详细说明，具体请参考 NAND Flash 颗粒的芯片手册。

表 6-3 NAND Flash 控制器支持的特殊命令

序号	命令名	命令值	基本命令字节	附加命令字节
Micron memories				
1	Page Read Cache Mode Start	0x1031	0x31	-
2	Page Read Cache Mode Start Last	0x103F	0x3F	-
3	Page Read Cache Mode Start 1	0x1131	0x31	-
4	Page Read Cache	0x113F	0x3F	-

	Mode Start Last 1			
5	Program Page 2	0x1280	0x80	0x11
6	OTP Program	0x10A0	0xA0	0x10
7	OTP Program 1	0x11A0	0xA0h	-
8	OTP Protect	0x10A5	0xA5	0x10
9	OTP Read	0x10AF	0xAF	0x30
10	OTP Read 1	0x11AF	0xAF	0x30
11	Plane Page Read	0x1230	0x00	0x30
12	Plane Page Read 1	0x1330	0x00	0x30
13	Plane Random Data Read	0x1206	0x06	0xE0
14	Plane Random Data Read 1	0x1306	0x06	0xE0
15	Plane Page Program	0x1081	0x81	0x10
16	Plane Page Program 1	0x1181	0x81	-
17	Plane Page Read for Internal Data Move	0x1235	0x00	0x35
18	Plane Page Program for Internal Data Move	0x1285	0x85	0x11
19	Plane Page Erase	0x1160	0x60	0xD0
20	Plane Page Read Status	0x1078	0x78	-
21	Write Plane Page	0x1011	0x11	-
STMicroelectronics memories				
1	Cache Read	0x2031	0x00	0x31
2	Cache Read 1	0x2131	0x00	0x31
3	Cache Read 2	0x2231	-	-
4	Exit Cache Read	0x2034	0x34	
5	Read Block Lock Status	0x207A	0x7A	-
6	Blocks Unlock	0x2023	0x23	0x24
7	Blocks Lock	0x202A	0x2A	-
8	Blocks Lock-Down	0x202C	0x2C	-
Samsung memories				
1	Program Page 2	0x0280	0x80	0x11
2	Program Page Cache	0x0580	0x80	0x15
3	Block Erase 1	0x0160	0x60	0xD0
4	Plane Page Program	0x0081	0x81	0x10
5	Plane Page Program 1	0x0181	0x81	-

6	Read EDC Status	0x007B	0x7B	-
7	Read Chip 1 Status	0x00F1	0xF1	-
8	Read Chip 2 Status	0x00F2	0xF2	
9	Write Plane Page	0x0011	0x11	-

6.3.7.1 整页读操作（Page Read）

控制器对 NAND Flash 颗粒进行整页读的操作是一种效率较高的读操作。首先对 NFC_CONF 寄存器进行配置，设置合适的时序参数。通过配置 NFC_ADDR0L 寄存器和 NFC_ADDR0H 寄存器来设置读操作对应的起始地址，由于是整页读操作，所以地址应是整页的起始处。最后在 NFC_COMM 寄存器写入整页读操作的命令值 0x0030，可参见表 6-2。控制器会将命令值 0x0030 解析为 ONFI 协议规定的命令，即 0x00 和 0x30。

如图 6-6 所示，控制器首先发送命令 0x00 给 NAND Flash 颗粒，紧接着发送地址（4 字节或是 5 字节，视 NAND Flash 颗粒容量大小而定），然后发送命令 0x30，最后控制器自动从 NAND Flash 颗粒读取一整页的数据存储到内部 BUFFER 中，如果 ECC 功能被使能，还会进行 ECC 校验和纠错。当 nfrnb 信号置高后，整页读操作结束。CPU 或是 DMA 可以直接从内部 BUFFER 读取这一整页数据。

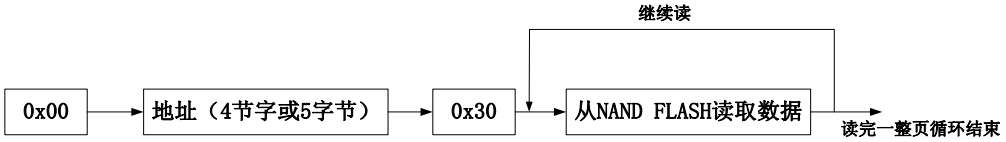


图 6-6 整页读操作

6.3.7.2 普通读操作（Page Read 1）

控制器对 NAND Flash 颗粒也可以进行普通读操作。对 NFC_CONF 寄存器进行配置，设置合适的时序参数。然后通过配置 NFC_ADDR0L 寄存器和 NFC_ADDR0H 寄存器来设置读操作对应的起始地址，最后在 NFC_COMM 寄存器写入普通读操作的命令值 0x0130，可参见表 6-2。控制器会将命令值 0x0130 解析为 ONFI 协议规定的命令，即 0x00 和 0x30。

如图 6-7 所示，控制器首先发送命令 0x00 给 NAND Flash 颗粒，紧接着发送地址（4 字节或是 5 字节，视 NAND Flash 颗粒容量大小而定），然后发送命令 0x30，最后 CPU 通过读取 NFC_DATA 寄存器可以获得颗粒中的数据。

需要注意的是，CPU 可以通过连续读取 NFC_DATA 寄存器，从而可以获得 NAND Flash 颗粒中连续地址上的数据。CPU 可以通过这种方式读取页内任意个数的连续数据。

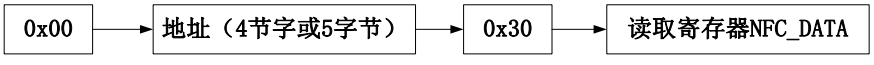


图 6-7 普通读操作

6.3.7.3 整页写操作（Program Page）

控制器对 NAND Flash 颗粒进行整页写的操作是一种效率较高的写操作。首先 CPU 或是 DMA 把将要发送给 NAND Flash 颗粒的整页数据写入控制器的内部 BUFFER。然后对 NFC_CONF

寄存器进行配置，设置合适的时序参数。通过配置 NFC_ADDR1L 寄存器和 NFC_ADDR1H 寄存器来设置写操作对应的起始地址，由于是整页写操作，所以地址应是整页的起始处。最后在 NFC_COMM 寄存器写入整页读操作的命令值 0x0080，可参见表 6-2。控制器会将命令值 0x0080 解析为 ONFI 协议规定的命令，即 0x80 和 0x10。

如图 6-8 所示，控制器首先发送命令 0x80 给 NAND Flash 颗粒，紧接着发送地址（4 字节或是 5 字节，视 NAND Flash 颗粒容量大小而定），然后从内部 BUFFER 读取数据，连续的向 NAND Flash 颗粒中写入数据，当写完一整页数据时，最后发送命令 0x10 来结束写操作。

图 6-8 整页写操作

6.3.7.4 普通写操作（Program Page 1）

控制器对 NAND Flash 颗粒也可以进行普通写操作。对 NFC_CONF 寄存器进行配置，设置合适的时序参数。然后通过配置 NFC_ADDR1L 寄存器和 NFC_ADDR1H 寄存器来设置写操作对应的起始地址，最后在 NFC_COMM 寄存器写入普通读操作的命令值 0x0180，可参见表 6-2。控制器会将命令值 0x0180 解析为 ONFI 协议规定的命令，即 0x80。

如图 6-9 所示，控制器首先发送命令 0x80 给 NAND Flash 颗粒，紧接着发送地址（4 字节或是 5 字节，视 NAND Flash 颗粒容量大小而定），然后 CPU 向 NFC_DATA 寄存器写入数据，此时 NFC_DATA 寄存器的数将会被控制器写入 NAND Flash 颗粒。

需要注意的是，CPU 可以向 NFC_DATA 寄存器连续写入数据，这些数据将会被控制器写入 NAND Flash 颗粒的连续地址中去。

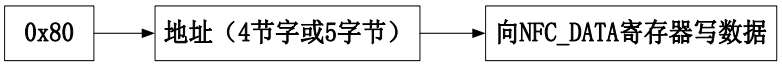


图 6-9 普通写操作

6.3.7.5 块擦除操作（Block Erase）

控制器可以擦除 NAND Flash 颗粒的一整块数据。首先通过配置 NFC_ADDR1L 寄存器和 NFC_ADDR1H 寄存器来设置块擦除操作对应的起始地址，然后在 NFC_COMM 寄存器写入块擦除操作的命令值 0x0060，可参见表 6-2。控制器会将命令值 0x0060 解析为 ONFI 协议规定的命令，即 0x60 和 0xD0。

如图 6-10 所示，控制器首先发送命令 0x60 给 NAND Flash 颗粒，紧接着发送需要擦除的块地址（3 字节），然后最后发送命令 0xD0，NAND Flash 颗粒开始擦除目标块，当 nfrnb 信号置高后，块擦除操作结束。

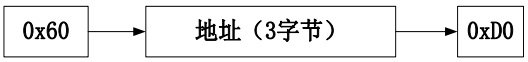


图 6-10 块擦除操作

6.3.8 NAND Flash 启动

GSC3281 支持 NAND Flash 启动。如果通过芯片外部引脚配置将 GSC3281 设置成 NAND Flash 启动，GSC3281 在上电或是系统复位之后，NADN Flash 控制器会自动将 NADN Flash 存储器颗粒第 0 块第 0 页的数据加载到控制器内部的 BUFFER 中，CPU 将直接在 BUFFER 中执行启动代码。

需要注意的是，在自动导入期间，ECC 功能没有使能，不进行 ECC 校验和纠错，所以 NAND Flash 颗粒的第 0 块第 0 页绝不能有错误。

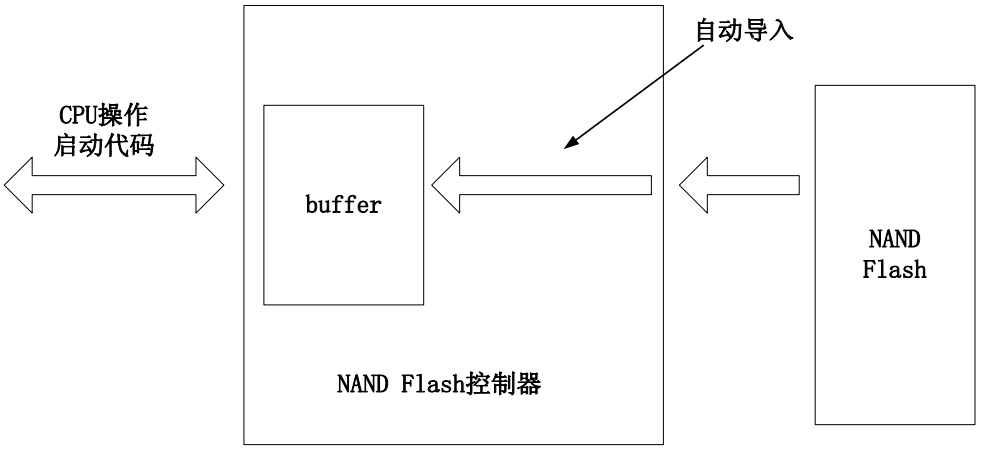


图 6-11 NAND Flash 启动

6.3.9 NAND Flash 接口时序

NAND Flash 接口连接 NAND Flash 控制器与 NAND Flash 颗粒，使两者能够正常通信。除了 nfce 和 nfrnb，其余信号均与 NAND Flash 的工作时钟 fclk 同步。

NAND Flash 接口的命令时序、地址时序、写数据时序和读数据时序分别如图 6-12、图 6-13、图 6-14 和图 6-15 所示，图中的时序参数可由 NFC_CONF 寄存器配置得到。

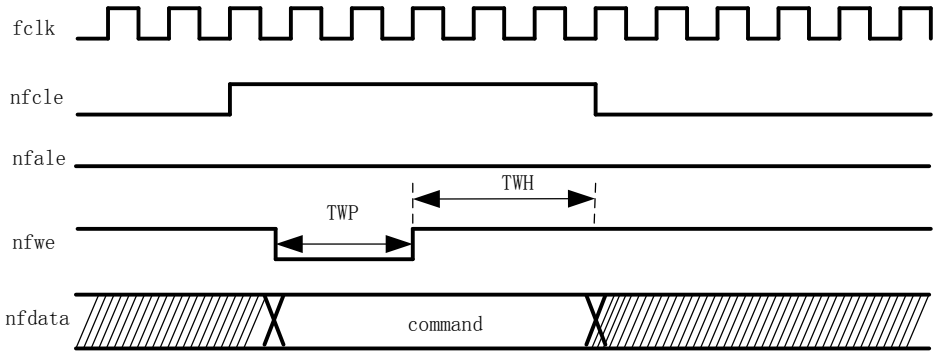


图 6-12 命令时序

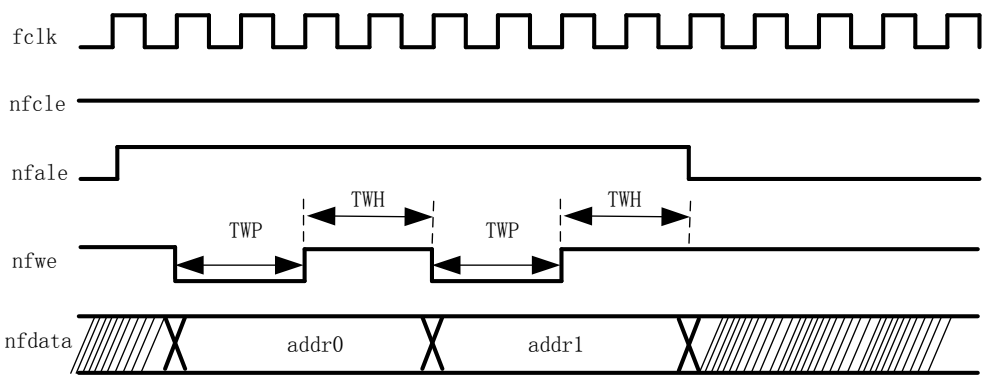


图 6-13 地址时序

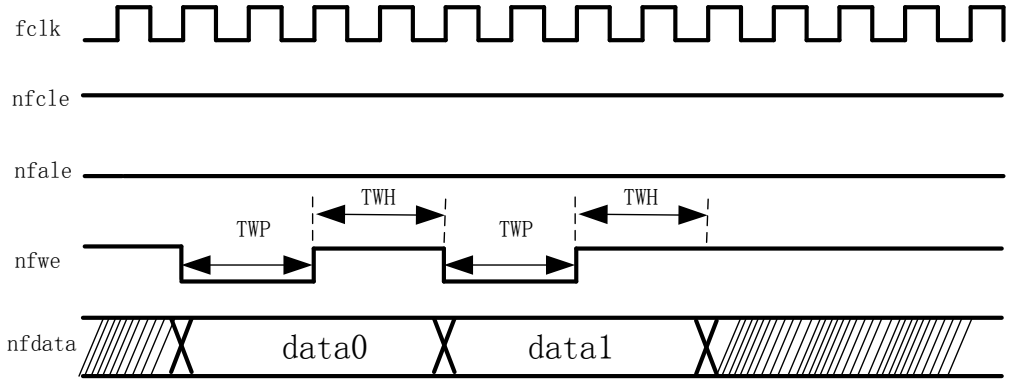


图 6-14 写数据时序

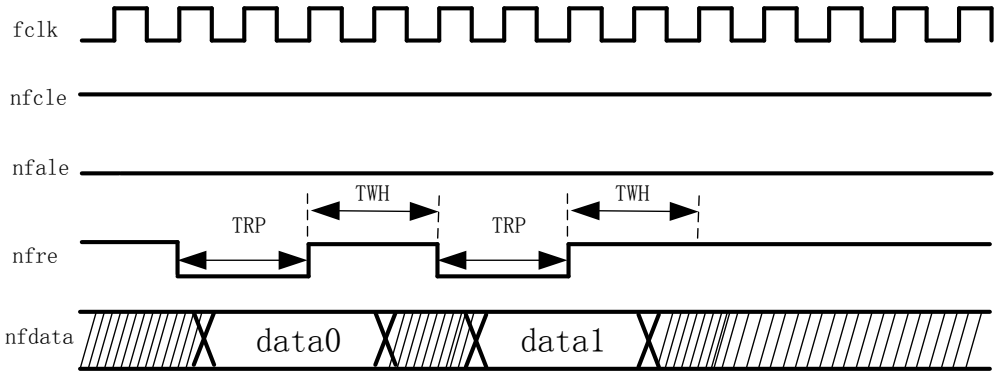


图 6-15 读数据时序

6.4 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

6.5 编程指导

当前版本用户手册暂不提供详细编程指导。

7 USB2.0 OTG 控制器

7.1 概述

GSC3281 USB OTG 控制器完全兼容以下规范：

- On-The-Go Supplement to the USB 2.0 Specification, Revision1.3a 和 Revision2.0；
- USB 2.0 Specification；

控制器的主要特征如下：

- 支持 USB 主机模式；
- 支持 USB 设备模式；
- 支持高速、全速、低速模式；
- 支持控制（Control）、大块（Bulk）、中断（Interrupt）、同步（Isochronous）传输模式；
- 支持 Buffer DMA 和 Scatter-gather DMA 模式；
- 支持 8 个主机通道（Host Channel）；
- 支持 5 个设备端点（EndPoint）；
- 支持 root hub；
- 支持挂起和远程唤醒功能（Suspend&Resume）；

7.2 引脚描述

表 7-1 USB2.0 OTG 控制器引脚描述

名称	类型	上拉 下拉	功能描述
vbus	B		USB 5V 电源信号
usb_id	I	U	USB ID 信号
usb_dp	B		USB D+信号
usb_dm	B		USB D-信号
txrtune/rkelvin	B		USB2.0 PHY 的高速阻抗匹配调整
usb_xo	I		晶体的 XO 信号
usb_xi	I		晶体的 XI 信号
avdd33	I		USB 模拟电源 3.3V
avss33	I		USB 模拟地，对应 3.3V 电源
dvdd12	I		USB 数字电源 1.2V
dvss12	I		USB 数字地，对应 1.2V 电源
vss33c	I		USB 模拟地，对应 3.3V 电源
utmi_drvvbus	O		电荷泵使能

7.3 功能说明

USB OTG 控制器组成部分如图 7-1 所示，主要包括 USB OTG core、OTG PHY 以及用于数据缓存的 SPRAM（FIFO）。

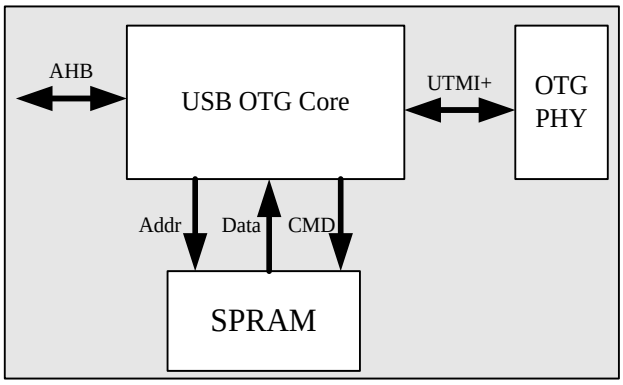


图 7-1 USB OTG 控制器结构图

USB OTG 控制器可以支持主机模式和设备模式，但是，在同一时刻下只能工作于一种模式。

USB OTG 控制器内部集成 DMA 控制器，USB OTG 控制器发送包和接收包的数据访存操作都是由 DMA 控制器来完成，SPRAM 是用来暂时存放发送和接收数据的 FIFO。

7.3.1 DMA 模式简介

USB OTG 控制器的 DMA 操作分为两类：Scatter Gather 模式和 Buffer DMA 模式。

Scatter Gather 模式下，USB OTG 控制器通过在内存中构建一个描述符表来实现 Scatter Gather 这种零散的内存访问请求。该描述符表如图 7-2 所示。

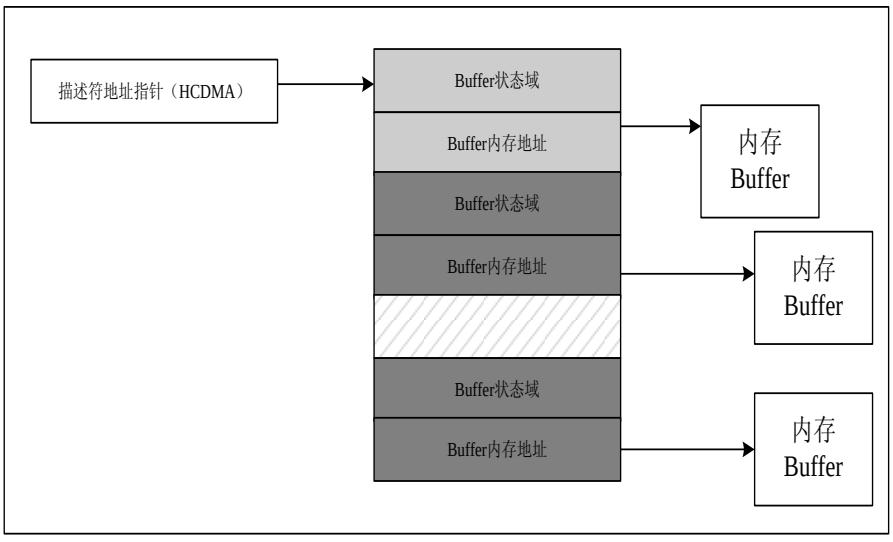


图 7-2 描述符表

描述符表的每一项都由 buffer 状态域和当前 buffer 所对应的内存物理地址两项组成。描述符表的首地址就是 USB_HCDMA 寄存器的值，USB OTG 控制器通过 USB_HCDMA 寄存器的值寻找到描述符中第一项 buffer 的内存地址值，根据该描述符中的 buffer 状态域中的 EOL 域来决定该描述符是否为最后一项，如果不是最后一项，完成该项的内存访问操作以后判断

当前 Scatter Gather DMA 操作是否完成；如果没有完成，对当前 buffer 的描述符地址（如果是第一项，那就是 USB_HCDMA 的值）进行加 4 操作，得到下一项 buffer 的描述符地址；依此类推，直到最后一项 buffer 处理完毕。

Buffer 状态域的格式分为输入/输出模式，输入/输出模式又分为同步和非同步传输两种类型。

Buffer 状态域定义如图 7-3 所示[注 1]：

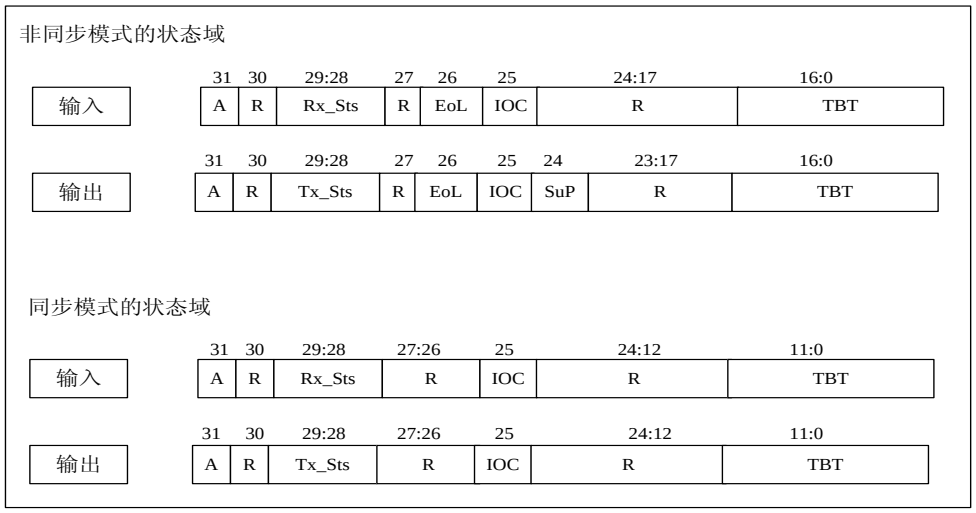


图 7-3 Scatter Gather 模式下 Buffer 状态域定义

描述符表中 Buffer 状态域定义如表 7-2 所示。

表 7-2 buffer 状态域定义

位域	位	功能描述
A	[31]	描述符是否准备好 0x0: 描述符未准备好 0x1: 描述符已经准备好
Rx_Sts/Tx_Sts	[29:28]	接收/发送数据状态 0x0: 成功 0x1: 包错误 0x2: 保留 0x3: 保留
EOL	[26]	描述符最后一项 0x0: 非描述符最后一项 0x1: 描述符最后一项
IOC	[25]	设置该域以后，每当该描述符处理结束以后，USB OTG控制器就会产生一个传输完成中断。
SuP	[24]	非同步模式的输出模式下，表示对应的Buffer数据为setup包的8字节数据
TBT	[16:0] / [11:0]	待传输的数据字节数 对于非同步模式，最大传输字节数为128K-1字节；对于同步模式，最大传输字节数为4K-1字节。

[注 1]：描述符表 buffer 状态域定义中的 R 字符是 Reserved 简写。

相对 Scatter DMA 模式而言，Buffer DMA 模式比较简单，USB OTG 控制器通过读取 USB_HCDMA 寄存器来获取存放数据的内存地址进行 DMA 访问即可。

7.3.2 地址映射

USB OTG 控制器寄存器有四大类，分别为：全局控制配置寄存器、主机模式控制配置寄存器、设备模式控制配置寄存器以及时钟门控配置寄存器。这些寄存器的地址分配如图 7-4 所示。

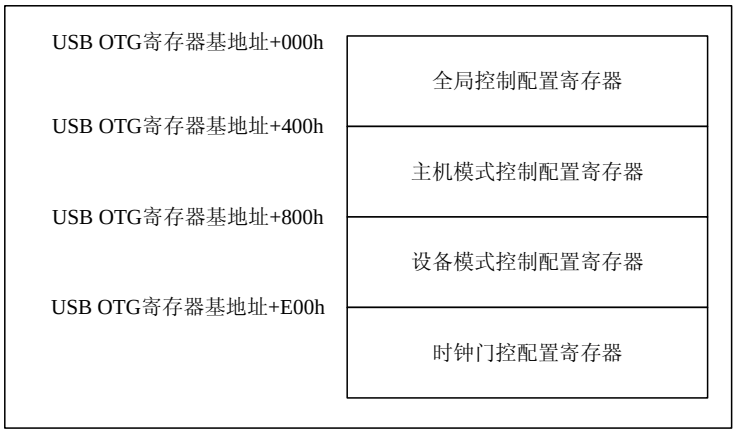


图 7-4 寄存器地址映射

USB OTG 控制器的 FIFO 组织比较复杂，主机和设备模式下都是用这一 FIFO 存储接收和发送数据包。

主机模式下，FIFO 的地址映射如图 7-5 所示。

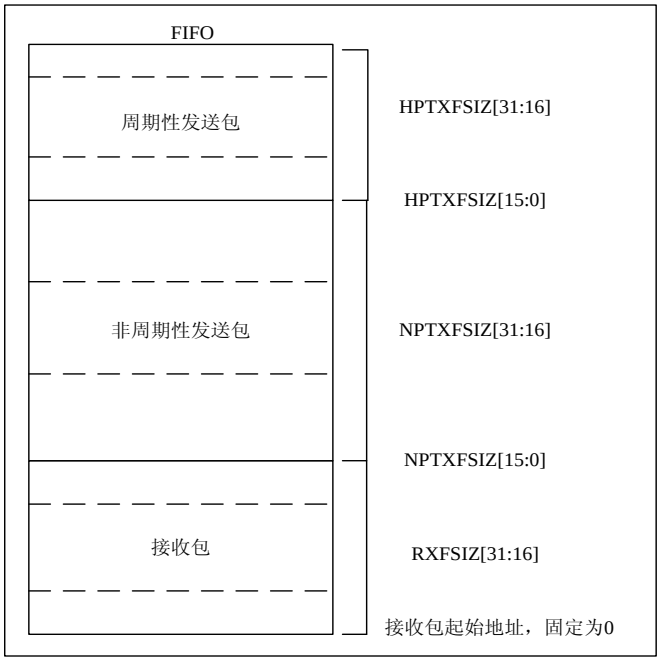


图 7-5 主机模式下 FIFO 地址映射

主机模式下，对于所有的非周期 OUT 交易，使用一个非周期性传输 FIFO；对于所有周期性 OUT 交易，使用一个周期性传输 FIFO。这些传输 FIFO 用作传输缓冲来存储将要发送到设备方的数据。对于多个 OUT 交易，在 USB OTG 控制器内部有一个请求队列来管理。USB OTG 控制器对于所有周期性和非周期性 IN 交易，使用一个接收 FIFO，该 FIFO 用作接收缓冲来存储接收到的数据包。另外，对于接收包，每个包的状态也存放在该 FIFO 中。

设备模式下，FIFO 的地址映射如图 7-6 所示。

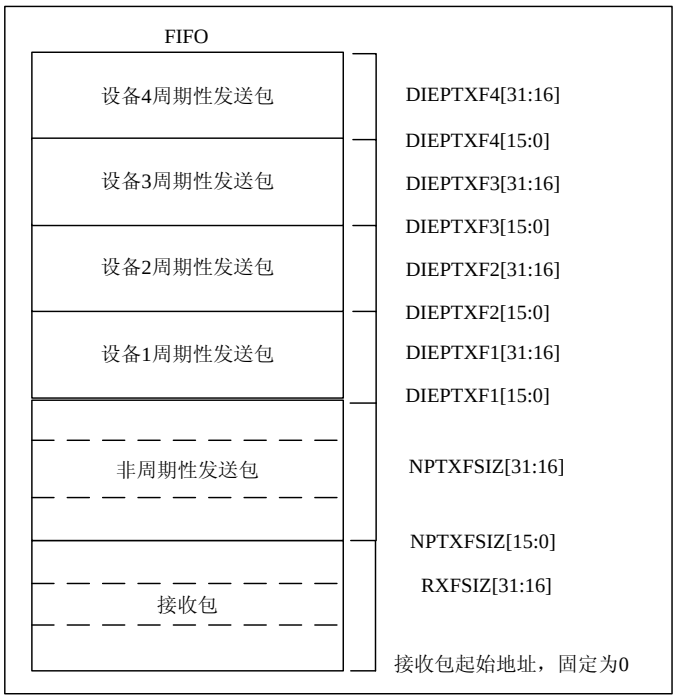


图 7-6 设备模式下 FIFO 地址映射

设备模式下，所有非周期性的 IN 交易，共用一个传输 FIFO；对于周期性的 IN 交易，每个端点单独使用一个传输 FIFO。对于所有 OUT 交易，USB OTG 控制器使用一个接收 FIFO 来接收主机发送的数据，这个接收 FIFO 也存储与接收包对应的状态信息，这些状态信息包括：字节数、数据 PID 和接收数据是否有效标识等等。

7.3.3 主机模式

GSC3281 USB OTG 控制器工作在主机模式时，共支持 8 个主机通道，支持 Hub。当工作在非 Hub 模式下时，DMA 模式支持 Buffer DMA 和 Scatter-Gather DMA；当工作在 Hub 模式下时，只支持 Buffer DMA 模式。

当 USB OTG 控制器工作在主机模式时，如果使用 Scatter Gather DMA 模式，控制器支持非字对齐的内存访问模式。在 Scatter Gather DMA 模式下，USB 传输协议中的 NAK/NYET 的处理是由 USB OTG 控制器直接处理，无需软件干预。

主机模式下，USB OTG 控制器支持用户将主机配置为 High Speed 模式或者 Full Speed 模式。速度模式的设置在 USB_HCFG 寄存器的 FSLSSupp 域实现。USB OTG 控制器负责检测设备的连接或者断开，处理 USB 复位和设备枚举过程，USB 的挂起（SUSPEND）和激活（RESUME），并且负责检测来自设备的远程唤醒。

主机可以发起挂起（Suspend）操作，此时 USB PHY 的时钟输出停止，USB 控制器处于节电模式。当 USB OTG 控制器处于节电模式时，只有当软件重新配置寄存器或者设备发出唤醒（Resume）功能时，USB 控制器才会退出挂起（Suspend）操作。

当 USB OTG 控制器工作在主机模式下，如果从主机配置寄存器获取一个 token 请求，则立即开始启动一个 USB 交易。此时，USB OTG 控制器建立并且发送该请求的 token 包。对于 OUT 或者 SETUP 交易，USB OTG 控制器从发送 FIFO 中读取数据，建立并且发送该数据包，并且等待设备传回对于该包的响应。接收到设备方的响应之后，USB OTG 控制器向上层软件汇报该交易的状态。

对于 IN 或者 PING 交易，USB OTG 控制器发送 IN 或者 PING token 之后，等待相应的数

据或者握手包。如果接收到相应的握手包，USB OTG 控制器将会向软件发送该响应包的状态；如果接收到的是数据包并且该包具有正确的 PID，USB OTG 控制器将会把这些数据写到相应的接收 FIFO 中并且检查数据的完整性，然后向设备方发送握手响应包。

7.3.4 设备模式

GSC3281 USB OTG 控制器工作在设备模式时，共支持 5 个设备端点，这 5 个端点都支持数据双向传输，其中端点 0 支持控制模式传输。

当 USB OTG 控制器工作在设备模式时，如果使用 Scatter Gather DMA 模式，控制器不支持非字对齐的内存访问模式，USB OTG 针对每个设备端点维护一套 DMA 描述符表，这些描述符表各自独立。

作为设备时，USB OTG 控制器默认为高速设备。软件可以通过对 USB_DCFG 寄存器 DevSpd 域的设置来将该设备设置为全速/低速设备。USB OTG 控制器负责处理 USB 的复位序列和设备枚举过程并且以此来决定 USB 该工作的速度模式。另外，USB OTG 控制器检测来自主机的 USB 的挂起（SUSPEND）和激活（RESUEM）操作。设备模式下，如果主机进入挂起模式，USB OTG 控制器可以对主机发起远程唤醒操作。

设备模式下，当接收到任何从主机发出的 token 包时，USB OTG 控制器负责解码并且检查 token 包的完整性。

如果接收到的 token 包是一个有效的 OUT 或者 SETUP 包，USB OTG 控制器将会等待并且检查接下来传输的数据包的 PID 域，然后将相应的数据写到接收 FIFO 中。当包接收完成时，USB OTG 控制器检查数据的完整性，如果主机有响应需求时，USB OTG 控制器要向主机发送相应的应答包，此时，控制器还需要将接收包的状态信息写到控制器内部相应的接收状态队列中。

如果接收到一个 OUT token，而此时接收 FIFO 没有准备好， USB OTG 控制器需要向主机发送一个 NAK 握手信号，以示主机继续发送该包。如果接收到的 token 包是一个有效的 PING 包，USB OTG 控制器需要基于 FIFO 状态和系统控制配置寄存器信息发送相应的响应包给主机。

如果接收到一个 IN token，而此时发送数据也在 FIFO 中准备好，USB OTG 控制器将会读取数据，建立相应的数据包，并且将其发送到主机。如果接收到 IN token，但是此时发送数据在 FIFO 中没有准备好，USB OTG 控制器将会发送 NAK 握手包给主机，以示主机进行后续操作。

7.4 主机/设备工作流程

7.4.1 USB OTG 上电初始化

无论是设备模式还是主机模式，USB OTG 的驱动软件必须对 USB OTG 控制器完成系统初始化工作。通过系统初始化，决定 USB 工作于何种工作模式（主机模式/设备模式）并且解除系统全局中断屏蔽。

具体的 USB OTG 上电初始化流程如下：

1、 设置 USB_GAHBCFG 寄存器，将 GlbIntrMsk 位设置为 1，解除全局中断屏蔽；

2、 设置 USB_GINTMSK 寄存器，将 RXFlv1 位屏蔽位设置为 0，此时，如果接收到数据包，则产生系统中断；

3、 设置 USB_GINTMSK 寄存器，解除 OTG 中断屏蔽以及模式不匹配中断屏蔽；
4、 软件读取 USB_GINTSTS 寄存器的 CurMod 位，决定当前 USB OTG 控制器工作于主机还是设备模式。

7.4.2 主机模式工作流程

在主机工作模式下，从用户的角度来说，与设备进行数据传输之前，首先要完成主机初始化，然后通过对通道进行初始化来完成相应数据传输的设置。

主机初始化流程如下：

（1） 配置 USB_GINTMSK 寄存器的 PrtIntMsk 位来解除对端口中断的屏蔽；
（2） 配置 USB_HCFG 寄存器来选择 Full-speed 或 High-speed 模式；
（3） 将 USB_HPRT 寄存器的 PrtPwr 位置为 1，来驱动 USB 接口上的 Vbus 信号；
（4） 等待 USB_HPRT 寄存器的 PrtConnDet 中断，该中断标识设备已经连接到端口上；
（5） 将 USB_HPRT 寄存器的 PrtRst 位置为 1，开始 USB 协议规定的 Reset 过程；
（6） 等待 Reset 过程的结束，需要等待至少 10ms；
（7） 将 USB_HPRT 寄存器的 PrtRst 位置为 0；
（8） 等待 USB_HPRT 寄存器的 PrtEnChng 中断；
（9） 读取 USB_HPRT 寄存器的 PrtSpd 域以获取枚举出来的设备速度；
（10） 配置 USB_HFIR 寄存器，设置相应的 SOF 间隔时间；
（11） 配置 USB_GRXFSIZ 寄存器来选择接收 FIFO 的大小；
（12） 配置 USB_GNPTXFSIZ 寄存器来选择非周期传输 FIFO 的大小和起始地址；
（13） 配置 USB_HPTXFSIZ 寄存器来选择周期传输 FIFO 的大小和起始地址。

通道初始化操作流程如下：

（1） 配置 USB_GINTMSK 寄存器的 HChIntMsk 位来解除对通道中断的屏蔽；
（2） 配置 USB_HAINTMSK 寄存器的 PrtIntMsk 位来解除对相应通道中断的屏蔽；
（3） 配置 USB_HCINTMSK 寄存器来取消对主机通道中断寄存器给出的交易相关中断的屏蔽。对于非 Scatter/Gather DMA 模式，执行（4）；对于 Scatter/Gather DMA 模式，执行（6）；
（4） 配置 USB_HCTSIZn 寄存器，设置整体传输数据大小以及包的个数；
（5） 对于 Hub 模式，需要在 USB_HCSPLTn 寄存器中设置 Hub 和端口地址；
（6） 配置选定通道的 USB_HCDMAn 寄存器的 buffer 起始地址；
（7） 配置选定通道的 USB_HCCHARn 寄存器，设定相应设备的端点特性（例如设备传输类型、速度以及方向等）；
（8） 当应用程序已经准备好发送或者接收数据包时，将 USB_HCCHARn 的 Enable 位设置为 1，USB OTG 控制器就开始自动进行包的发送或者接收工作。

7.4.3 设备模式工作流程

系统上电以后，必须执行设备初始化操作，设备才能接收主机发送的 token 并且做出相应的响应。下面为包括设备初始化操作在内的设备模式工作流程。

（1） 配置 USB_DCFG 寄存器的以下域：
a) DescDMA 位，使能 Scatter-Gather DMA 模式；
b) 设备速度；
c) 非零长度状态输出阶段握手协议；

d) 周期帧间隔。

（2） 配置 USB_GINTMSK 寄存器的以下域，来解除相应的中断屏蔽：

a) USB Reset；

b) Enumeration Done；

c) Early Suspend；

d) USB Suspend。

（3） 等待 USB_GINTSTS.USBReset 中断，该中断标识主机发送的 USB reset 操作已经被检测到至少 10ms；

（4） 对于所有输出端点，设置 USB_DOEPCTLn 的 SNAK 位为 1；

（5） 解除下列中断位的屏蔽：

a) USB_DAINTMSK.INEP0；

b) USB_DAINTMSK.OUTEP0；

c) USB_DOEPMSK.SETUP；

d) USB_DOEPMSK.XferCompl；

e) USB_DIEPMSK.XferCompl；

f) USB_DIEPMSK.TimeOut。

（6） 对于数据传输（输入/输出）端点，设置 USB_GINTMSK 寄存器的 NPTxFEmpMsk 和 RxFLvlMsk 位来解除中断屏蔽；

（7） 设置数据 FIFO RAM。配置 USB_GRXFSIZ 寄存器，使得能够接收控制 OUT 和 IN 数据和 Setup 数据。USB_GRXFSIZ 寄存器的最小值等于控制端点的最大包尺寸+64 位（控制 OUT 数据包的状态位）+320 位（setup 包）；

（8） 接收到枚举结束中断（USB_GINTSTS.EnumDone）时，读取 USB_DSTS 寄存器来决定枚举的速度；

（9） 配置 USB_DIEPCTL0 寄存器的 MPS 域来设定最大包尺寸；

（10） DMA 模式下，配置 USB_DOEPCTL0 寄存器来使能控制 OUT 端点 0，用来接收 SETUP 包。对于 Scatter/Gather DMA 模式，描述符必须在控制 OUT 端点 0 使能之前设置好；

（11） 解除 SOF 中断的屏蔽。到这一步，设备可以接收 SOF 包并且可以执行针对控制端点 0 的控制传输；

（12） 当设备接收到 SetAddress 命令时，需要配置 USB_DCFG 寄存器，设置相应的设备地址；

（13） 设置 USB OTG 控制器来发送 Status IN 包；

（14） 当在 SETUP 包中接收到 SetConfiguration 或者 SetInterface 命令时，需要设置 USB_DIEPCTLn/USB_DOEPCTLn 寄存器的 USBActEP 位来激活该端点（将 USB_DIEPCTLn/USB_DOEPCTLn 寄存器的 USBActEP 位设置为 1）；

（15） 配置中曾经激活的端点，在 SetConfiguration 或者 SetInterface 命令产生新设置以后，需要将其释放（将 USB_DIEPCTLn/USB_DOEPCTLn 寄存器的 USBActEP 位设置为 0）；

（16） 解除对新激活的端点的中断屏蔽，同时屏蔽已经释放端点的中断；

（17） 设置 DATA FIFO RAM 对应的接收 FIFO 空间，接收 FIFO、周期性和非周期性发送 FIFO 空间最大值皆为 256 字节，而且要保证所有端点周期性发送 FIFO 加起来最大空间为 256 字节。此时，USB OTG 控制器在设备模式下，可以接收或者发送数据。

7.5 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

7.6 编程指导

当前版本用户手册暂不提供详细编程指导。

8 以太网 MAC 控制器

本章节主要说明本芯片内集成的以太网 MAC 控制器的功能、应用时钟、结构、工作原理，并详细说明了以太网 MAC 控制器的寄存器。

本章节中提到的 DMA 都是指本 MAC 控制器中的 DMA 模块。

8.1 概述

本芯片集成了 10/100Mbps MAC（Media Access Control）控制器，兼容 IEEE 802.3-2008 协议标准，支持全双工和半双工操作，支持标准的 RMII 接口。

MAC 控制器的数据收发通过 AHB 总线与 DDR2 关联，MAC 控制器的寄存器通过 APB 总线由 CPU 来操作，MAC 通过标准的 MDIO 接口配置和控制片外的 PHY 芯片（如 TI 的 DP83848C）。AHB 总线为 Little-endian。

MAC 控制器支持 DMA 接收和发送，内部在接收和发送方向各有一个 2048 字节的 FIFO 作为缓存。由于 FIFO 深度所限，MAC 控制器不支持硬件自动流控机制。

8.2 引脚描述

表 8-1 以太网 MAC 控制器引脚描述

名称	类型	上拉 下拉	功能描述
rmclk	I [注 1]	-	RMII 模式下数据收发时钟 50MHz
rmtxen	O	-	MAC 发送数据使能信号
rmtxd0	O	-	MAC 发送数据第 0 位
rmtxd1	O	-	MAC 发送数据第 1 位
rmrxdv	I	-	RMII 模式载波状态和接收数据有效
rmrxd0	I	-	MAC 接收数据第 0 位
rmrxd1	I	-	MAC 接收数据第 1 位
mdc	O	-	MAC 控制字时钟
mdio	B	-	MAC 控制字串行数据

[注 1]：I 表示输入，O 表示输出。

8.3 功能说明

以太网 MAC 控制器模块支持以下功能：

- 支持 IEEE 802.3 协议
- 支持标准 RMII 接口
- 支持 10/100Mbps 操作
- 支持全双工和半双工操作模式

- 支持接收和发送 DMA
- 接收和发送 FIFO 各 2K 字节
- 自动丢弃错误帧
- 支持对一个特殊 MAC 地址的检测
- Hash 表支持对单播和多播地址的匹配
- 支持混杂模式，即可接收 LAN 中所有帧
- 通过 MDIO 支持对 PHY 的管理
- 支持 VLAN 帧的识别
- 支持 IP 报文头中 checksum 字段的检验
- 支持 TCP/IP 报文中 checksum 字段的插入

8.3.1 以太网 MAC 帧格式

本节主要介绍 MAC 控制器支持的以太网 MAC 帧格式、传输次序和错包类型。

图 8-1 说明了以太网 MAC 帧各个字段的大小和内容以及传输次序。该格式中每个字段的字节次序是先传输的字节在左，后传输的字节在右。在每个字节中的位次序正好相反，低位在左，高位在右。FCS 将作为一个特殊的 32 位字段（最高位在左），而不是 4 个单独的字节。

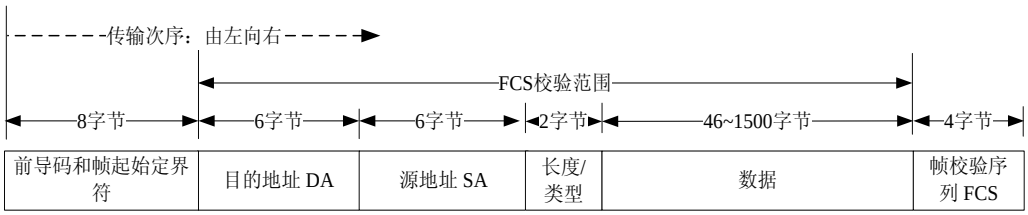


图 8-1 以太网 MAC 帧格式

- 前导码(Preamble)和帧起始定界符(SFD)

前导码包含 8 个字节。前 7 个字节(56 位)的值为 0x55，而最后一个字节为帧起始定界符，其值为 0xD5。前导码为一个由 62 个 1 和 0 间隔（10101010---）的串行比特流，最后 2 位是连续的 1，表示数据链路层帧的开始。其作用就是提醒接收系统有帧到来，以及使到来的帧与输入的随路时钟进行同步。

- 地址字段

每个 MAC 帧包含两个地址字段：目的地址(Destination Address)和源地址(Source Address)。目的地址标识了帧的目的地站点，源地址标识了发送帧的站点。DA 可以是单播地址（单个目的地）或组播地址（组目的地），SA 通常是单播地址（即第 1 位是 0）。

- 长度/类型(Length/Type)

长度/类型字段具有两种意义中的一种。如果这个字段的值小于 1518，那么这个字段就是长度字段，并定义后面的数据字段的长度。但是如果这个字段的值大于 1518，它就标识了在以太网上运行的客户端协议，例如 0x0800 表示数据为 IP 报文。

- 数据(Data)

数据字段包含 46～1500 字节。数据域封装了通过以太网传输的高层协议信息。由于 CSMA/CD 算法的限制，以太网帧必须不能小于某个最小长度。高层协议要保证这个域至少包含 46 个字节。数据域长度的上限为 1500 字节。

- 帧校验序列(FCS)

帧校验序列包含 4 个字节。FCS 是从 DA 开始到数据域结束这部分的校验和。校验和的

算法是 32 位的循环冗余校验法(CRC)。

当对端传输的报文到达 MAC 时，MAC 按照以上描述进行报文检测。当检测到的报文满足以下任意一个条件时，认为接收到的是错包，根据配置或丢弃或继续接收，最后将错误信息写入相应的 Descriptor 位（见下文所述）。

（1）帧长度和 Length/Type 字段中指定的长度不一致。但如果 Length/Type 中包含的是类型值，则不认为是无效帧。

（2）实际接收到的报文长度不是整数字节。

（3）对接收到的帧进行 CRC 校验，发现 CRC 错误。

8.3.2 应用时钟方案

MAC 控制器模块支持标准的 RMII 接口，RMII 接口使用 50MHz 的时钟，数据线为 4 位，接收发送分别占 2 位。

在 RMII 工作模式下，MAC 控制器在数据收发时使用的时钟来自于片外的 50MHz 时钟源，或者来自于 PHY 芯片；但由于 PHY 芯片输出的时钟可能与数据不同步，所以建议 PHY 芯片与 MAC 控制器使用相同的时钟源，如图 8-2 所示。

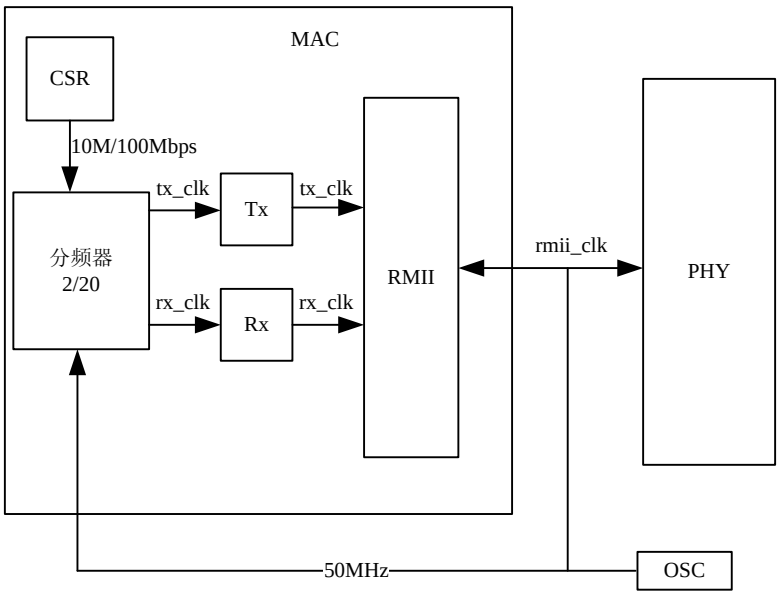


图 8-2 RMII 模式下时钟参考方案

如图 8-2 所示，在 MAC 控制器中有一个分频器，用来对来自晶振的 50MHz 时钟进行分频。根据带宽配置，分频器产生 25MHz 或 2.5MHz 的两个时钟，用以接收和发送逻辑。在 RMII 接口逻辑中，PHY 侧使用的是与 MAC 芯片相同的时钟，即来自于晶振的 50MHz 时钟，而此时钟在接受和发送方向都作为数据的同步时钟来使用。可以看到这样做的好处是，保证了 PHY 与 MAC 在 RMII 接口处时钟的统一性。（注：此方案只是作为参考建议方案，不是唯一可行的方案。）

8.3.3 工作模式切换

为了进行双工模式和带宽的切换，在 MAC 开启时，如果当前的数据传输结束，软件先将 MAC 收发功能关闭，然后可以通过配置 MAC_CONFIG 寄存器进行双工模式和带宽的切换。需要注意的是，在收发功能开启的状态下不能对双工模式或速率进行改变，否则会造成 MAC

接收或发送大量错包。

8.3.4 模块结构与工作原理

如图 8-3 示，以太网 MAC 控制器由以下模块组成：AHB Master 接口，APB 接口，DMA 模块，发送/接收传输控制模块，MAC 模块（含 MAC 寄存器），DMA 寄存器。

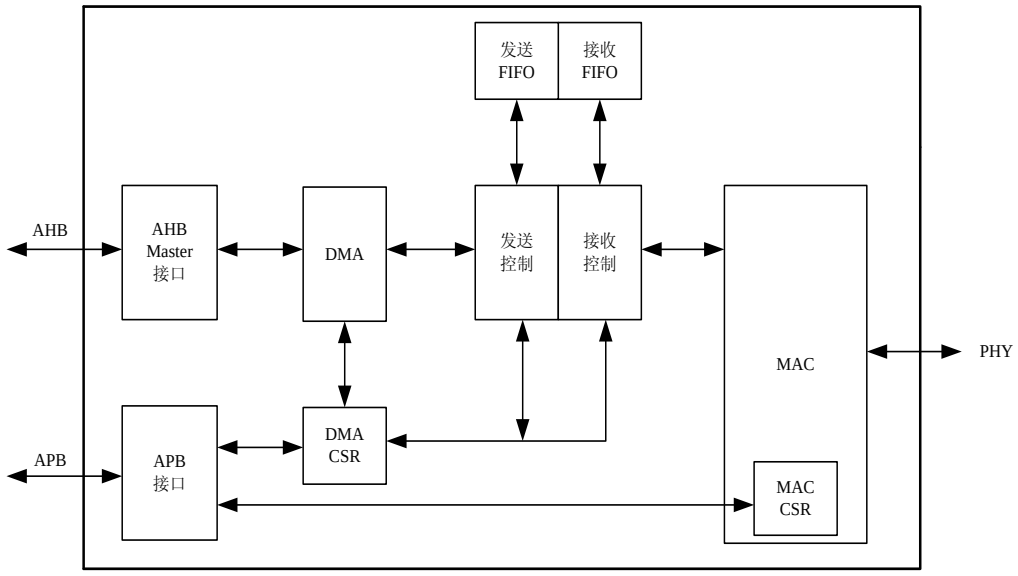


图 8-3 以太网 MAC 控制器结构图

MAC 模块支持以太网 PHY 的 RMII 接口。

CSR（Control & Status 寄存器）共占有 8KB 地址空间，分为两段：DMA CSR 和 MAC CSR。

AHB Master 接口为 DMA 模块与系统主机的接口。

APB 接口即 CSR 接口，用于读写访问 DMA CSR 和 MAC CSR。

MAC 模块实现数据在系统时钟域（AHB 时钟）和 MAC 时钟域（PHY 侧输入的时钟）间的变换和传输。

DMA 模块有独立的发送和接收引擎，进行系统内存与 MAC 间的数据搬运，将 CPU 的干预最小化，只在帧发送或接收结束以及其他一些条件（如发生错误）下中断 CPU。DMA 数据传输基于具有链表结构的 Descriptor 和数据缓冲区。

Descriptor 是主机与 MAC 控制器对收发数据缓存进行访问的数据结构，其内容主要是收发状态和收发数据缓存的起始地址，接收和发送各有一组 Descriptor。物理上，Descriptor 是系统内存中的一段数据，起始地址由主机分配。在分配好 Descripor 的起始地址后，主机该地址将写入 MAC 中的寄存器，以供 MAC 通过 AHB 总线来访问。

在发送数据前，软件把待发送的数据写入内存中分配的数据缓存中，然后将数据缓存的首地址和长度写入内存中分配的发送 Descriptor 空间内，最后启动 MAC 的发送引擎，从而开始发送。

类似地，接收数据前，软件需要分配一段内存空间作为接收数据缓存，并将接收缓存的首地址写入内存中分配的接收 Descriptor 空间，最后启动 MAC 的接收引擎。MAC 会通过 AHB 总线访问到接收 Descriptor 的内容从而得知接收数据缓存的地址，这样就会把接收到的报文写入数据缓存。

DMA 传输中使用到两个 Descriptor 列表，在接收和发送方向各有一个，这些列表的基址存放在相应的寄存器中，每个 Descriptor 的长度是 4 个字。Descriptor 列表分为两种结构：

一种是环形，另一种是链形，如图 8-4 所示。环形 Descriptor 列表中的 Descriptor 地址以基址为起始，以 4 为固定步长递增，每个 Descriptor 可以映射两片数据缓存，最后一个 Descriptor 链接到起始的 Descriptor 上，形成一个周而复始的环形。链形 Descriptor 列表中的第一个 Descriptor 的地址也是寄存器配置的基址，但下一个 Descriptor 的地址是在上一个 Descriptor 中配置写入的。链形 Descriptor 列表中的每一个 Descriptor 映射一个数据缓存。事实上，Descriptor 单元的结构在两种列表形态下完全相同，不同的是环形单元中存放 Buffer2 地址的字，在链形单元中存放的是下一个 Descriptor 的地址。

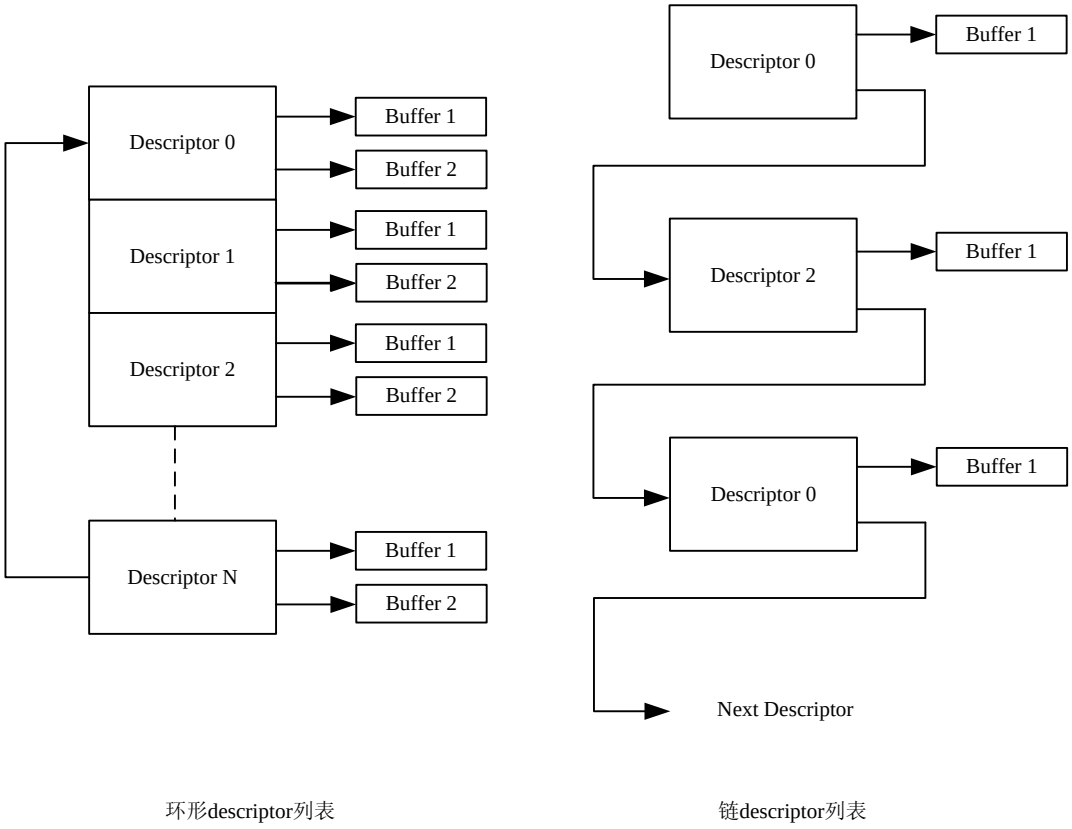


图 8-4 Descriptor 列表结构

8.3.4.1 收发过程

在了解了模块的整体结构后，本节简要描述 MAC 控制器的收发过程。

- 接收过程

接收时，MAC 模块按照 10/100Mbps 的带宽通过 RMII 接口接收并解析来自 PHY 芯片的数据流。MAC 模块首先查找标志 MAC 帧起始的前导码（Preamble）和帧起始定界符（SDF），然后继续向后解析报文，直到当前帧结束。如果收到的帧长度比一个时隙的长度（64 字节）小，MAC 将丢弃掉该帧（混杂模式除外）。如果收到的帧满足最小长度要求，则接着进行 CRC 校验。如果收到帧的 CRC 值不正确，就丢弃掉该帧（混杂模式除外）。假设一个长度有效的帧具有有效的 CRC 值，而且 MAC 不是工作在混杂模式下，则 MAC 将检查目的地址是否是如下三者之一：

（1）单播地址，帧的目的地址与接收站点的 MAC 地址相同

（2）组播地址，上层协议定义一些特殊的地址作为组播地址，帧中的目的地址是否可以通过 HASH 表命中。

（3）广播地址，目的地址为全 1 的地址。

这些地址表明该帧确实是以此网络端为目的地，那么 MAC 模块将把帧写入接收 FIFO 中，DMA 模块再将数据从 FIFO 中读取出写入接收数据缓存，随后软件开始处理这些数据。

- 发送过程

发送时，软件准备好待发送的数据，然后配置 Descriptor 列表，启动 DMA 模块将数据传输到发送 FIFO 中。MAC 模块从 FIFO 中读取出来，按照以太网 MAC 帧格式，将待发送的数据与目的地址，源地址，类型/长度字段进行组合，并根据数据长度添加适当的填充字段以达到 802.3 标准规定的最小帧长度，然后计算 CRC 校验作为 FCS 字段添加在帧尾，形成一个完整的 MAC 帧。在发送时，模块首先自动生成并发送前同步码和帧开始定界符，然后开始发送组装好的 MAC 帧。

在半双工模式下，在向 PHY 芯片发送之前，为了避免其他主机竞争媒体而产生的冲突，MAC 控制器通过监听载波信号来得知是否有其他网络端点在发送信息。该信号由 PHY 芯片提供。如信道忙，MAC 控制器会暂缓发送自己的数据，直到信号变为空闲时，才开始发送。当信道变为空闲后，MAC 控制器并不立刻发送数据，而是继续等待一个帧间间隔，目的是给物理层以及其他站点的 MAC 控制器处理上一个帧的时间。当一切准备就绪后，MAC 控制器再以 10/100Mbps 通过 RMII 的接口发送到 PHY 芯片。而在全双工模式下，不需要载波监听，待发送的帧只要等待一个帧间间隔就可以立刻发送，不需要考虑对端是否正在接收数据。

在了解了收发过程之后，用户需要深入理解本芯片中 MAC 控制器的 DMA 工作原理与过程，在以下的若干章节中，将详细说明这些内容。

8.3.4.2 默认模式 DMA 发送过程

DMA 发送过程如下：（默认模式）

1. 主机配置发送 Descriptor，并且在将要发送的数据写入发送数据缓存后，设置 TDES0 的第 31 位 OWN。
2. 当 MAC_OP_MODE 寄存器的 ST 位置位时，DMA 进入工作状态。
3. 进入工作状态后，DMA 轮询发送 Descriptor 列表。当 DMA 检测到 Descriptor 被主机占用（TDES0[31]=0），或者发送过程中出错时，发送将中断，并且将 MAC_STATUS 寄存器的第 2 位和第 16 位置位。发送模块跳到第 9 步。
4. 如果 DMA 获取的 Descriptor TDES0 OWN 字段为 1，那么 DMA 可以从该 Descriptor 中获取发送数据缓存的物理地址。
5. DMA 从存储器中获取发送数据给 MAC 模块，由 MAC 模块对其进行协议处理后通过接口发送给 PHY。
6. 如果一个以太报文跨越几个 Descriptor 单元，DMA 获取下一个 Descriptor，然后重复第 3、4 和 5 步，直至完整的报文发送完毕。
7. DMA 将发送状态写入 TDES0，释放 Descriptor。
8. 当一个完整的报文发送完毕，如果当前 Descriptor TDES1[31]置位，则将 MAC_STATUS 的第 0 位发送中断置位。然后 DMA 返回第 3 步。
9. DMA 进入暂停状态。当 DMA 收到一个发送轮询命令并且当前的发送 FIFO 空中断状态清零时，尝试重新获取 Descriptor，然后到第 3 步。

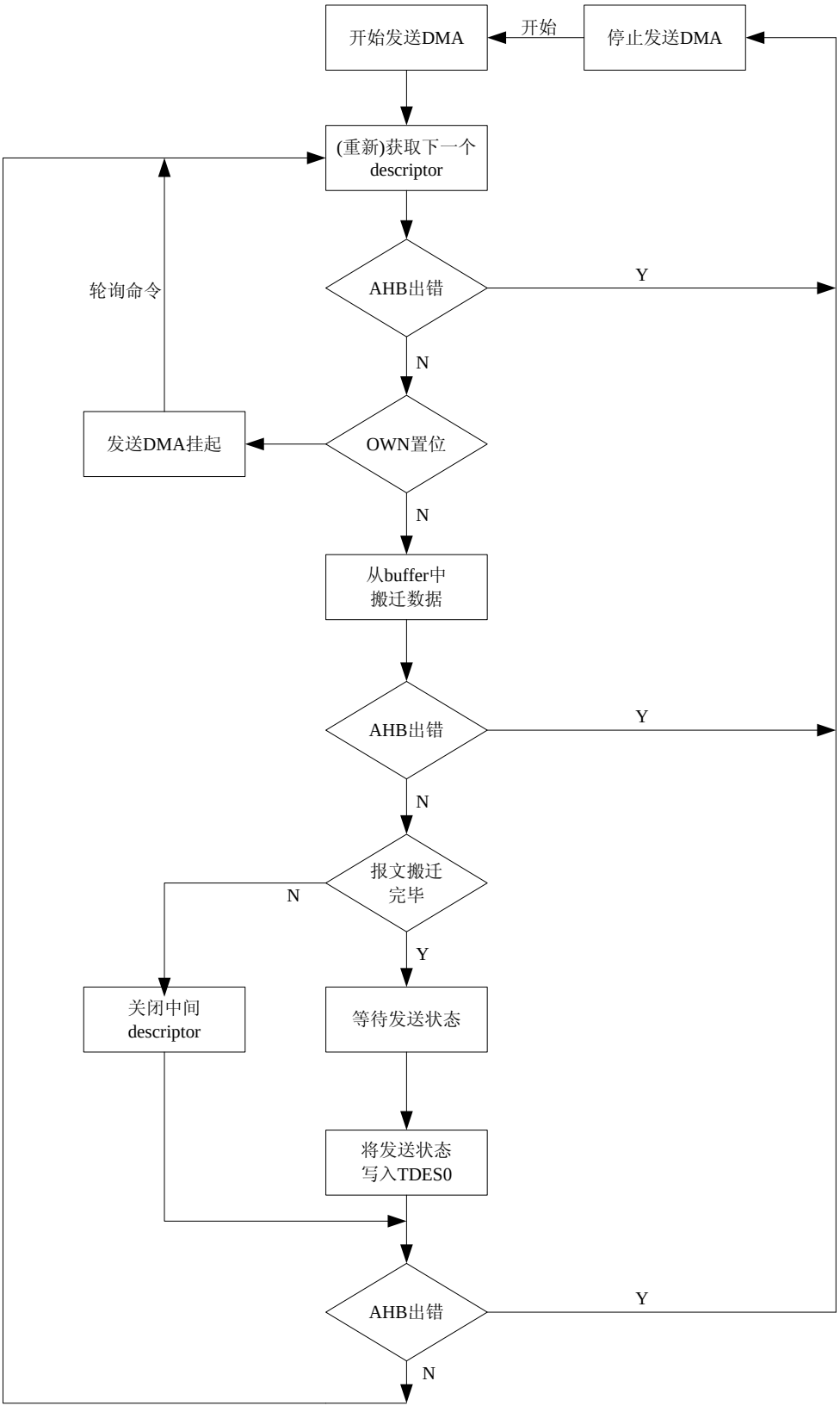


图 8-5 默认模式发送 DMA 工作流程图

8.3.4.3 OSF 模式 DMA 发送过程

OSF 模式 DMA 发送过程如下：

当进入工作状态时，如果 MAC_OP_MODE 寄存器的 OSF(Operate on Second Frame)置位，那么 DMA 工作在 OSF 模式下。在 OSF 模式下，DMA 在完成第一个数据缓存的发送之后，立即轮询 Descriptor 来获取第二个数据缓存的物理地址。

在 OSF 模式下，工作模式下的 DMA 按照以下顺序执行：

1. DMA 按照默认模式的第 1 步到第 6 步执行。
2. DMA 获取下一个 Descriptor 的控制权，同时继续占用上一个报文的最后一个 Descriptor。
3. 如果 DMA 可以获取下一个 Descriptor，那么从中读取数据缓存的物理地址；否则跳转到第 7 步，进入挂起状态。
4. DMA 将数据缓存中的数据搬迁到 MAC，一直到当前报文结束；如果报文跨越多个 Descriptor，那么释放中间的 Descriptor。
5. 待发送状态送达后，DMA 将其写入 TDES0，然后释放当前的 Descriptor。
6. 如果发送中断开启，那么置位发送中断，DMA 获取下一个 Descriptor，如果 Descriptor 的状态正常则转至第 3 步。如果获取的 Descriptor 状态中有向下溢出错误，那么 DMA 进入挂起状态，即跳到第 7 步。
7. 在挂起状态下，如果有未写入的状态，那么 DMA 将其写入对应的 TDES0，将对应的中断置位，然后重新进入挂起状态。
8. 在接受到一个发送轮询命令后，DMA 根据当前的状态跳至第 1 步或第 2 步，进入工作状态。

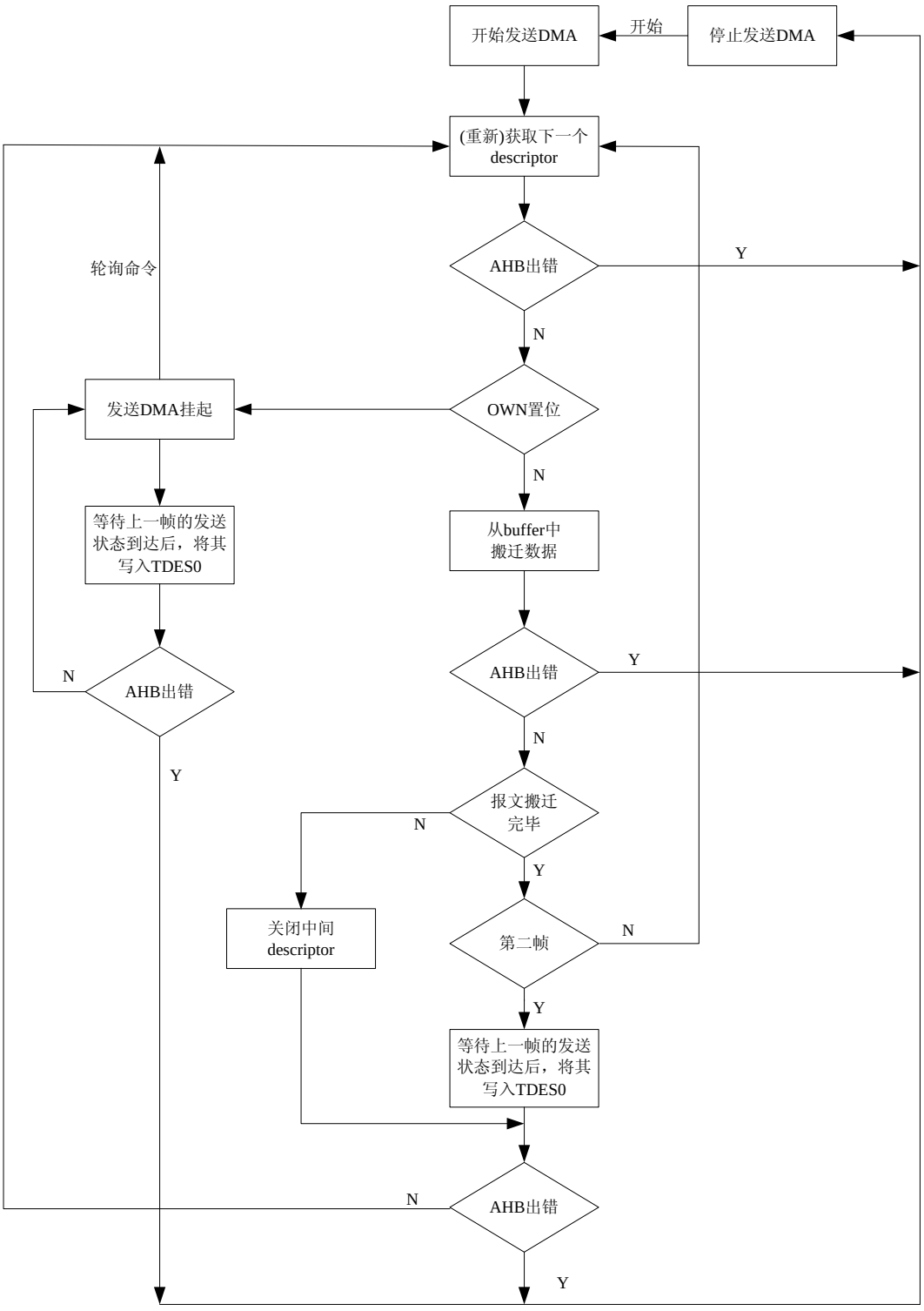


图 8-6 OSF 模式下发送 DMA 工作流程图

8.3.4.4 DMA 接收过程

DMA 接收过程如下:

1. 主机设置接收 Descriptor 单元并且将 RDES0[31]置位。

2. 当 MAC_OP_MODE 寄存器的 SR 位（bit 1）置位时，DMA 进入工作状态，此时 DMA 轮询接收 Descriptor 列表，尝试获取空闲的 Descriptor 来读取。如果没有空闲的 Descriptor，DMA 进入挂起状态，跳到第 9 步。
3. DMA 从获取的 Descriptor 中读得接收数据缓存的物理地址。
4. MAC 接收并处理报文，DMA 将报文写入数据缓存中。
5. 当数据缓存满或者当前报文完成接收，DMA 获取下一个 Descriptor。
6. 如果当前报文完成接收，DMA 跳到第 7 步。如果 DMA 不能使用获取的下一个 Descriptor，而当前帧并不是报文的最后一帧（包尾没有接收到），DMA 将 RDES0 的 DE 位置位（除非在 MAC_OP_MODE 寄存器的 bit24 中将 flush 操作关闭）。DMA 将当前 Descriptor 的 OWN 位清零，并将 RDES0 的 LS 位清零（如果 flush 操作开启，则 LS 置位），然后跳到第 8 步。如果 DMA 可以占用下一个 Descriptor，而当前帧不是报文的最后一帧，那么 DMA 释放当前 Descriptor，并将该 Descriptor 的 LS 清零，返回第 4 步。
7. DMA 将接收报文的状态写入当前 Descriptor 的 RDES0，同时将 OWN 位清零，将 LS 位置位。
8. DMA 检测最后一个 Descriptor 的 OWN 位。如果主机占用，MAC_STATUS 寄存器的 bit7 置位，DMA 将进入挂起状态，即第 9 步。如果 DMA 可以占用该 Descriptor，那么跳到第 4 步。
9. 在进入挂起状态之前，不完整的帧将从接收 FIFO 中清除（flush），flush 相关的控制在 MAC_OP_MODE 寄存器里设置。
10. 当接收到一个接收轮询命令，或 MAC 从 PHY 侧接收到报文并将其写入 FIFO 后，接收方向 DMA 退出挂起状态，进入第 2 步，重新获取下一个 Descriptor。

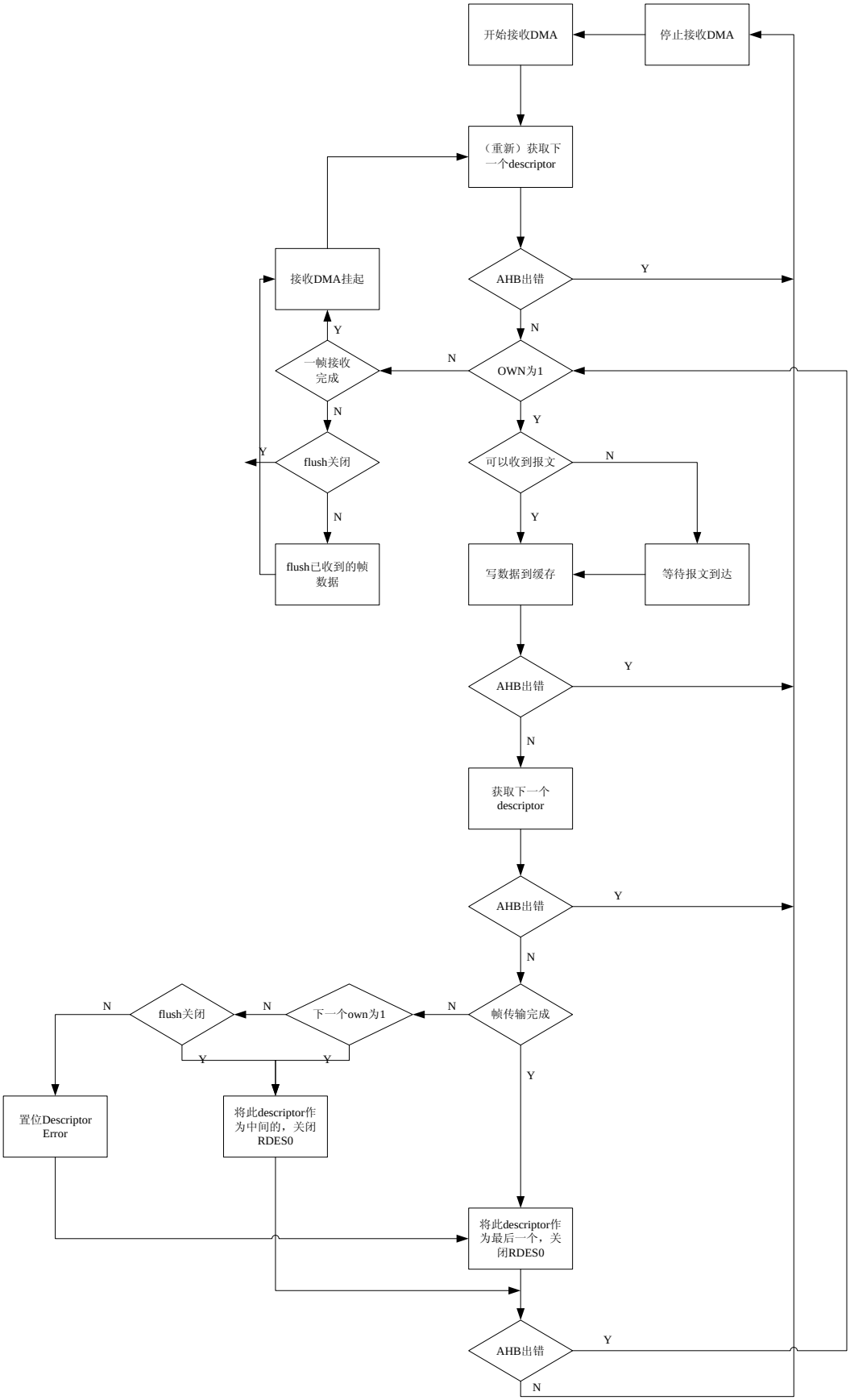


图 8-7 接收 DMA 工作流程图

下一节将对 Descriptor 单元的结构与含义做详细说明。

8.3.5 Descriptor

一个 Descriptor 单元由 4 个双字组成。在发送方向和接收方向各有指向发送和接收数据缓存的 Descriptor。

发送 Descriptor、接收 Descriptor 与总线一样，均为 Little-endian。

8.3.5.1 发送 Descriptor

由于发送带宽是由主机决定，所以 DMA 需要至少一个 Descriptor 来发送一帧。发送 Descriptor 的结构如表 8-2 所示。

表 8-2 发送 Descriptor

	31			0
TDES0	OWN	Status		
TDES1	Control Bits	Byte Count Buffer 2	Byte Count Buffer 1	
TDES2	Buffer 1 Address			
TDES3	Buffer 2 Address/Next Descriptor Address			

表 8-3 TDES0

位	说明
[31]	OWN 位。置位表示 Descriptor 被 DMA 占有；复位表示主机占有。当 DMA 结束一帧的发送或 Descriptor 分配的 buffer 为空时，DMA 清除此位。一帧的第一个 Descriptor 的 OWN 位置位应在同一帧后续的 Descriptor 的 OWN 位置位之后。
[30:16]	保留。
[15]	ES: Error Summary。出错信息摘要。值为 TDES0[14:1]的逻辑或，即为 1 表明出现错误，但具体错误需要进一步查看 TDES0[14:1]。
[14]	JT: Jabber Timeout。置位表示 MAC 发送器遇到 jabber 报文超时。只能在 MAC_CONFIG 寄存器的 JD 位为 0 时置位。
[13]	FF: Frame Flushed。置位表示软件发出清空命令使发送 FIFO 被清空。
[12]	PCE: Payload Checksum Error。如果在插入 checksum 过程中，遇到诸如 IP 头长度不够，报文类型不支持，或者插入 checksum 后，发送 FIFO 满等错误时， PCE 位置位。
[11]	LC: Loss of Carrier。发送时发生载波无效，即帧传输中 CRS 信号在一个或多个周期内无效。仅当帧传输时无冲突且 MAC 操作在半双工模式下有效。
[10]	NC: No Carrier。置位表示发送过程中发现 PHY 输入的载波监听信号无效。
[9]	LC: Late Collision。置位表示报文发送因发生冲突而取消（冲突持续 64 字节传输时间后）。Underflow Error 置位时 LC 无效。
[8]	EC: Excessive Collision。置位表示在连续 16 个冲突发生后（尝试发送当前报文）发送取消。如果 MAC_CONFIG 寄存器的位 9（禁止重试）被置位，当第一个冲突发生后发送取消，EC 置位。
[7]	VF: VLAN Frame。置位表示发送的帧为 VLAN 帧。
[6:3]	CC: Collision Count。4 位计数器表示帧发送完成之前冲突发生的次数。当 EC 置位

	CC 无效。
[2]	ED: Excessive Deferral。当 MAC_CONFIG 寄存器的延期检查位（位 4）置位，ED 置位表示因超过 24288 位传输时间的过度延期而导致发送终止。
[1]	UF: Underflow Error。DMA 在帧传输中发现空的发送缓冲区。发送过程进入挂起状态，UF 置位。
[0]	DB: Deferred Bit。置位表示载波存在而 MAC 在发送前延缓操作。仅半双工模式下有效。

表 8-4 TDES1

位	说明
[31]	IC: Interrupt on Completion。当置位表示发送完成中断位被置位。
[30]	LS: Last Segment。置位表示缓冲区包含帧的最后一段。当 LS 置位，TBS1 和 TBS2 应为非 0 值。
[29]	FS: First Segment。置位表示缓冲区包含帧的第一段。
[28:27]	CIC: Checksum Insertion Control。CIC 控制在以太网帧中插入 checksum。 当 CIC 为 2'b01 时插入 IPv4 头部 checksum；为 2'b10 时，假设头部的 checksum 字段已经存在，只插入 TCP/UDP/ICMP 有效载荷 checksum；为 2'b11 时，假设头部的 checksum 字段为全 0，从而计算所有的 checksum 字段并插入。
[26]	DC: Disable CRC。当置位，MAC 不在帧的结尾附加 CRC。此位仅当 TDES1[29]置位时有效。
[25]	TER: Transmit End of Ring。置位表示 Descriptor 列表已到达最后一个 Descriptor。回到基地址，形成环形结构。
[24]	TCH: Second Address Chained。置位表示 Descriptor 中的第二个地址是下一个 Descriptor 的地址，而不是缓冲区 2 的物理地址。TER 优先级高于 TCH。
[23]	DP: Disable Padding。置位时，MAC 不自动为小于 64 字节的帧添加 PAD 字节；复位时，MAC 自动为小于 64 字节的帧添加 PAD 字节和 CRC，CRC 的添加不考虑 DC 位（TDES1[26]）的设置情况。仅当 TDES1[29]置位时有效。
[22]	TTSE: Transmit Timestamp Enable。置位则使能发送帧的时间戳。仅当 TDES1[29]置位时有效。
[21:11]	TBS2: Transmit Buffer 2 Size。第二个数据缓冲区的大小（字节数）。TDES1[24]置位时无效。
[10:0]	TBS1: Transmit Buffer 1 Size。第一个数据缓冲区的大小（字节数）。如果 TBS1 为 0，DMA 忽略此缓冲区，而根据 TDES1[24]的值选择使用第二个数据缓冲区或下一个 Descriptor。

表 8-5 TDES2

位	说明
[31:0]	Buffer 1 Address Pointer。发送缓冲区 1 的物理地址。

表 8-6 TDES3

位	说明
[31:0]	Buffer 2 Address Pointer/Next Descriptor Address。发送缓冲区 2 的物理地址。如 TDES1[24]置位，则为下一个 Descriptor 的物理地址。

8.3.5.2 接收 Descriptor

在接收报文时，为了不丢失报文，DMA 需要至少两个 Descriptor 来接收一帧。接收方向的 Descriptor 结构如表 8-7 所示。

表 8-7 接收 Descriptor

	31			0
RDES0	OWN	Status		
RDES1	Control Bits	Byte Count Buffer 2	Byte Count Buffer 1	
RDES2	Buffer 1 Address			
RDES3	Buffer 2 Address/Next Descriptor Address			

表 8-8 RDES0

位	说明
[31]	OWN 位。置位表示 Descriptor 被 DMA 占有；复位表示主机占有。当 DMA 接收到一帧或该 Descriptor 对应的 buffer 填满时，DMA 会清 0 此位。
[30]	AFM：Destination Address Filter Fail。当此为置 1 时，表示当前帧不能通过目的地址过滤。
[29:16]	FL：Frame Length。该字段表示接收到报文（包含 CRC 字段）的长度。这里指一个完整的报文长度，所以当 Last Descriptor(RDES0[8])置为 1，并且当 Descriptor Error(RDES0[14])和 Overflow Error(RDES0[11])位为 0 时才有效。
[15]	ES:Error Summary。置位表示在接收报文的过程中遇到错误。该位只有在 Last Descriptor(RDES0[8])置位时才有效。
[14]	DE: Descriptor Error。置位时表示报文由于当前 Descriptor 对应的缓存不能存放当前报文而导致报文截断，同时 DMA 不再占有下一个 Descriptor。该位只在 Last Descriptor(RDES0[8]) 置位时有效。
[13]	SAF:Source Address Filter Fail。置位表示报文的源地址字段不能通过 MAC 中的源地址过滤。
[12]	LE：Length Error。置位时，表示实际收到的报文长度与报文 Length/Type 字段不相符。该位只在 Frame Type(RDES0[5])为 0 时有效，并且在 CRC 错误的情况下也无效。
[11]	OE: Overflow Error。当置位表示接收 FIFO 溢出。
[10]	VLAN: VLAN Tag。置位表示当前 Descriptor 指向一个 VLAN 报文，且 VLAN 域与 MAC 预设的值相符。
[9]	FS: First Descriptor。置位时表示当前 Descriptor 指向包含当前报文的第一个数据缓存。
[8]	LS: Last Descriptor。置位时表示当前 Descriptor 指向包含当前报文的最后一个数据缓存。
[7]	IPC Checksum Error。置位时表示由 MAC 计算得的 16 位 IPv4 报文头的 checksum 与实际得到的不同。
[6]	LC: Late Collision。置位表示在半双工模式下接受报文时发生 late conllision。
[5]	FT: Frame Type。置位时表示接收到的报文是一个以太报文(LT 字段大于或等于 16’h0600)。清零时表示接收的报文是一个 IEEE802.3 报文。当接收到小于 14 字节的 Runt 报文时，该位无效。
[4]	RWT: Receive Watchdog Timeout。置位表示接收报文时间超过 Receive Watchdog 预

	设时间。
[3]	RE: Receive Error。置位表示在接收有效信号 rxdv 有效时，rxerr 信号同时有效。
[2]	保留
[1]	CE: CRC Error。置位表示接收到的报文 CRC 错误。此位只有当 Last Descriptor(RDES0[8])有效时才有效。
[0]	保留

表 8-9 RDES1

位	说明
[31]	Disable Interrupt on Completion。置位时表示 MAC_STATUS 寄存器的 RI 位（bit6）对于当前包含报文结尾的数据缓存不起作用，即不会由于 RI 而产生中断。
[30:26]	保留
[25]	RER: Receive End of Ring。置位时表示当前 descirptor 是环形链表的最有一个 Descriptor，DMA 将返回到基址寄存器所指向的第一个 Descriptor。
[24]	RCH: Second Address Chained。置位时表示 Descriptor 中的第二个地址指向下一个 Descriptor，而非第二个数据缓存。当 RDES1[24]置位时，RBS2(RDES1[21:11])没有意义。RDES1[25]优先级高于 RDES1[24]。
[23:22]	保留
[21:11]	RBS2: Receive Buffer 2 Size。该字段的值为第二个数据缓存的字节数。字节数必须是 4 的整数倍。
[10:0]	RBS1: Receive Buffer 1 Size。该字段的值为第一个数据缓存的字节数。字节数必须是 4 的整数倍。

表 8-10 RDES2

位	说明
[31:0]	Buffer 1 Address Pointer。该字段用于保存指向第一个数据缓存（buffer1）的物理地址。

表 8-11 RDES3

位	说明
[31:0]	Buffer 2 Address Pointer/Next Descriptor Address。第二个数据缓存（buffer2）的物理地址。如 TDES1[24]置位，则为下一个 Descriptor 的物理地址。

8.3.6 Checksum

接收方向，以太网 MAC 模块支持 IP 报文头的 checksum 校验。发送方向，以太网 MAC 模块支持 TCP/IP 报文的 checksum 插入。以太网 MAC 模块只支持 IPv4 报文类型。

用户可以通过设置 MAC_CONFIG 寄存器的 IPC 位来开启 IP 报文头的 checksum 校验。MAC 模块计算接收到的 IP 报文头的 checksum 值，并与接收到的报文头中第 25 和 26 字节（带 VLAN 标签报文的第 29 和 30 字节）相比较，即将 checksum 值与 checksum 字段相比较，将比较的结果写入接收 Descriptor 的 RFC 位。如果 IP 头的 Length 字段小于 5，或者 IP 类型不等于 4，那么 IP 头 checksum 出错，并写入接收 Descriptor 的 RFC 位。

在发送方向，以太网 MAC 支持 TCP/IP 报文的 checksum 插入。发送 Descriptor DES1 的 CIC 位用以控制 checksum 的插入。CIC 位为非零时，checksum 插入模块工作。具体地，当 CIC 为 2'b01 时插入 IPv4 头部 checksum；为 2'b10 时，假设头部的 checksum 字段已经存在，只插入 TCP/UDP/ICMP 有效载荷 checksum；为 2'b11 时，假设头部的 checksum 字段为全 0，从而计算所有的 checksum 字段并插入。

如果在插入 checksum 过程中，遇到诸如 IP 头长度不够，报文类型不支持，或者插入 checksum 后，发送 FIFO 满等错误时，发送 Descriptor 的 PCE 位置位。

8.3.7 流控机制

由于 MAC 模块不支持硬件自动流控机制，即以接收缓存的水线来自动发送流控帧或置起反压信号以控制发送端或前级模块，所以只能通过软件来实现流控。

如果软件想阻止帧到来，可以通过配置流控寄存器 MAC_FLOW_CTRL 来发送一个带有参数的 Pause 帧，该参数指明了全双工中的另一方在开始继续发送数据前需要等待的时间。当网络对端接收到 Pause 帧后，将在指定的时间内停止发送数据。当停止时间过后，对端将从暂停的位置继续发送数据帧。Pause 帧能禁止对端发送数据帧，但它不影响对端发送 Pause 帧等 MAC 控制帧。

已发送了 Pause 帧的网络端，可以再发送一个时间参数为 0 的 Pause 帧取消剩余的暂停时间，即新收到的 Pause 帧将覆盖掉当前执行的 Pause 操作。类似地，该端也可以在前一个 Pause 时间还未结束时，发出另一个包含非零时间参数的帧延长暂停时间。

因为 Pause 操作使用标准的以太网 MAC 传输，所以不能保证接收者一定能收到帧。Pause 帧也可能出问题，而使接收者可能不知道对端曾发出了这样的帧。在软件设计 Pause 传输策略时必须考虑这样的问题。

8.3.8 Hash

在对接收到的多个 MAC 地址进行匹配时，为了快速查找到是否通过过滤，一般采用 Hash 查找表的方法。本 MAC 控制器内有一个 64 位的 Hash 表，从高位到低位寄存在 MAC_HASH_H 和 MAC_HASH_L 两个寄存器中。MAC 接收到一帧的 MAC 目的地址后，按照以下几个步骤进行查找匹配：

（1）按照以下多项式计算出 MAC 目的地址的 32 位 CRC 结果：

CRC_32 = $X^{32}+X^{26}+X^{23}+X^{16}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X+1$；（详见 IEEE 802.3 协议）

（2）对步骤（1）得到的值按位取反；

（3）对步骤（2）得到的值取高 6 位；

（4）步骤（3）的到 6 位数中最高位为 1，那么在 MAC_HASH_H 中进行查找，如果最高位为 0，则在 MAC_HASH_L 中查找；

（5）选定一个 32 位 Hash 表寄存器后，由其余 5 位的值决定对应选中寄存器的哪一位；即这 5 位的值为 0 对应寄存器第 0 位，为 1 对应寄存器第 1 位，为 31 对应寄存器第 31 位。如果对应寄存器位的值为 1，表示该 MAC 目的地址命中，该帧通过 MAC 目的地址的过滤；为 0 表示没有命中，没有通过 MAC 目的地址的过滤。

Hash 表的配置需要上层软件根据网络的特征对 MAC 地址进行 Hash 运算而得。

8.3.9 中断与出错

8.3.9.1 中断

在硬件或软件复位后，所有中断被禁止，需要通过配置 MAC_INT_EN 寄存器来开启部分或全部中断。如果 MAC_INT_EN 寄存器中相应中断位被使能，那么当所使能的中断条件触发时，MAC_STATUS 寄存器中的相应中断位被置位，MAC 控制器进入挂起状态。软件可以通过轮询的方式读取到中断信息，待响应完成后，写 1 到 MAC_STATUS 寄存器中对应的中断位，则可清除中断。

MAC 控制器中的中断分为正常中断和异常中断。

正常中断包括有如下子中断：

- 发送中断：表示一帧发送完成。当一帧发送完成后，发送 Descriptor TDES1 的最高位清零，软件可以读取发送信息。
- 不能访问发送数据缓存中断：表示 DMA 模块不能获取下一个 Descriptor，此 Descriptor 正在被主机占用。
- 接收中断：表示一帧接收完成。当一帧接收完成后，接收 Descriptor RDES1 的最高位清零，软件可以对该 Descriptor 进行访问。
- 较早接收中断：表示第一个接收数据缓存已经被 DMA 写入，此中断可以告知主机接收的数据已经到达接收数据缓存。

异常中断包括有如下子中断：

- 发送停止中断：在一个报文的发送没有完成时，发送不能继续进行，将会产生此中断。如发生发送停止中断，软件可以重新发送此报文。
- 发送超时中断：表示发送的报文长度超过 2048 字节（当 Jumbo 报文允许时，大于 10240 字节）而发送超时。
- 接收 FIFO 上溢出中断：表示接收 FIFO 上溢出。此种情况的出现会造成大量丢包，网络中断的情况。
- 发送 FIFO 下溢出中断：表示在发送过程中发送 FIFO 下溢出，从而造成中断。在此情形下，会出现断包传输到网络上。
- 不能访问接收数据缓存中断：在接收过程中，DMA 无法访问接收数据缓存，从而产生中断以通知主机。
- 接收停止中断：在一个报文没有完全接受时，接收不能继续进行，将会产生此中断，此中断发生时，意味着将有报文在我端丢失。
- 接收超时中断：表示接收的报文长度超过 2048 字节（当 Jumbo 报文允许时，大于 10240 字节）而超时产生中断。
- 较早发送中断：表示待发送的报文完全写入发送 FIFO。
- 严重总线错误中断：表示在数据缓存或 Descriptor 的操作中出现严重的总线错误。

8.3.9.2 出错

MAC 控制器在发送或接受的工作过程中会遇到出错的情况，而且即使工作过程正常，也会出现报文内容本身的错误。MAC 控制器会将这些出错信息在将 Descriptor 释放之前写入各自对应的标志位。这样，在主机轮询获取到 Descriptor 访问权后，可以读取上次操作的过程和结果情况。各种出错情况的具体说明，请参考本文档中 Descriptor 一章节。

8.4 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

8.5 编程指导

当前版本用户手册暂不提供详细编程指导。

9 DMA 控制器

DMA 控制器能够实现 GSC3281 内存到内存，内存到外设、外设到内存以及外设到外设之间的数据传输。

9.1 概述

DMA 控制器包括两个 AHB Master，分别接在 GSC3281 SOC 的 AHB1 和 AHB2 总线上，其中 Master1 接在 AHB1 总线上，Master2 接 AHB2 总线上。主要特性包括：

1. 支持 AHB 总线协议；
2. 支持 4 通道；
3. 支持硬件握手及软件握手，最多支持 16 个外设硬件接口；
4. 支持字节、半字及字传输；
5. 支持地址递增、递减或者不变；
6. 支持发散/聚集（Scatter/Gather）传输；
7. 通道 0 支持多块传输，其它通道不支持多块传输

9.2 功能说明

DMA 主要模块如所示。

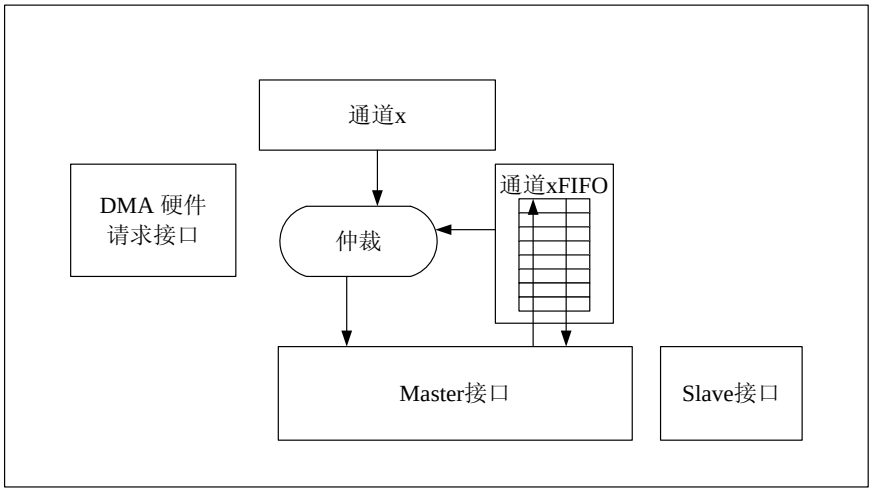


图 9-1 DMA 框图

图 9-1 中，GSC3281 的 DMA 控制器有两个 AHB Master 接口。Master 接口负责从源端读取数据并写到目的端，有两个 Master 表明可以有两个独立的通道同时操作。Master 的选择在使用过程中一定要根据实际情况正确配置，否则会出现错误，不能启动 DMA，详细配置含义见寄存器说明。DMA 控制器的 AHB slave 接口挂在 AHB1 总线上，负责 DMA 的配置。通道及 FIFO 负责数据的传输。DMA 硬件请求单元负责与外设握手。图 9-2 是 GSC3281 中 DMA 控制器与相关外设连接示意图。DMA 控制器的两个 Master 分别连接在 AHB1 和 AHB2 总线上，DMA 控制器可以通过 AHB2 总线访问内存，可以通过 AHB1 总线访问 SPI1 外设，或者跨

过 AHB1 总线访问 APB 总线上的外设如 SPI0，UART3 等。APB 总线上有哪些外设可以通过 DMA 访问，可以参考 9.2.6 节及表 9-2 的描述。在使用 DMA 控制器中，要正确选择 Master，通过图 9-2 可以看出通过 Master2 访问的只有内存，其它外设都是通过 Master1 访问。

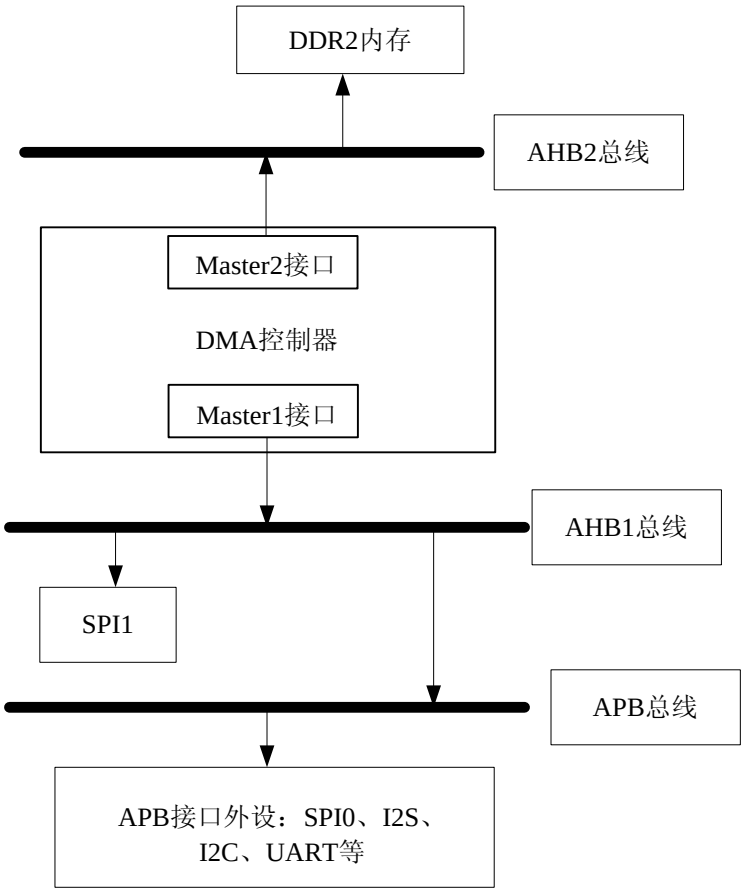


图 9-2 DMA 与相关外设连接图

9.2.1 DMA 类型

DMA 传输包括单个块和多个块传输，但只有通道 0 既支持单块传输也支持多块传输，其它通道只支持单块传输。

多块传输包含以下 3 种形式：

1. 链表形式，也就是块链式传输。链表寄存器指向下一个描述符的位置。每组描述符描述了每块传输的地址及控制信息，DMA 读取每个描述符，根据描述符启动 DMA 传输；
2. 自动重新装载，DMA 在每块传输结束自动重新装载通道寄存器；
3. 块间为连续地址；

而控制寄存器 DMA_CTLx（x=0~3，表示通道）的更新只有通过链表更新或者自动更新两种方式。

表 9-1 是 DMA 各种传输类型所对应的各参数更新方式，其中 LOC 是 DMA_LLP0 寄存器的 LOC 域，LLP_SRC_EN，RELOAD_SRC，LLP_DST_EN，RELOAD_DST 可以参考控制寄存器 DMA_CTL0 和配置寄存器 DMA_CFG0 的说明。

表 9-1 传输类型及参数更新方式

传输类型	LLP.LOC=0	LLP_SRC_EN	RELOAD_SRC	LLP_DST_EN	RELOAD_DST	CTL0,LLP 更新方式	SAR 更新方式	DAR 更新方式	写回
1.单块或多块的最后一块	是	0	0	0	0	不更新	不更新	不更新	不写回
2.源地址连续自动装载	是	0	0	0	1	自动重载	连续	自动重载	不写回
3. 目的地址连续自动装载	是	0	1	0	0	自动重载	不更新	不更新	不写回
4.自动装载多块传输	是	0	1	0	1	自动重载	自动重载	自动重载	不写回
5. 单块或多块的最后一块	不	0	0	0	0	不更新	不	不	写回
6.源地址连续的链表传输	不	0	0	1	0	链表更新	连续	链表	写回
7 源地址自动装载的链表传输	不	0	1	1	0	链表更新	自动重载	链表更新	写回
8.目的地址连续的链表传输	不	1	0	0	0	链表更新	链表更新	连续	写回
9.目的地址自动装载的链表传输	不	1	0	0	1	链表更新	链表更新	自动重载	写回
10. 链表传输	不	1	0	1	0	链表更新	链表更新	链表更新	写回

9.2.2 链表多块传输

GSC3281 只有通道 0 支持链式多块传输。在启动 DMA 前，DMA 控制器从系统内存里读取链表描述符来编程配置通道寄存器。链表描述符包括：SAR0, DAR0, LLP0, CTL0, SSTAT0, DSTAT0。图 9-3 画出了 DMA 的链表描述符，在图中，DSTAT, SSTAT 由硬件自动更新并写回到链表描述符相应的位置，软件不需要对这两个位置编程，但软件必须留出这两个地址。另外

链表描述符中的 LLP0(1)，LLP0(2)……是指向下一个链表描述符的地址，这个地址的低两位用于 Master 选择，必须要按寄存器 LLP 定义正确选择 Master。

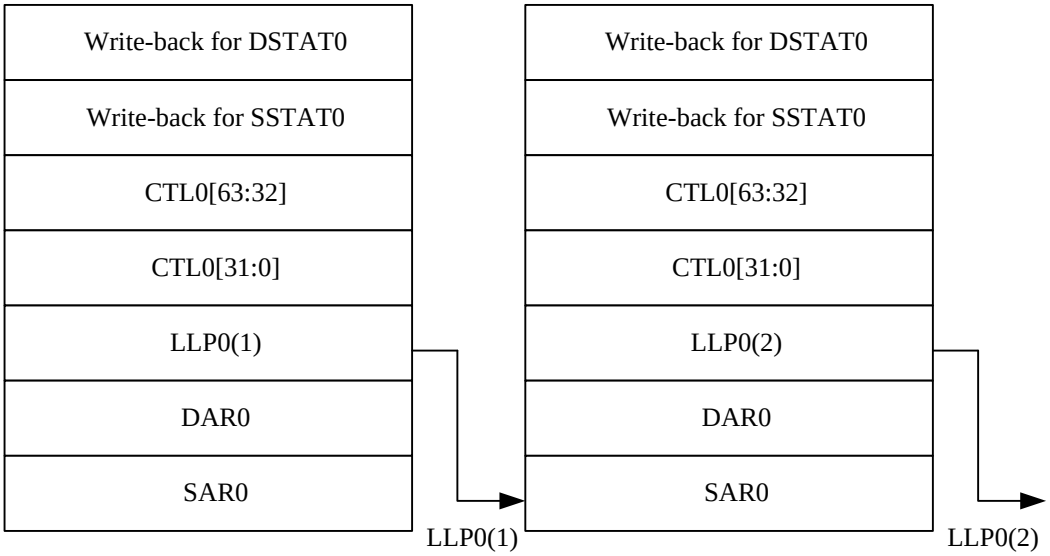


图 9-3 链表描述符

9.2.3 发散和聚集传输

发散和聚集传输分别对应目的端和源端。对发散传输，目的端地址在达到发散边界时以发散间隔增加或者递减。发散间隔是通过 DSR 寄存器的 DSI 域来配置的。控制寄存器 DMA_CTLx 的 DST_SCATTER_EN 域用于使能发散传输，地址递增还是递减由控制寄存器 DMA_CTLx 的 DINC 域控制。聚集传输是相对源端传输的，它的使能与控制和发散传输类似，可以参考相关寄存器说明。

图 9-4 和图 9-5 是发散和聚集传输的示意图，图中说明了发散聚集间隔、传输宽度等关系。

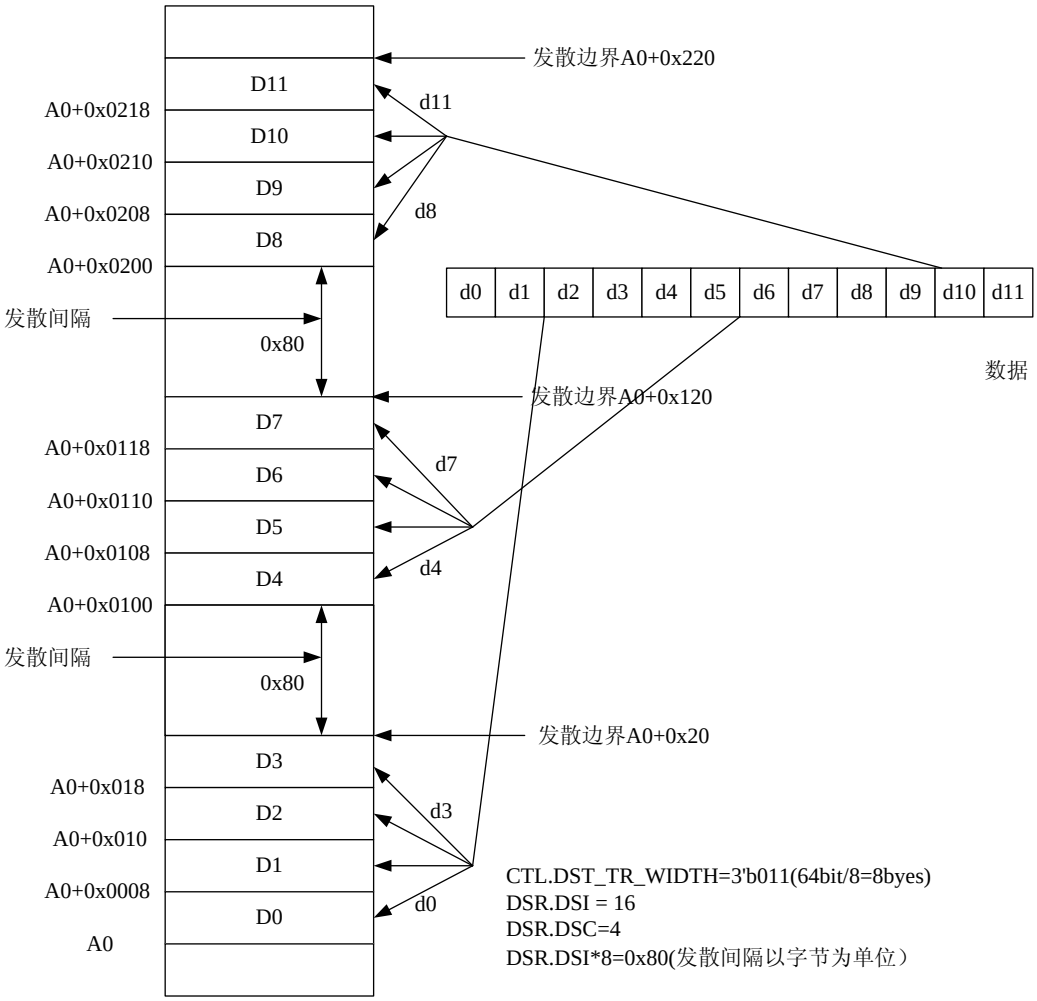

发散边界A0+0x220
D11
d11
A0+0x0218
D10
A0+0x0210
D9
A0+0x0208
d8
D8
A0+0x0200
d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11
发散间隔
0x80
数据
发散边界A0+0x120
D7
A0+0x0118
d7
D6
A0+0x0110
D5
A0+0x0108
d4
D4
A0+0x0100
发散间隔
0x80
发散边界A0+0x20
D3
A0+0x018
d3
D2
A0+0x010
D1
A0+0x0008
d0
D0
A0
CTL.DST_TR_WIDTH=3'b011(64bit/8=8byes)
DSR.DSI = 16
DSR.DSC=4
DSR.DSI*8=0x80(发散间隔以字节为单位）

图 9-4 目的端发散传输

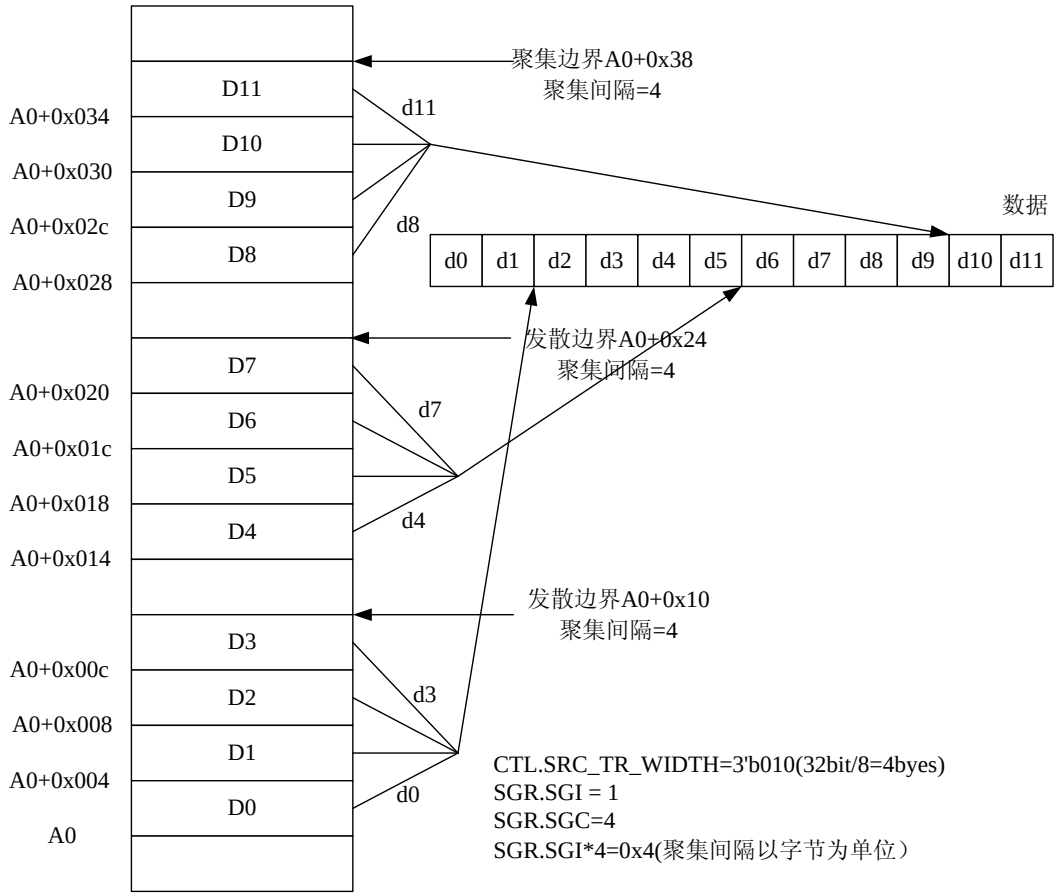


图 9-5 源端聚集传输

9.2.4 DMA 寄存器访问及非法访问

DMA 控制器所有的寄存器都是 64 位宽，地址是 64 位对齐的。有些寄存器的高 32 位是保留的，对保留位的写操作无效，被忽略，所有对保留位的读操作，返回值都是 0。有些寄存器的高 32 位是有意义的，由于 GSC3281 的地址和数据位宽是 32 位的，所以这些寄存器的高 32 位和低 32 需要分开操作，例如寄存器 DMA_CTL0，低 32 位操作的偏移地址是 0x18，高 32 位的偏移地址是 0x18+0x4。

以下几种都是非法寄存器访问。

1. 通道使能后，对寄存器 SARx，DARx，LLPx，CTLx，SSTATx，DSTATx，SSTATARx，DSTATARx，SGRx， DSRx（x=0~3）的写；
2. 对 ClearTfr 等中断清除寄存器的读；
3. 对 StatusTfr 等中断状态寄存器的写；
4. 对 StatusInt 寄存器的写；
5. 对 DmaIdReg 寄存器的写；

以上操作均为非法操作，返回给 AHB 总线错误响应，在实际应用中应该避免非法操作。

9.2.5 DMA 软件握手

当外设需要通过 DMA 传输数据时，外设可以给 CPU 或中断控制器发出中断，在中断服务程序里编程 DMA 软件握手寄存器来初始化及控制 DMA 传输。如果用软件握手机制，第

一步需要配置外设，正确配置外设的 FIFO 阈值等参数，第二步配置 DMA 控制器的控制寄存器和通道寄存器，第三步，在外设发中断后，在中断服务程序里，配置 DMA 的软件握手寄存器。

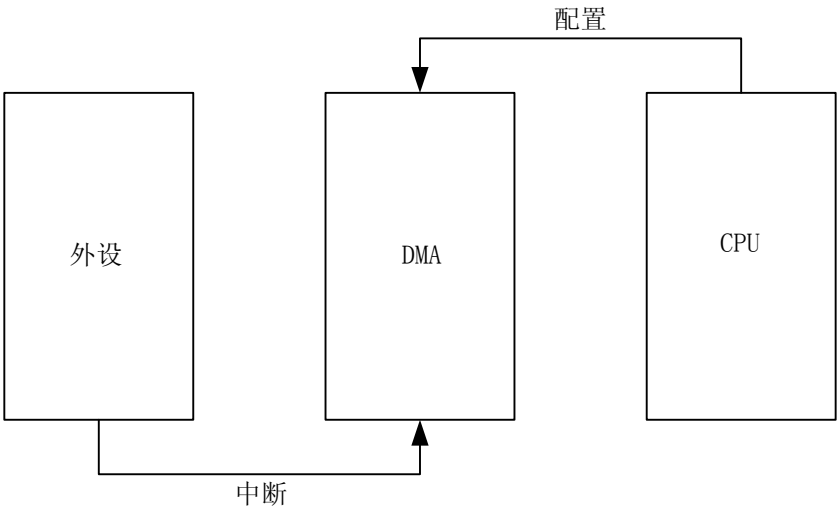


图 9-6 DMA 外设软件握手

9.2.6 DMA 硬件握手及编号

外设也可以通过硬件握手接口信号来告诉 DMA 控制器是否准备好传递或接收数据。图 9-7 是外设、DMA 控制器以及 CPU 之间硬件握手示意图。外设通过 DMA 请求信号 dma_req 向 DMA 控制器发出请求，DMA 控制器根据请求号判断是哪个外设发出的请求，然后发起 DMA 操作。DMA 控制器最多支持 16 个外设 DMA 请求，不同外设分配固定的硬件接口号，使用时，必须参考表 9-2 来正确配置 CFGx（x 为 0~3）寄存器。

GSC3281 系统中，所有需要通过 DMA 控制器进行 DMA 操作的外设，都在硬件上分配好了固定的硬件接口号，相对于软件握手机制，使用硬件握手更加方便。实际应用中，推荐使用硬件握手进行 DMA 操作。

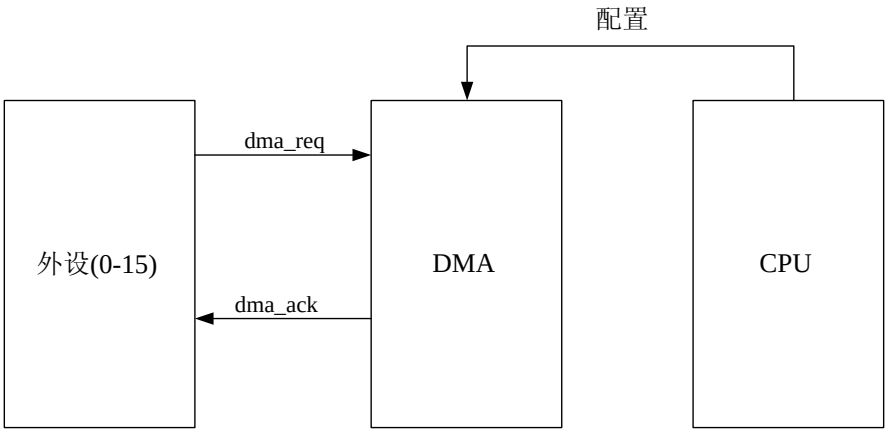


图 9-7 DMA 外设硬件握手

表 9-2 DMA 的外设硬件接口号

DMA 硬件接口号	对应的外设
0	串口 3 的发送方向
1	串口 3 的接收方向
2	串口 4 的发送方向
3	串口 4 的接收方向
4	串口 5 的发送方向
5	串口 5 的接收方向
6	串口 6 的发送方向
7	串口 6 的接收方向
8	SPI0 的发送方向
9	SPI0 的接收方向
10	SPI1 的发送方向
11	SPI1 的接收方向
12	I2S 的发送方向
12	I2S 的接收方向
14	I2C 的发送方向
15	I2C 的接收方向

9.2.7 DMA FIFO 模式

FIFO 模式选择决定了 DMA 控制器什么时候向 AHB 总线发起突发传输请求，也就是 FIFO 内有多少空间或有多少数据时，发起 DMA 的突发传输。软件可以通过配置寄存器 DMA_CFGx 的第 33 位即 FIFO_MOD 位来选择哪种模式。当选为 0 时，表示当 FIFO 里有能够进行一次 AHB 传输所需要的 FIFO 空间或者数据时，就发起 AHB 传输。例如传输宽度为半字（16 位），当 FIFO 里有一个半字空间或者 FIFO 里有一个 16 位的数据时，DMA 就发起单个 AHB 写或者读操作。当选为 1 时，FIFO 里的空间或者数据数目大于或等于 FIFO 深度的一半时，才发起 AHB 操作，当然如果是最后传输的结尾部分，DMA 会按实际需要的空间或数据进行 AHB 操作。当 FIFO_MODE 为 1 时，相对于 FIFO_MODE 为 0 时，每个块的传输需要的突发长度会长一些，相应的突发个数就会少一些，这样总线的利用率就会高一些，但是由于 FIFO 里的空间或数据必须等到等于或者大于 FIFO 深度一半才发起 AHB 传输，而 FIFI_MODE 为 0 则不需要，所以 FIFO_MODE 为 0 时，DMA 的延迟可能会降低一些。总之，如果想要高的总线利用效率，可以将 FIFO_MODE 设为 1，如果想要小的 DMA 延迟，将 FIFO_MODE 设为 0。

9.2.8 DMA 中断

DMA 控制器中断在 GSC3281 中的中断号是 0 号，以下几种情况都会产生 DMA 中断：

1 DMA 传输结束。
2 块传输结束。
3 单个突发传输结束。
4 错误产生时。

DMA 控制器支持中断使能和屏蔽，具体请参考中断寄存器的含义及配置。

9.2.9 DMA 的传输类型及流控

DMA 控制器支持内存到内存，内存到外设，外设到内存，外设到外设四种传输类型，在 GSC3281 中，由于与 DMA 连接的外设都没有流控功能，所以流控都是由 DMA 控制器完成。在配置 DMA 的控制寄存器的 TT_FC 域时，需要根据应用，正确配置好 TT_FC。TT_FC 的编码见表 9-3。

表 9-3 DMA 传输类型及流控编码

TT_FC	传输类型	流控
0x0	内存到内存	DMA 控制器
0x1	内存到外设	DMA 控制器
0x2	外设到内存	DMA 控制器
0x3	外设到外设	DMA 控制器
0x4-0x7	Reserved	Reserved

9.2.10 DMA 传输长度及传输宽度

突发传输的传输长度 MSIZE，即每次源端或者目的端突发交易的长度，以传输宽度为单位，传输宽度对应于 AHB 协议的 hsize，对于非内存类的外设，一般将传输宽度配置成外设的 FIFO 宽度。传输宽度必须小于或者等于 GSC3281 的总线宽度（32 位），实际使用 GSC3281 时，传输宽度最大只能配置为 32 位。

表 9-4 DMA 传输长度 MSIZE 编码

SRC_MSIZE/DST_MSIZE	传输的数据个数（宽度是 TR_WIDTH）
0x0	1
0x1	4
0x2	8
0x3	16
0x4	32
0x5	64
0x6	128
0x7	256

表 9-5 DMA 传输宽度

SRC_TR_WIDTH/DST_TR_WIDTH	传输宽度（单位是位 bits）
0x0	8
0x1	16
0x2	32
0x3	64
0x4	128
0x5	256
0x6	256
0x7	256

在与外设进行 DMA 传输时，需要合理设置 MSIZE 及外设的 FIFO 阈值，否则将会产生外设的 FIFO 上溢出（对于发送 FIFO）或者下溢出（对于接收 FIFO）。以下以 SPI0 为例说明 MSIZE 及外设 FIFO 阈值的设置，这里 DMA 控制器的传输宽度 TR_WIDTH 设置为 16 位。

1. DST_MSIZE 及外设发送 FIFO 的阈值设置：

当 SPI0 的发送 FIFO 里的待发送数据的数目小于或者等于发送 FIFO 的发送阈值时（参考 SPI 控制器的寄存器 SPI_TXFTLR，地址为 0xBC1010080），SPI0 控制器将向 DMA 控制器发出 DMA 发送请求，DMA 控制器将向 SPI0 的发送 FIFO 里发送长度为 DST_MSIZE，宽度为 DSTTR_WDITH 的突发数据。

如图 9-8 所示外设 SPI0 通过 DMA 发送数据。SPI0 的 FIFO 深度为 8，当软件设置 SPI_TFLR 寄存器为 2 时，即发送阈值为 4，当 SPI 的发送 FIFO 里的数据数目小于或者等于 4 时，SPI0 控制器就会发出 DMA 请求，DMA 控制器就可以往 SPI0 的发送 FIFO 里写数据。从图中可以看出，如果设置 DMA 的 DST_MSIZE 为 1（DMA 的传输长度为 4），即 DMA 向 SPI0 的发送 FIFO 里写长度为 4 的突发数据，这时如果 SPI0 在 DMA 往 SPI0 的发送 FIFO 写数据的同时没有往 SPI0 总线上发送数据，即 SPI0 的发送 FIFO 里原来的 4 个数据还在 FIFO 里面，则当 DMA 写完 4 个数据时，SPI0 的发送 FIFO 就正好全部满了。如果 DMA 的 DST_MSIZE 设置的传输长度大于 4，即 DMA 往 FIFO 里写多于 4 个数据，这时 SPI0 的发送 FIFO 有可能发生上溢出。所以 DMA 的传输长度 DST_MSIZE<=外设的发送 FIFO 深度-外设的发送阈值。

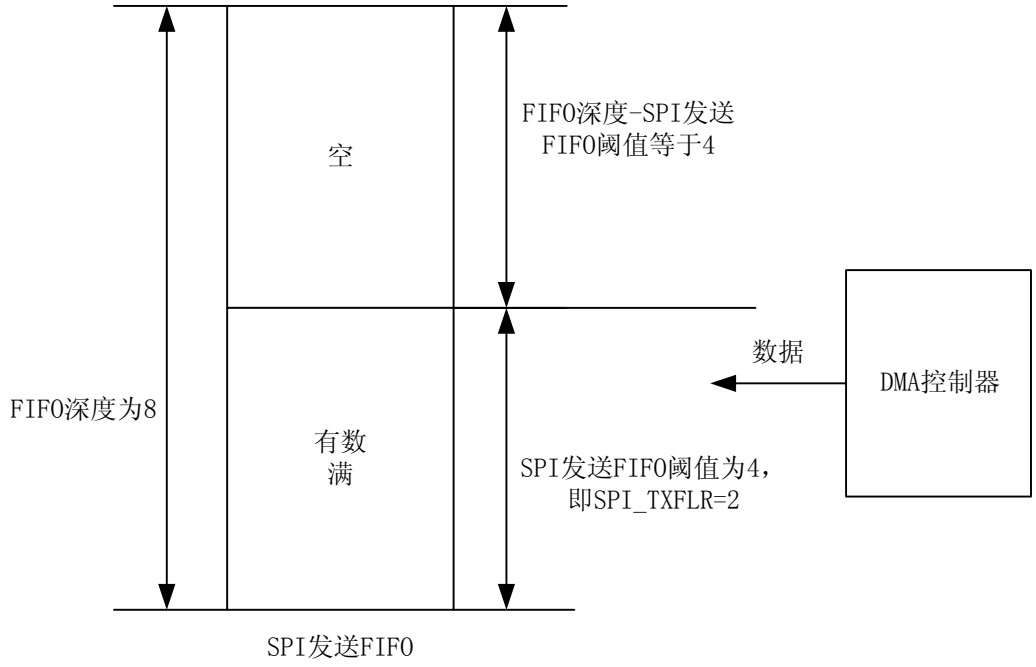


图 9-8 DMA DST_MSIZE 和外设 FIFO 设置

2. SRC_MSIZE 及外设接收 FIFO 的阈值设置：

同样的道理，对于 SRC_MSIZE 的设置也要考虑外设接收 FIFO 的接收阈值。同样以 SPI0 接收数据为例，如图 9-9 所示。SPI0 的接收 FIFO 的接收阈值为 4（SPI_RXFLR=2），FIFO 深度仍然为 8，当接收 FIFO 里的数据多余或者等于 4 时，SPI0 就向 DMA 发起读请求。这时 DMA 的读取突发长度为 4（SRC_MSIZE=1），即 DMA 从 SPI0 的接收 FIFO 里读取长度为 4 的突发数据，如果此时 SPI0 没有从 SPI0 的总线上接收新数据，那么 DMA 的 4 个数据读取将把 SPI0 的接收 FIFO 全部读空，所以对于 DMA 的读取数据时的传输长度必须小于或者等于外设的接收 FIFO 的接收阈值，否则就有可能产生外设的接收 FIFO 的下溢出。

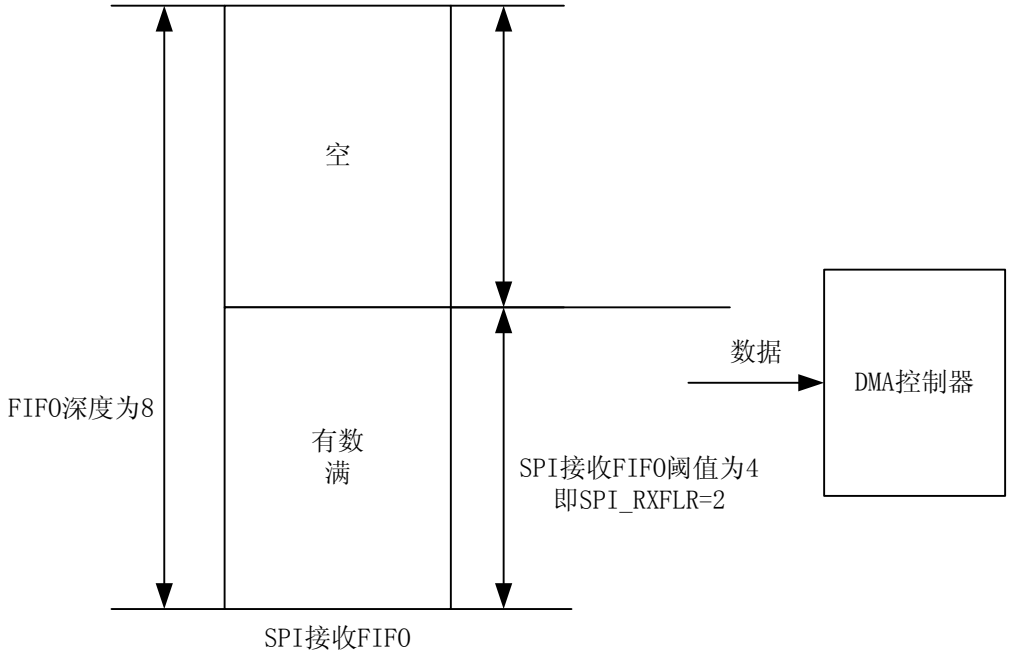


图 9-9 DMA SRC_MSIZE 和外设 FIFO 设置

以上以 SPI0 为例说明 MSIZE 和 FIFO 阈值的设置关系，不同外设的 FIFO 阈值寄存器含义可能定义不一样，使用时需要参考相关外设的寄存器说明。

9.2.11 DMA 传输结束及提前终止

在正常操作下，软件通过向通道使能寄存器的 CH_EN 位里写 1 来使能某个通道，硬件在传输结束时自动清除 CH_EN 位。

软件可以配合使用通道配置寄存器 DMA_CFGx 的 CH_SUSP 位和 FIFO_EMPTY 位来禁止某个通道：

1. 如果软件想在传输结束之前要停掉某个通道传输，软件可以设置 CH_SUSP 位来告诉 DMA 挂起所有源端传输，这时，通道 FIFO 里将不再接收新的数据；
2. 软件这时可以轮询 FIFO_EMPTY 位，直到查到 FIFO 是空的；
3. 一旦 FIFO 是空的，这时软件可以清掉 CH_EN 位；

传输非正常中止，在 DMA 传输过程中，软件突然清除 DMA_ChEnReg.CH_EN 位，这时通道并不一定马上被禁止，软件必须轮询 DMA_ChEnReg.CH_EN 位来确保某个通道是不是已经被禁止（读 CH_EN 位返回 0）。如果软件直接清除 DMA 配置寄存器 DMA_DMACfgReg[0] 来禁止所有通道，软件也必须要轮询 DMA_ChEnReg.CH_EN 位来确保某个通道是不是已经被禁止（读 CH_EN 位返回 0）。

所以在 DMA 使用时，首先需要判断通道是否是空闲的（读 CH_EN 位返回 0）。

9.2.12 DMA 编程流程

图 9-10 给出了链式多块传输的完整编程流程，在实际使用 DMA 时，可能不是多块传输或者不是链式传输，有些设置可以不用设置，同样可以参考下图进行编程。

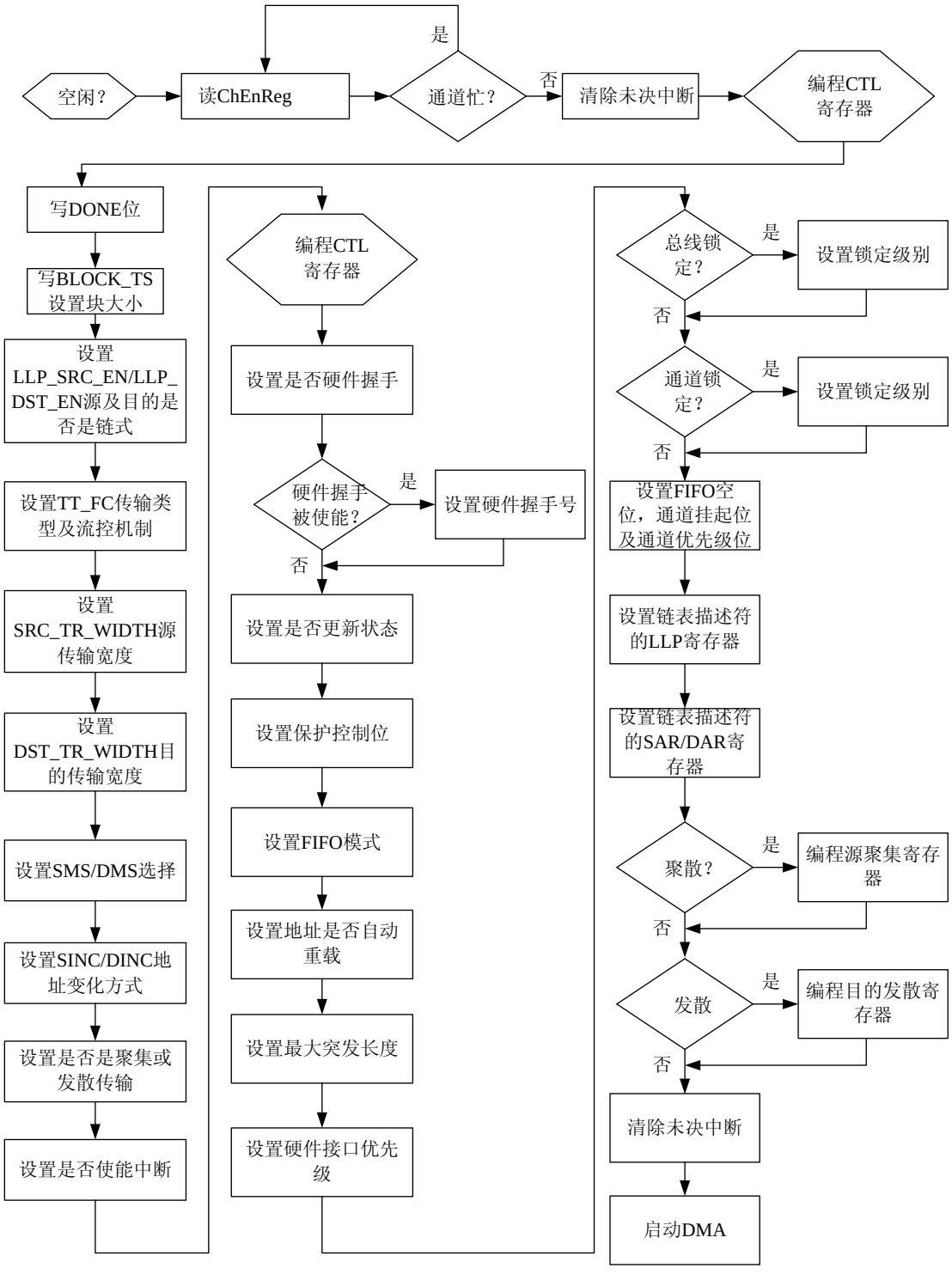


图 9-10 DMA 编程流程

9.3 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

9.4 编程指导

当前版本用户手册暂不提供详细编程指导。

10 中断控制器

10.1 概述

GSC3281 芯片所有片内与片外中断源首先汇总到中断控制器，中断控制器根据预先设置进行处理之后向 CPU 发出中断请求，CPU 将在适当时刻响应中断请求并进行中断处理。如果 CPU 接收到中断请求时正处于休眠状态，则 CPU 将退出休眠状态并立即响应中断。

中断控制器一共有 31 个中断源，每个中断源基本上与一个设备相对应，根据设备的不同，每一个中断源内部可能包含若干个子中断源，由中断处理程序进一步对多个子中断源进行判断与相应处理。GSC3281 中断控制器中所有的中断源具有平等的优先级和相同的中断入口，CPU 从相同入口进入中断处理程序之后，按照软件分配的中断优先级分别进行中断处理。

10.2 功能特性

GSC3281 中断控制器具有如下的功能特性：

- 31 个中断源；
- 高电平触发；
- 每一个中断可分别进行使能与屏蔽；
- 所有中断源具有相同的中断优先级；
- 可通过软件强制某一个中断源产生中断；
- 采用电平触发中断；
- 中断控制器可在时钟关闭的情况下接收中断并向 CPU 发出中断请求；

10.3 结构框图

中断控制器的结构框图如图 10-1 所示，31 个中断源进入中断控制器之后，首先与软件配置的强制中断信号分别进行汇总，并根据中断控制器的配置进行中断使能，使能处理之后的中断信号进一步根据配置进行中断屏蔽，最后产生的中断请求信号发送给 CPU 进行处理，同时也发送给时钟管理模块用于时钟唤醒。

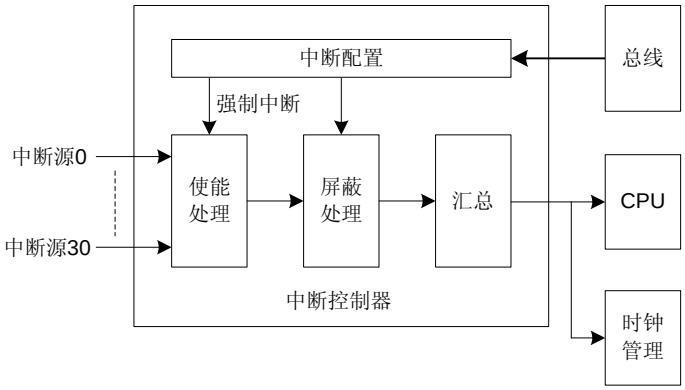


图 10-1 中断控制器结构框图

10.4 功能说明

10.4.1 中断源

GSC3281 中断控制器一共有 31 个中断来源，具体如下表 10-1 所示。在 31 个中断源中，绝大多数中断由 GSC3281 片内设备产生，对应的每个中断源在设备内部又包含若干个子中断源，由中断处理程序根据需要进行进一步判断与处理。片外设备可通过 GPIO 引脚向 GSC3281 发起 GPIO 中断请求，GSC3281 一共有 32 个 GPIO 引脚可以产生 GPIO 中断，由中断处理程序判断具体是哪一个 GPIO 引脚产生了中断请求。

GSC3281 中断控制器的 30 号中断源是一个特殊的中断源，该中断用于表明发生了总线错误中断。当 CPU 访问非法地址（即实际不存在的地址）时，GSC3281 芯片内部总线将发起一个总线错误中断，CPU 接收到该中断之后可进行必要的处理。GSC3281 的地址空间分配情况如表 2-1 所示，所有在表 10-1 中未进行分配的地址空间均属于非法地址，对非法地址的访问将引起一个总线错误中断。另一方面，分配给某个设备的地址空间一般情况下不会被全部使用，例如中断控制器实际只使用了 8KB 空间中的几百个字节；如果 CPU 访问了已分配给某个设备但实际没有使用的空间，则根据具体设备的不同，有可能会发生总线错误中断，也有可能等效于一个空操作（写无效，读出为 0）。

表 10-1 GSC3281 中断控制器中断源列表

中断源编号	中断名称	中断描述
0	IRQ_DMA	DMA 控制器中断
1	IRQ_NFC	NAND Flash 控制器中断
2	Reserved	保留
3	IRQ_MAC	以太网 MAC 控制器中断
4	Reserved	保留
5	IRQ_USB	USB2.0 OTG 控制器中断
6	IRQ_SPI0	SPI0 控制器中断
7	IRQ_SPI1	SPI1 控制器中断
8	IRQ_SCI0	7816-0 接口中断
9	IRQ_SCI1	7816-1 接口中断
10	IRQ_PS2_0	PS2-0 接口控制器中断
11	IRQ_PS2_1	PS2-1 接口控制器中断
12	IRQ_UART0	UART0 中断
13	IRQ_UART1	UART1 中断
14	IRQ_UART2	UART2 中断
15	IRQ_UART3	UART3 中断
16	IRQ_UART4	UART4 中断
17	IRQ_UART5	UART5 中断
18	IRQ_UART6	UART6 中断
19	IRQ_UART7	UART7 中断
20	IRQ_I2S	I2S 接口控制器中断
21	IRQ_I2C	I2C 接口控制器中断
22	Reserved	保留

23	IRQ_KEYPAD	矩阵键盘中断
24	IRQ_KEYPAD_WAKE	矩阵键盘唤醒中断；在 CPU 处于休眠状态、keypad 时钟也关闭的情况下，任意 keypad 按键操作均可触发该中断用于唤醒 CPU
25	IRQ_PWM	旋转编码器与 PWM 中断
26	IRQ_TIMER	可编程定时器中断
27	IRQ_WATCHDOG	看门狗定时器中断
28	IRQ_GPIO	可编程输入输出接口中断
29	Reserved	保留
30	IRQ_BUSERR	总线错误中断，当 CPU 访问未定义的非法地址时，GSC3281 内部总线将发起一个总线错误中断通知 CPU

10.4.2 中断产生

从一个中断源向中断控制器发出中断请求，到中断控制器向 CPU 发出中断请求，中间经过了合并强制中断、中断使能、中断屏蔽以及中断汇总等操作，图 10-2 表示了中断请求的产生过程，其中 int_force_0 表示软件配置中断控制器中的寄存器位强制使 0 号中断源发生中断，最后输出的 irq 为中断控制器向 CPU 发出的中断请求，中断请求的状态将保存到 CPU 中 CP0 寄存器 Cause 的第 11 位（即 Cause[10]）。

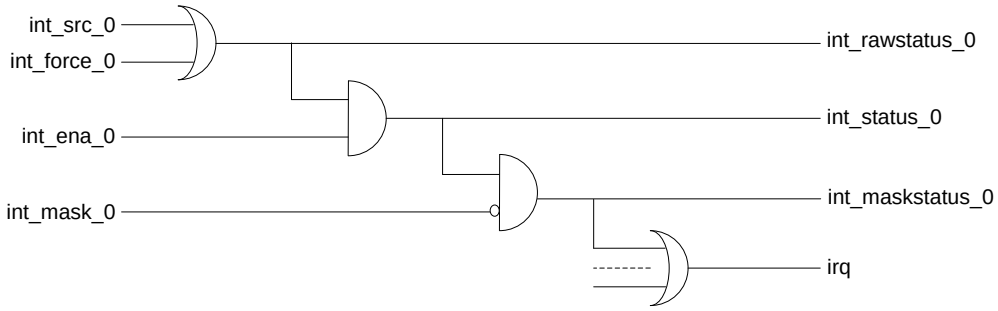


图 10-2 中断产生过程

10.4.3 强制中断

GSC3281 中断控制器提供了一种强制产生中断的方法，通过对 ICTL_INTFORCE 寄存器中的某一位写 1，可以强制让对应的中断号发生中断，效果上等同于对应的中断源发生了中断。

10.4.4 中断状态

GSC3281 中断控制器向软件提供了中断产生过程中不同阶段的若干中断状态，如图 10-2 所示，包括有 rawstatus、status 与 maskstatus 三种中断状态，软件可以通过读取对应的中断状态寄存器来获取这三种状态。rawstatus 表示从设备发出的未经任何使能与屏蔽处理的原始中断状态，任何中断源发生了中断，则对应的 rawstatus 必定为 1。status 表示经过中断使能之后的中断状态，所有未经使能的中断将不能反应到该状态中；如果某个中断源发生了中断，但该中断没有被使能，则 rawstatus 相应位为 1，但 status 相应位将为 0。maskstatus 表示已经使能并且未被屏蔽的中断状态，即真正有效的中断状态，只有 maskstatus 状态有效的

中断才可以向 CPU 发出中断；如果某个中断的 status 状态为 1，但该中断设置为被屏蔽，则 maskstatus 将会为 0，该中断不能向 CPU 发出中断请求。通过提供三种不同的中断状态，可以方便中断处理程序灵活进行中断处理。

10.4.5 中断初始化

在使用中断控制器之前，软件应该对中断控制器进行初始化，一个正常的初始化流程包含如下几个步骤：

1. 禁止（即将中断使能寄存器 ICTL_INTEN 设置为 0）所有中断；由于芯片复位之后 ICTL_INTEN 寄存器默认状态为 0，因此这一步骤也可以省略；
2. 初始化可以产生中断的设备；
3. 根据需要配置中断屏蔽寄存器 ICTL_INTMASK；
4. 配置中断使能寄存器 ICTL_INTEN 相应位为 1 以使能中断；

10.4.6 中断处理

当中断请求被 CPU 接受时，CPU 将从通用的入口地址进入中断处理程序，在中断处理程序中，CPU 读取 ICTL_MASKSTASTUS 寄存器的 maskstatus 中断状态判断有哪些设备发生了中断，并按照一定的优先级进行中断处理。尽管 rawstatus 状态有效的中断与 status 状态有效的中断并不会直接产生中断，但中断处理程序可根据需要读取这个两个状态并进行进一步的处理。

GSC3281 芯片中断控制器采用了高电平有效的电平触发方式，ICTL_RAWSTATUS、ICTL_STATUS 与 ICTL_MASKSTATUS 三个中断状态寄存器中的中断状态位均为只读模式；当发生中断时，中断状态位变为 1，在中断服务程序对中断源进行处理消除了产生中断的条件之后，中断状态位将自动变为 0，不需要中断处理程序显式对中断控制器的中断状态进行清零。在对一个中断进行处理的时候，为了避免同一个中断反复触发中断，可以将该中断暂时屏蔽，并在中断处理完成之后取消对该中断的屏蔽。

10.5 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

10.6 编程指导

当前版本用户手册暂不提供详细编程指导。

11 外部静态存储器接口

11.1 概述

GSC3281 芯片包含一个外部静态存储器接口（External Memory Interface，缩写为 EMI），具有 8bit 数据线和 9bit 地址线，支持与 SRAM 或 NOR Flash 工作时序类似的设备，例如小尺寸 LCD 显示屏等。GSC3281 芯片通过 EMI 接口连接外部设备，同时支持三个设备；配置好 EMI 接口的时序，即可通过 EMI 接口对外部设备进行读写等操作。

11.2 引脚描述

表 11-1 EMI 接口引脚描述

名称	类型	上拉 下拉	功能描述
emi_csn0	O	-	EMI 片选第 0 位，低有效
emi_csn1	O	-	EMI 片选第 1 位，低有效
emi_csn2	O	-	EMI 片选第 2 位，低有效
emi_oen	O	-	EMI 读使能，低有效
emi_wen	O	-	EMI 写使能，低有效。
emi_rdy	I	U	外部 NOR Flash Ready 信号
emi_d0	B	-	EMI 数据第 0 位
emi_d1	B	-	EMI 数据第 1 位
emi_d2	B	-	EMI 数据第 2 位
emi_d3	B	-	EMI 数据第 3 位
emi_d4	B	-	EMI 数据第 4 位
emi_d5	B	-	EMI 数据第 5 位
emi_d6	B	-	EMI 数据第 6 位
emi_d7	B	-	EMI 数据第 7 位
emi_a0	O	-	EMI 地址第 0 位
emi_a1	O	-	EMI 地址第 1 位
emi_a2	O	-	EMI 地址第 2 位
emi_a3	O	-	EMI 地址第 3 位
emi_a4	O	-	EMI 地址第 4 位
emi_a5	O	-	EMI 地址第 5 位
emi_a6	O	-	EMI 地址第 6 位
emi_a7	O	-	EMI 地址第 7 位
emi_a8	O	-	EMI 地址第 8 位

11.3 功能框图

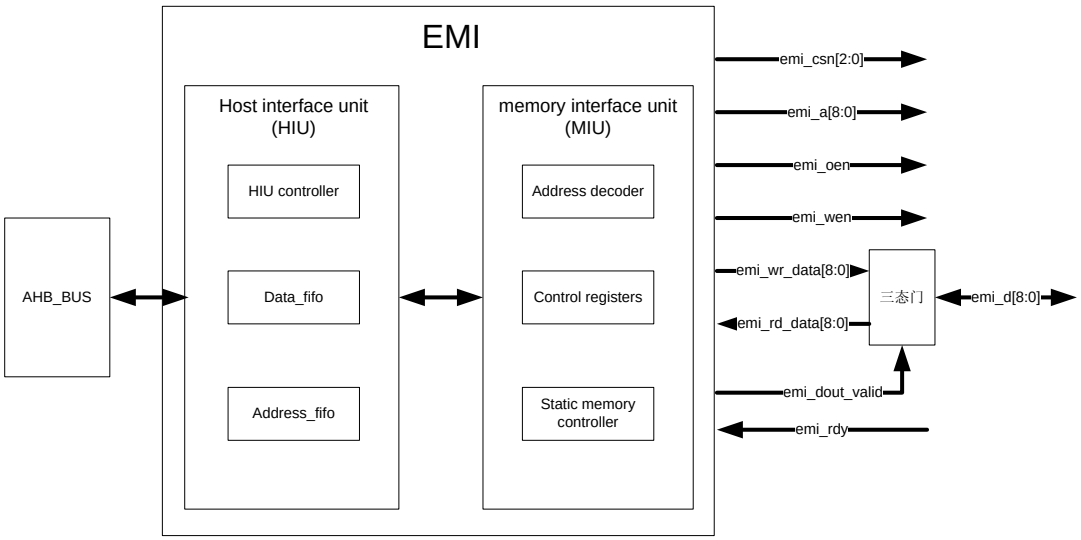


图 11-1 EMI 功能框图

emi_wr_data[8:0]信号为 EMI 控制器的写数据信号。

emi_rd_data[8:0]信号为 EMI 控制器的读数据信号。

emi_dout_valid 信号为 EMI 控制器的输出使能信号。

emi_wr_data、emi_rd_data、emi_dout_valid 通过一个三态门与外部设备的数据输入输出信号 emi_d 连接。

11.4 功能说明

GSC3281 芯片 EMI 接口有三个片选信号，数据总线宽度 8bit，地址总线宽度 9bit，支持 3 个与 NOR Flash 或 SRAM 工作时序类似的设备，可单独配置每个设备的读写时序，也可配置每个片选对应的地址。

11.4.1 配置片选对应地址

EMI 有 3 个片选信号，默认情况下第一个片选 emi_csn[0]对应物理地址为 0x1DC00000~0x1DC001FF，第二个片选 emi_csn[1]对应物理地址为 0x1DE00000~0x1DE001FF，第三个片选 emi_csn[2]对应物理地址 0x1DF00000~0x1DF001FF。

这三个片选信号对应的基地址可以通过 EMI 的寄存器 EMI_CSSLR0_LOW、EMI_CSSLR1_LOW、EMI_CSSLR2_LOW 来重新设置，重新设置的物理地址范围在 0x1DC00000~0x1DFFFFFF。注意向 EMI 控制器的寄存器中设置的地址均为物理地址，下同。

11.4.2 配置外部存储器读写时序参数

由于不同的外部设备的读写时序不同，因此需要对 EMI 配置相应的读写时序以适应不同的外部设备。每一个片选信号对应一个配置读写时序的寄存器 EMI_SMTMGR_SET0、EMI_SMTMGR_SET1 或 EMI_SMTMGR_SET2。因此，EMI 可同时支持三个读写时序不同的外

部设备。

11.4.2.1 异步 SRAM

如果 EMI 连接的设备的工作时序类似异步 SRAM，则需要配置的时序参数包括：读周期时间 t_rc、写地址建立时间 t_as、写操作宽度 t_wp、写地址/数据保持时间 t_wr、间隔宽度 t_bta。

11.4.2.2 异步 NOR Flash

如果 EMI 连接的设备的工作时序类似异步 NOR Flash，则需要配置的时序参数包括：读周期时间 t_rc、页模式读周期时间 t_prc、写地址建立时间 t_as、写操作宽度 t_wp、写地址/数据保持时间 t_wr、间隔宽度 t_bta。

11.4.2.3 读时序

SMTMGR_SET0、SMTMGR_SET1 和 SMTMGR_SET2 的 bit0~bit5 为参数 t_rc，表示读周期时间。如图 11-2 所示，t_rc 值为 2 表示 3 个 hclk 周期。t_rc 的具体时间可由 hclk 的频率计算得出。

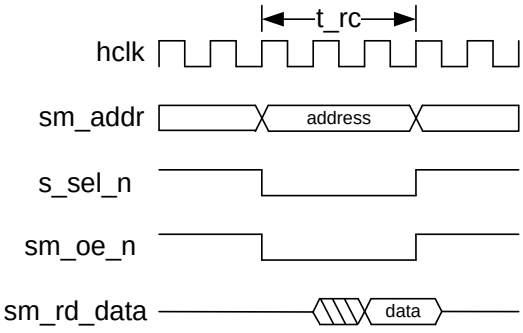


图 11-2 读周期时序

SMTMGR_SET0、SMTMGR_SET1 和 SMTMGR_SET2 的 bit23 为参数 page_mode，表示外部存储器设备是否支持页模式；bit19~bit22 为参数 t_prc，表示页模式读周期时间。如图 11-3 所示，page_mode 为 1 表示使用 pagemode 功能，t_prc 值为 0 表示 1 个 hclk 周期，t_rc 为 1 表示 2 个 hclk 周期。t_prc、t_rc 的具体时间可由 hclk 的频率计算得出。

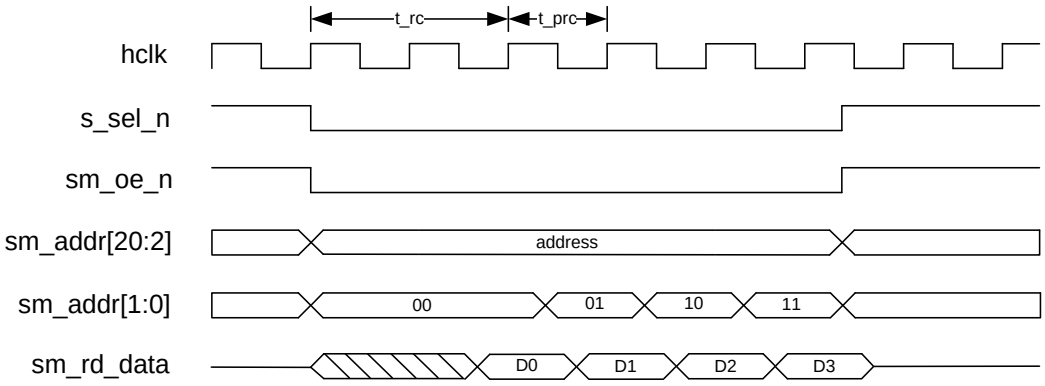


图 11-3 页模式读周期时序

11.4.2.4 写时序

SMTMGR_SET0、SMTMGR_SET1 和 SMTMGR_SET2 的 bit6~bit7 为参数 t_as，表示地址建立时间。如图 11-4 所示，t_as 为 1 表示 1 个 hclk 周期。t_as 的具体时间可由 hclk 的频率计算得出。

SMTMGR_SET0、SMTMGR_SET1 和 SMTMGR_SET2 的 bit10~bit15 为参数 t_wp，表示写操作宽度。如图 11-4 所示，t_wp 为 1 表示 2 个 hclk 周期。t_wp 的具体时间可由 hclk 的频率计算得出。

SMTMGR_SET0、SMTMGR_SET1 和 SMTMGR_SET2 的 bit8~bit9 为参数 t_wr，表示数据保持时间。如图 11-4 所示，t_wr 为 1 表示 1 个 hclk 周期。t_wr 的具体时间可由 hclk 的频率计算得出。

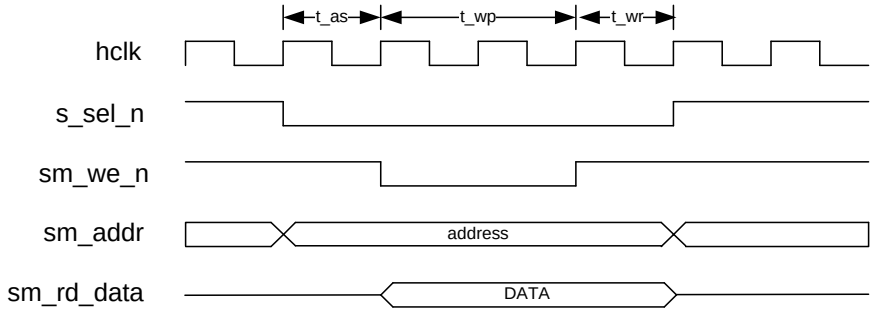


图 11-4 写周期时序

SMTMGR_SET0、SMTMGR_SET1 和 SMTMGR_SET2 的 bit16~bit18 为参数 t_bta，表示读写间隔宽度。如图 11-5 所示，t_bta 为 1 表示 1 个 hclk 周期。t_bta 的具体时间可由 hclk 的频率计算得出。

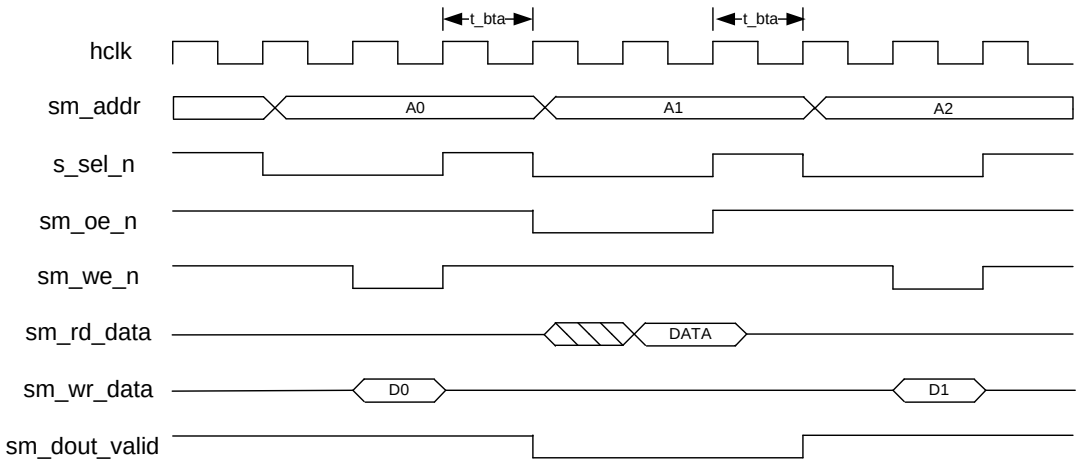


图 11-5 读写间隔时序

11.5 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

11.6 编程指导

当前版本用户手册暂不提供详细编程指导。

12 SPI 控制器

12.1 概述

GSC3281 系统中有两个 SPI 接口，接在 APB 和 AHB 上，分别叫 SPI0，SPI1。本文余下部分除有特殊说明外，SPI0 指的是接在 APB 总线上的 SPI 接口，SPI1 指的是接在 AHB 上的 SPI 接口。SPI 总线是一种高速的、全双工、同步的通信总线，并且在芯片的管脚上最多只占用四根线。

两个 SPI 接口，都符合 SPI 协议，用户在使用上，只要是符合标准 SPI 协议的器件，都可以任意选择用 SPI0 或 SPI1。GSC3281 片内已经通过 SPI0 接口集成了 ADC 控制器，默认状态下，ADC 控制器是不使能的，需要使用 ADC 控制器时，可以通过编程系统控制模块的 ADC 配置寄存器来使能 ADC 控制器，不管是否使能 ADC 控制器，用户都可以使用 SPI0 外接其它 SPI 器件，ADC 的片选通过编程 SPI0 的片选寄存器 SPI_CSN（物理地址是 0x1C101050）来实现，外部接的其它 SPI 器件通过 GPIO 来实现片选。SPI1 支持系统从 SPI Flash 启动，当系统从 SPI Flash 启动时，SPI Flash 的片选信号接 GSC3281 的 SPI1_CSN 管脚。用户使用 SPI1 时，既可以通过编程 SPI1 的片选寄存器 SPI_CSN（物理地址是 0x1C04C050）来控制 SPI1_CSN 管脚实现片选，也可以通过 GPIO 来实现片选。

12.2 引脚描述

表 12-1 SPI 控制器引脚描述

名称	类型	上拉 下拉	功能描述
SPI 控制器 0 接口			
spi0_miso	I	U	SPI0 主设备数据输入信号
spi0_mosi	O		SPI0 主设备数据输出信号
spi0_sck	O	U	SPI0 器件工作时钟信号，主设备输出
SPI 控制器 1 接口			
spi1_miso	I		SPI1 主设备数据输入信号
spi1_mosi	O		SPI1 主设备数据输出信号
spi1_sck	O		SPI1 器件工作时钟信号，主设备输出
spi1_csn	O		SPI1 片选，低有效。系统不从 SPI Flash 启动时，SPI1 片选也可以用 GPIO 实现

表 12-1 中的 spi0_miso 和 spi0_sck 两根信号上拉是由于在 GSC3281 中，这两个信号分别与 keypad 的 col1 和 col0 复用，而 keypad 的 col1，col0 需要上拉，所以这两根信号在片内上拉，此处上拉无关 SPI 控制器的接口特性，也不会影响 SPI 的接口功能。

12.3 功能特性

GSC3281 中的两个 SPI 接口，均完全符合标准的 SPI 协议，支持全部四种 SPI 操作模式，

可以方便地与符合标准 SPI 协议的从设备进行通信。SPI 控制器支持查询、中断及 DMA 传输模式，支持 256 种波特率传输，支持字节空闲模式，数据帧长度从 2-17 位可配置，MSB 及 LSB 都支持。

图 12-1 是 SPI0（APB 接口）的框图，SPI1 的结构类似。

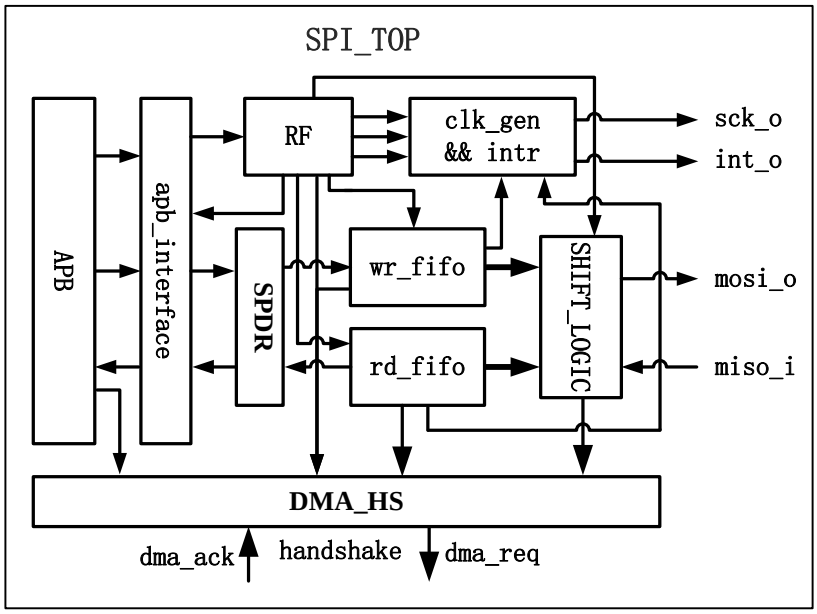


图 12-1 SPI 框图

12.3.1 SPI 传输模式

GSC3281 中两个 SPI 接口均支持 4 种 SPI 传输模式，如图 12-2 所示，CPOL 表示 SPI 时钟的极性，当 CPOL=0 时，时钟 SPICLK 在空闲状态时为低电平，当 CPOL=1 时，时钟 SPICLK 在空闲状态时为高电平。CPHA 表示 SPI 时钟的相位，当 CPHA=0 时，在时钟 SPICLK 的第一个跳变沿（上升或者下降）数据被采样，当 CPHA=1 时，在时钟 SPICLK 的第二个跳变沿（上升或下降）数据被采样。主控制器的传输模式必须和从设备的传输模式设为一致，一般 SPI 器件的传输模式在其手册里都会明确给出的，如 SPI Flash 大部分都是以模式 1 传输或模式 3 进行传输。

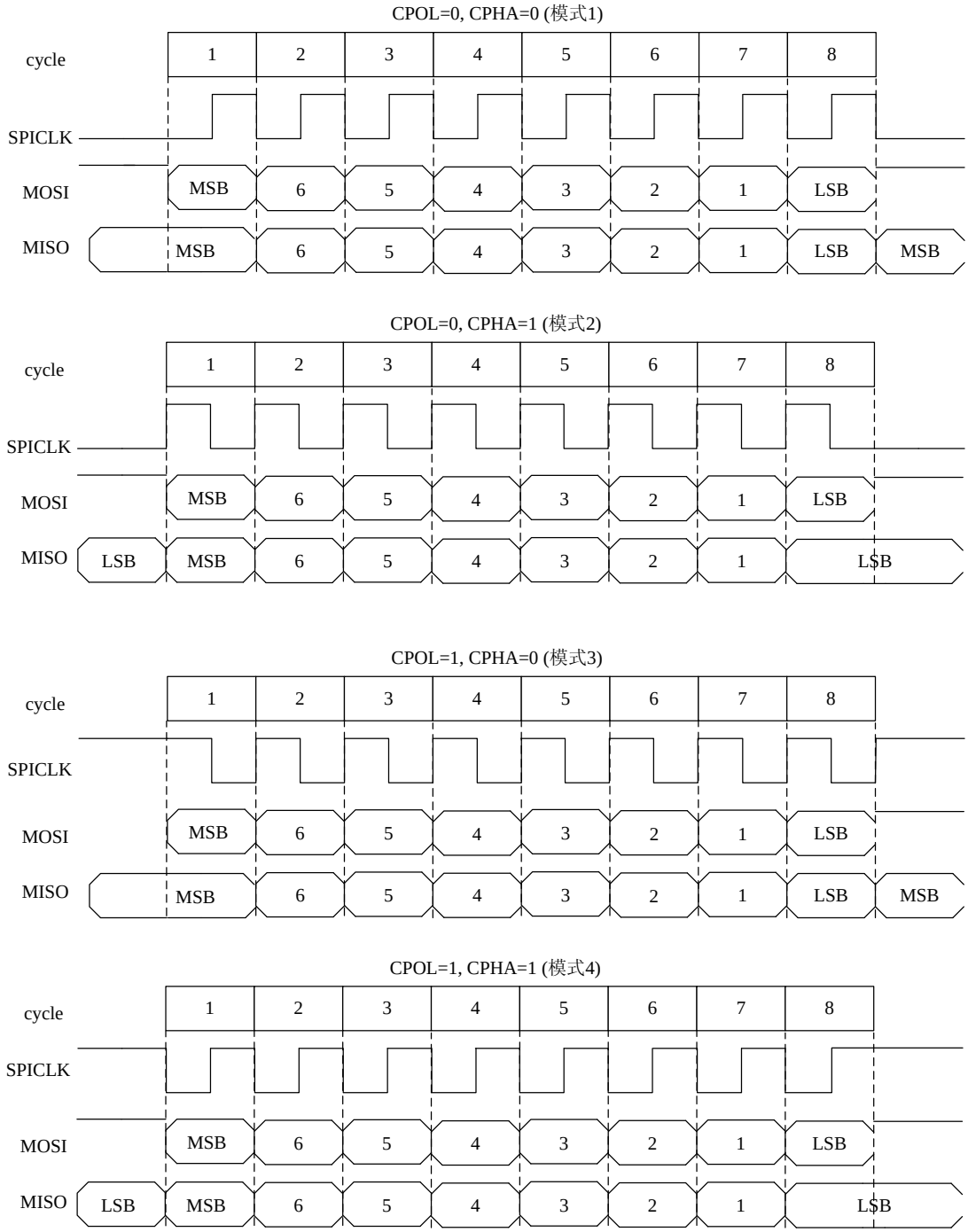


图 12-2 SPI 传输模式

12.3.2 SPI 时钟

在 GSC3281 系统中，由系统控制模块中的时钟管理电路给 SPI0 与 SPI1 模块产生输入时钟，其中 SPI0 控制器的输入时钟 SPI_CLK 由 APB 总线时钟 pclk 整数分频得到，SPI1 控制器的输入时钟由 AHB 总线时钟 hclk 整数分频得到，软件可分别对 SYSCTL_CLKDIV_SPI0 与 SYSCTL_CLKDIV_SPI1 寄存器编程配置分频系数。SPI 模块的输出时钟，即输出给 SPI 器件的时钟是对 SPI 输入时钟分频得到的，软件通过配置寄存器 SPI_CPSR（偏移地址 0x4）来得到输出时钟，输出时钟和输入时钟的关系为 SPICLK_OUT=SPI_CLK/(2*(CPSR+1))，从公式中看出，

输出时钟是输入时钟的偶数分频。

12.3.3 SPI 中断

SPI 控制器有接收 FIFO 满中断（rxf_intr）、接收 FIFO 上溢出中断（rxo_intr）、接收 FIFO 下溢出中断（rxu_intr）、发送 FIFO 上溢出中断（txo_intr）以及发送 FIFO 空中断（txe_intr）几种中断类型，并且每个中断都有相应的屏蔽位，详细参考寄存器说明。

接收 FIFO 满中断（rxf_intr）：当接收 FIFO 里接收到的数据个数等于或者大于接收 FIFO 阈值时，就会产生接收 FIFO 满中断，这里满是根据接收 FIFO 阈值来判断的。当接收 FIFO 里的数据个数少于接收 FIFO 阈值时，这个中断将自动清除。

接收 FIFO 上溢出中断（rxo_intr）：当接收 FIFO 已经完全满了，接收逻辑还往接收 FIFO 里放数据时，将产生接收 FIFO 上溢出中断。软件往中断状态寄存器 SPI_ISR（偏移地址 0x20）的 rxois 位写 1，将清除此中断。

接收 FIFO 下溢出中断（rxu_intr）：当接收 FIFO 已经空了，CPU 再去接收 FIFO 里读数据时，将产生接收 FIFO 下溢出中断。此时的 FIFO 空，是真正意义上的空，表示 FIFO 里没有数据了。软件往中断状态寄存器 SPI_ISR（偏移地址 0x20）的 rxuis 位写 1，将清除此中断。

发送 FIFO 上溢出中断（txo_intr）：当发送 FIFO 里待发送的数据已经完全填满发送 FIFO 了，CPU 再向发送 FIFO 里写数据，将产生发送 FIFO 上溢出中断。此时 FIFO 满是真正意义上满，跟发送 FIFO 阈值没关系。软件往中断状态寄存器 SPI_ISR（偏移地址 0x20）的 txois 位写 1，将清除此中断。

发送 FIFO 空中断（txe_intr）：当发送 FIFO 里待发送的数据的个数等于或者小于发送 FIFO 阈值时，就会产生发送 FIFO 空中断，这里的空是根据发送 FIFO 阈值来判断的。当 CPU 往发送 FIFO 里写数据，让发送 FIFO 里的数据个数大于发送 FIFO 阈值时，此中断自动清除。

以上接收或者发送 FIFO 阈值可以通过软件配置，详细配置参考寄存器说明。

12.3.4 SPI 控制器编程流程

下面是简单的 SPI 控制器编程流程：

1. 配置控制寄存器，主要配置时钟极性和时钟相位以及帧格式；
2. 配置时钟分频寄存器，确定输出的 SPI 时钟；
3. 配置 FIFO 阈值寄存器；
4. 配置中断使能以及中断屏蔽寄存器；
5. 配置控制寄存器中的控制器使能位（spe）；
6. 然后开始对数据寄存器操作进行数据传输；
7. 如果软件采用查询的方式进行操作，则必须查询 SPI_SR 寄存器确定 SPI 的工作状态；
8. 如果软件响应中断，软件必须查询 SPI_ISR 寄存器来检查是哪类中断，然后再跳转到相应的中断请求服务程序；

以上简单的 SPI 操作流程，包含查询及中断模式操作，详细的编程指导可以参考 SPI 的编程指导

12.3.5 SPI 的 DMA 操作

SPI 模块除了利用中断和查询的方式进行数据传输外，还可以利用 DMA 进行数据传输。

我们知道 DMA 可以不通过 CPU 的干预进行数据传输，这样大大提高了数据的传输效率。

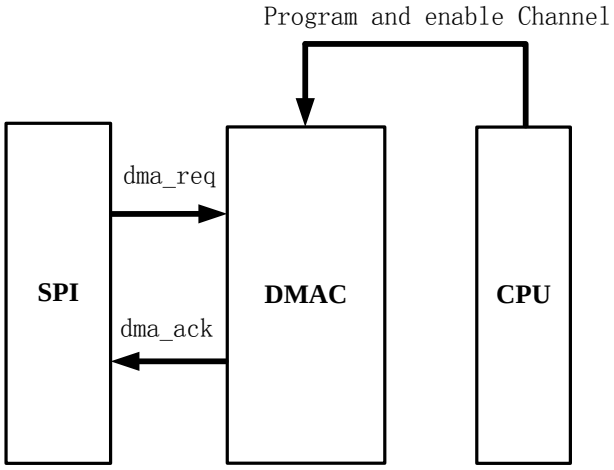


图 12-3 SPI DMA 接口示意图

通过 DMA 来进行 SPI 传输时，除了配置 SPI 控制器外，还需要正确配置 DMA 控制器。下面以 SPI 发送为例，简单描述一下 SPI 的 DMA 操作。

1. 配置 SPI 寄存器，使能 dma_wr_en 位；
2. 配置 DMA 控制器；
3. SPI 请求 DMA 发送；
4. DMA 控制器按配置发送数据；
5. DMA 发送结束，DMA 控制发 dma_ack 给 SPI 控制器；
6. SPI 控制器收到 dma_ack，撤销 DMA 请求 dma_req；

由于 SPI 是全双工的，在 DMA 通过 SPI 发送数据的同时，SPI 的接收线上也在同时接收数据，基于这个原理，SPI 通过 DMA 的读数据时，可以使能两个 DMA 通道，一个通道负责发送数据（任意数据即可），另一个通道负责接收数据。

SPI0 的 FIFO 深度为 8，SPI1 的 FIFO 深度为 16，FIFO 的发送及接收阈值可以设置，在 DMA 传输时，需要正确配置 DMA 控制器的 MSIZE 和 SPI 的 FIFO 发送和接收阈值。具体可以参考 DMA 控制器的用户手册。

12.3.6 SPI Flash 启动

AHB 总线上的 SPI1 支持从 SPI Flash 启动，GSC3281 有两种启动模式，当选从 SPI Flash 启动时，用户只要在片外直接接 SPI Flash，不需要任何配置即可实现系统从 SPI Flash 启动。支持常见厂商如 ATMEL、Winbond、ST 等厂家的 SPI Flash。

系统启动时，CPU 从地址 0xBFC00000 处开始取指，系统分配给启动的地址范围是 0xBFC00000-0xBFF00000，一共 4MB 大小，只要启动代码不要超过这个范围就可以。启动过后，SPI Flash 的访问不受这个地址范围的限制，也就是说在使用中，根据实际需要可以选用合适大小容量的 SPI Flash，只是启动代码的地址范围不要超过上述范围就可以。

12.3.7 SPI 片选

SPI0 在 SOC 内部接有 SPI 接口的 ADC 控制器，ADC 控制器的片选通过编程 SPI0 的片选寄存器 SPI_CSN（物理地址为 0x1C101050）寄存器来实现。当 SPI0 在片外接 SPI 器件时，片选只能通过 GPIO 实现，GPIO 的使用可以参考 GPIO 用户手册。

SPI1 支持从 SPI Flash 启动，GSC3281 有一根管脚 SPI1_CSN 用作启动时 SPI Flash 的片选，用户直接将 SPI1_CSN 管脚接 SPI Flash 的片选管脚即可，在启动代码里，软件不用编程 SPI1 的片选寄存器 SPI_CSN 来控制 SPI1_CSN 管脚。系统启动后，由于在启动时，Flash 的片选已经接在 GSC3281 的 SPI1_CSN 管脚上，所以在启动过后访问 SPI Flash 时，片选也只能通过编程 SPI_CSN 寄存器（物理地址 0x1C040050）来控制 SPI1_CSN 管脚。系统不从 SPI Flash 启动时，SPI1 的片选可以通过 GPIO 实现，也可以通过编程 SPI_CSN 寄存器（物理地址 0x1C040050）控制 SPI1_CSN 管脚来实现。如果 SPI1 外接多个 SPI 器件时，只有一个器件可以使用 SPI1_CSN，其它器件只能通过 GPIO 实现片选。

SPI 的片选必须在 SPI 控制器使能之后才能选中 SPI 器件，也就是 SPI_CTRL（偏移地址为 0x0）寄存器的 spe 位置 1 之后，软件才能编程选中 SPI 器件。

12.3.8 SPI0 的 ADC 采样操作

SPI0 在内部接有一个 SPI 接口的 ADC 控制器，CPU 可以通过 SPI0 接口操作 ADC 控制器。通过 SPI0 控制 ADC 控制器时，需要配置系统控制单元中的 ADC 使能寄存器，然后才能正常使用 ADC 控制器。ADC 的操作可以参考 ADC 的用户手册。

SPI0 控制器还支持自动 ADC 采样，自动采样时，需要配合 DMA 控制器使用。ADC 采样可以是 CPU 通过软件发命令方式进行采样，操作流程可参考 ADC 控制器的用户手册，这期间需要 CPU 参与，并且由 CPU 查询判断是否采到有效数据。为了节省 CPU 开销，SPI0 控制器在硬件上支持自动采样，不需要 CPU 参与 ADC 的初始化、发测量命令及有效数据的判断，这些过程由硬件完成，每次采到的数据放到 SPI0 的接收 FIFO 里面，等达到一定阈值，由 SPI0 控制器向 DMA 控制器发出 DMA 读请求，数据通过 DMA 控制器传到用户想要的位置。

使用时，首先 CPU 配置好 SPI0 控制器，和正常使用 SPI0 控制器的配置类似，只是需要使能 SPI0 控制器的 DMA 读，将数据宽度配置成 16 位；然后配置好 DMA 控制器。ADC 自动采样时通过 SPI0 控制器来发起 DMA 请求，所以 DMA 控制器的外设硬件接口号用 SPI0 控制器接收方向的硬件接口号。设置好哪些通道需要采样，如果几个通道同时使能，将按 0-1-2-3-0-1-2-3-的循环次序进行采样，ADC 的有效数据是 12 位，硬件自动将高 4 位用作通道号的标志。如果是 CPU 启动采样，需要设置好一共需要采样的数据个数，等采样计数到配置的个数就停止。还有一种是由定时器 0 启动采样，软件配置好定时器 0，当定时器 0 计数到配置数值时，定时器 0 将触发 ADC 的自动采样逻辑来进行 ADC 采样。定时器 0 启动采样时，不需要设置采样数据个数，将无限制采样下去，直到 CPU 往启动寄存器 SPI_ADC_TRIGGER 写 0 来结束采样。

普通 SPI 的 DMA 读操作，需要使能两个 DMA 通道，一个负责往 SPI 总线上发数据，一个通道负责接收数据。在 ADC 自动采样时，不需要启动另外一个 DMA 通道往 SPI 总线上写数据，只需要使能一个 DMA 通道负责接收数据即可。

12.4 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

12.5 编程指导

当前版本用户手册暂不提供详细编程指导。

13 UART 接口

13.1 概述

GSC3281 芯片包含共 8 个串口： UART0、UART1、……、UART7。3 个串口 UART0、UART1、UART2 支持 RS232 物理接口；3 个串口 UART3、UART4、UART5 支持 RS485 接口，兼容 RS232 物理接口且支持 DMA 功能；1 个串口 UART6 为 8 线全功能串口且支持 DMA 功能；1 个串口 UART7 支持 RS232 物理接口并支持红外功能。

下文中用 UARTn 代替 UART0、UART1、……、UART7，n= 0、1、2……7。

13.2 引脚描述

表 13-1 UART 引脚描述

名称	类型	上拉 下拉	功能描述
UART0 接口			
u0_rxd	I	-	UART0 的数据输入接口
u0_txd	O	-	UART0 的数据输出接口
UART1 接口			
u1_rxd	I	-	UART1 的数据输入接口 (RS-232)，由于引脚复用信号 jtdi 上拉而产生上拉
u1_txd	O	-	UART1 的数据输出接口 (RS-232)
UART2 接口			
u2_rxd	I	-	UART2 的数据输入接口 (RS-232)
u2_txd	O	-	UART2 的数据输出接口 (RS-232)
UART3 接口			
u3_rxd_0 u3_rxd_1	I	-	UART3 的数据输入接口 (RS-485) 软件选择其中一组引脚
u3_txd_0 u3_txd_1	O	-	UART3 数据输出接口 (RS-485) 软件选择其中一组引脚
u3_txe_0 u3_txe_1	O	-	UART3 的数据输入输出控制(RS-485) 软件选择其中一组引脚
UART4 接口			
u4_rxd_0 u4_rxd_1	I	-	UART4 的数据输入接口 (RS-485) 软件选择其中一组引脚
u4_txd_0 u4_txd_1	O	-	UART4 的数据输出接口 (RS-485) 软件选择其中一组引脚
u4_txe_0 u4_txe_1	O	-	UART4 的数据输入输出控制(RS-485)。其中 u4_txe_1 由于引脚复用信号 col3 上拉而产生上拉。软件选择其中一组引脚
UART5 接口			

名称	类型	上拉 下拉	功能描述
u5_rxd	I	-	UART5 的数据输入接口 (RS-485)
u5_txd	O	-	UART5 的数据输出接口 (RS-485)
u5_txe_0 u5_txe_1	O	-	UART5 的数据输入输出控制(RS-485) 软件选择其中一组引脚
UART6 接口			
u6_rxd	I	-	UART6 的数据输入接口
u6_txd	O	-	UART6 的数据输出接口
u6_rts_n	O	-	UART6 的发送请求，低有效
u6_dtr_n	O	-	UART6 的数据终端准备完毕，低有效
u6_cts_n	I	-	UART6 的发送可用，低有效
u6_dsr_n	I	-	UART6 的数据准备完毕，低有效
u6_ri	I	-	UART6 的响铃
u6_dcd_n	I	-	UART6 的数据载波检测，低有效
UART7 接口			
u7_rxd	I	-	UART7 的数据输入接口（UART7 的红外输入）
u7_txd	O	-	UART7 的数据输出接口（UART7 的红外输出）

13.3 功能框图

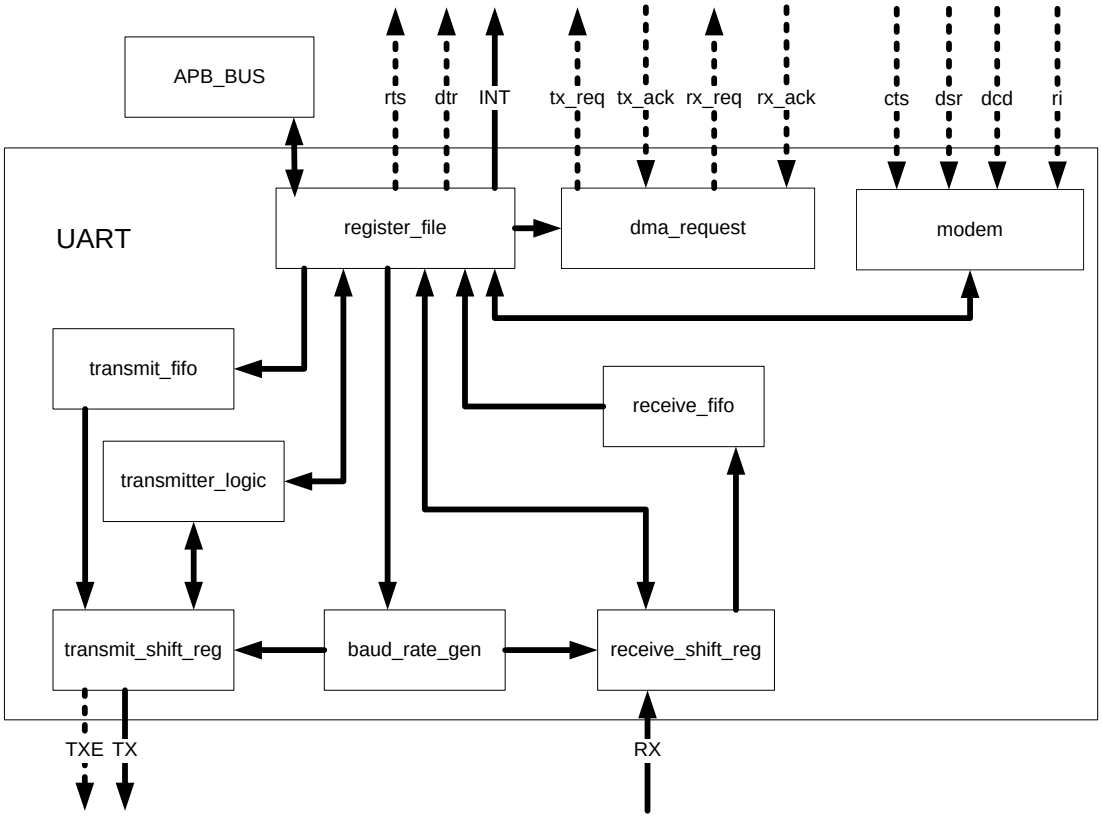


图 13-1 UART 功能框图

其中实线的模块和实线的信号为 8 个串口均包含，dma_requset 模块为 UART3、UART4、UART5、UART6 包含，Modem 模块为 UART6 包含，TXE 信号为 RX485 接口需要的输出使能信号，UART3、UART4、UART5 包含。

13.4 功能说明

UART0、UART1、UART2、UART7 为两线串口，支持 RS232 物理接口；UART3、UART4、UART5 支持 RS485 物理接口；UART6 为 8 线全功能串口。

UART 均符合 16550A 协议，支持 5bit、6bit、7bit 或 8bit 数据位，1bit 或无奇偶校验位，1bit、1.5bit 或 2bit 停止位。

所有的 UART 均可配置波特率时钟，UART0~UART5 为 8 个波特率时钟周期采样 1bit 数据，UART6、UART7 为 16 个波特率时钟周期采样 1bit 数据。

UART 模块使用 0.5 个波特率时钟周期进行再同步，允许波特率误差可达 4%～7%。

每个 UART 均具有发送和接收两个 16 位深 8 位宽 FIFO。

所有的 UART 均具有可编程发送缓冲器空中断功能，当可编程发送缓冲器空中断功能禁止时，UARTn_LSR 寄存器 bit5 为 1 时表示发送 FIFO 空，当可编程发送缓冲器空中断功能使能时，UARTn_LSR 寄存器 bit5 为 1 时表示发送 FIFO 满。

13.4.1 波特率计算

UART0~UART7 的波特率是由各自的工作时钟 uart_clk 按照波特率因子寄存器：UARTn_DLL 和 UARTn_DLH 中的值进行分频得到的。波特率因子的低字节存在 UARTn_DLL 寄存器中，波特率因子的高字节存在 UARTn_DLH 寄存器中。其中 UART0~UART5 波特率的计算方式为：

波特率=工作时钟频率/(8*波特率因子)。

UART6 和 UART7 的波特率计算方式与 UART0~UART5 有些不同，为：

波特率=工作时钟频率/(16*波特率因子)。

需要注意的是，写寄存器 UARTn_DLL 和 UARTn_DLH 时，两个寄存器的写顺序没有要求，但是必须对这两个寄存器都进行写操作。即使波特率因子的高 8 位都是 0，也要向 UARTn_DLH 寄存器写入数据 0，不能省略。

各 UART 工作时钟频率通过系统控制模块 SYSCTL 设置。

13.4.2 时钟产生

每个 UART 均使用两个时钟信号：pclk 和各自的工作时钟 uart_clk，可以在系统控制模块 SYSCTL 中设置。pclk 为 APB 总线时钟信号，uart_clk 为各个 UART 的工作时钟信号，波特率根据这个时钟计算。

UART0、UART1、UART2、UART6、 UART7 各自的 uart_clk 是 pclk 的分频时钟，互相独立，可以为 pclk 的 2 分频、3 分频、4 分频……64 分频。

UART3、UART4、UART5 各自的 uart_clk 是 hclk 的分频时钟，互相独立，可以为 hclk 的 2 分频、3 分频、4 分频……128 分频。通常情况下 uart_clk 比 pclk 频率慢，如果 uart_clk 比

pclk 频率快的话，uart_clk 频率不能超过 pclk 频率的 4 倍。在设置 uart_clk 和 pclk 时，尽量满足 uart_clk 频率与 pclk 频率为整数倍数关系。

13.4.3 中断与中断屏蔽

每个 UART 均包含一个中断信号，支持收到数据状态中断、收到数据中断、超时中断、发送缓冲器空中断等。每个中断都可以通过配置使能或者禁止。其中收到数据中断和超时中断两个中断通过中断屏蔽寄存器的 bit0 位同时使能或者禁止，发送缓冲器空中断通过中断屏蔽寄存器的 bit1 位使能或者禁止，收到数据状态中断通过中断屏蔽寄存器的 bit2 位使能或者禁止。

另外 UART6 支持 Modem 功能，支持 Modem 中断。

13.4.4 可编程发送缓冲器空中断

UART 具有可编程发送缓冲器空中断功能，每个 UART 的发送缓冲器空中断(THRE)均可编程。当可编程发送缓冲器空中断功能禁止时，相应 UART 的传输状态寄存器 UARTn_LSR.bit5 为 1 时表示发送 FIFO 空，而且中断状态寄存器 UARTn_IIR 的发送缓冲器空中断 THRE 位为 1 表示发送缓冲器中没有任何数据。当可编程发送缓冲器空中断功能使能时，传输状态寄存器 UARTn_LSR.bit5 为 1 时表示发送 FIFO 满，而且中断状态寄存器 UARTn_IIR 的发送缓冲器空中断 THRE 位表为 1 示发送缓冲器数据个数低于 FIFO 控制寄存器 UARTn_FCR 中设定的 TFTL 发送 FIFO 空阈值。

UART 可编程发送缓冲器空中断功能通过相应 UART 的中断屏蔽寄存器的 bit7 位使能或者禁止。

UART 把发送 FIFO 中即将发送的数据写入发送缓冲器中发送。UART 把正在接收的数据存入接收缓冲器，接收完毕后存入接收 FIFO 中。

13.4.5 DMA 功能

UART3、UART4、UART5、UART6 具有 DMA 功能。

UART 支持硬件 DMA 握手信号。可设置相应 UART 的 FIFO 控制寄存器 UARTn_FCR 的接收 FIFO 满阈值与发送 FIFO 空阈值，当使用 DMA 功能时， UART 接收缓冲器 UARTn_RBR 中数据个数高于 FIFO 控制寄存器 UARTn_FCR 的接收 FIFO 满阈值，UART 会自动向 DMA 控制器发送读请求信号；当 UART 发送缓冲器 UARTn_THR 中数据个数低于 FIFO 控制寄存器 UARTn_FCR 的发送 FIFO 空阈值，UART 会自动向 DMA 控制器发送写请求信号。接收 FIFO 阈值与发送 FIFO 阈值的设置参见 UARTn_FCR 寄存器说明。

M_size 表示 DMA 控制器设置的每次 DMA 传输的数据个数，接收 FIFO 满阈值与发送 FIFO 空阈值要满足：

发送时 M_size <= FIFO 深度–发送 FIFO 空阈值(FIFO 深度为 16)；

接收时 M_size <=接收 FIFO 满阈值；

13.4.6 自流控功能

UART6 全功能串口具有自流控功能，当 UART6 工作在全功能模式，可在数据传输过程

中自动控制 cts 信号和 rts 信号。cts 信号和 rts 信号均为低有效信号。

使用自流控功能时，UART6 的 cts 信号可与外部的全功能串口的 rts 信号相连，UART6 的 rts 信号可与外部的全功能串口的 cts 信号相连。

要使用自流控功能必须要使能 FIFO，即令 UART6 的 UART6_FCR.bit0 置 1。可通过 UART6 的 UART6_MICR.bit5 置 1 来使能自流控功能。同时 UART6_MICR.bit6 要置 0。

当使能自流控功能后，如果 UART6 的输入信号 cts 为高表示无效时，UART6 的发送功能被禁止，UART6 不能发送数据，这样可避免外部接收 UART 的 FIFO 溢出。外部接收 UART 要想令 UART6 停止发送，需要在 UART6 发送的最后一个数据的停止位中间时刻之前令 UART6 输入信号 cts 为高表示无效。需要注意的是，虽然 UART6 的发送功能被禁止，但是软件仍然能够向 UART6 的发送 FIFO 中写入数据，所以需要注意避免发送 FIFO 溢出。

当使能自流控功能后，可通过 UART6 的 UART6_MICR.bit1 置 1 来使能自流控的 rts 功能。即当 UART6 接收 FIFO 中的数据个数达到接收 FIFO 阈值，UART6 的 rts 输出信号被强制为高表示无效。当 UART6 接收 FIFO 中的数据全部读出时，UART6 的 rts 信号才会重新被置为低表示有效。

13.4.7 IRDA 红外功能

UART7 集成了红外模块，支持 IRDA 1.0 SIR 红外协议，要使用 UART7 的红外功能，需要先通过系统控制模块的 SYSCTL_UART7_CFG 寄存器 bit0 置 1 选择 UART7 的红外输入输出引脚，然后通过 UART7 的 MICR.bit6 位使能红外功能。

当使能红外功能后，UART7 的数据输出信号 TX 被红外数据输出信号 SIR_OUT 取代，UART7 的数据输入信号 RX 被红外数据输入信号 SIR_IN 取代。按照 IRDA 1.0 SIR 红外协议发送和接收数据，红外功能发送和接收数据的脉冲宽度为普通模式下 1bit 数据宽度的 3/16，红外功能发送和接收数据格式如图 13-2 所示：

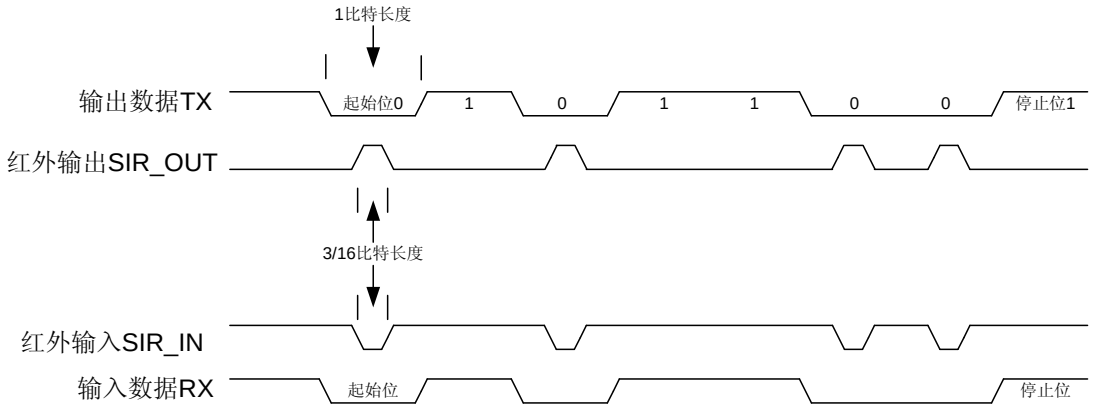


图 13-2 红外发送和接收数据格式

13.4.8 RS485 输出使能信号 TXE

UART3、UART4、UART5 为支持 RS485 物理接口的串口。由于 RS485 为半双工结构，所以需要额外的一个控制信号控制读写状态。因此它们各有一个输出使能信号 TXE 提供给 RS485 物理接口的收发芯片(例如 MAX3485ESA)。这个信号为硬件自动控制，默认情况下为低电平，表示读状态，UART 为接收状态。当用户向 UART3、UART4、UART5 的发送缓冲器中写入数据令 UART 发送数据时，输出使能信号 TXE 自动置为高电平，表示发送状态。

在使用中需要软件来判断 UART 的发送时机，避免 RS485 总线上有数据传输的情况下令 UART 发送数据。

另外，UART3、UART4、UART5 也可以兼容 RS232 物理接口，当 UART3、UART4、UART5 用作 RS232 串口时，TXE 信号不使用，可以浮空。此时 TXE 信号不可作为 GPIO 使用。

输出使能信号 TXE 的时序图如图 13-3 所示：

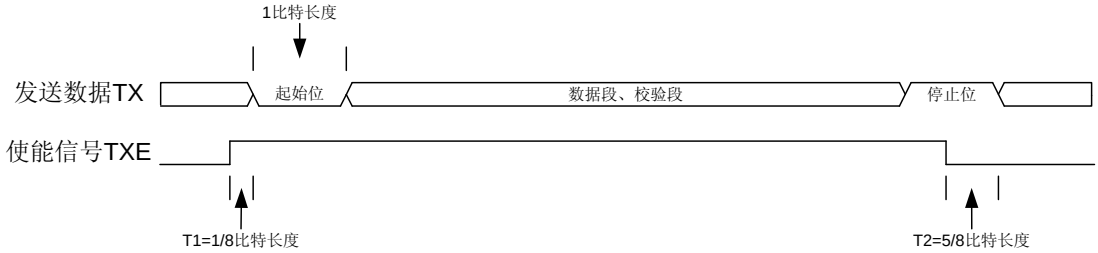


图 13-3 输出使能信号 TXE 时序图

T1 表示 TXE 信号比发送数据 TX 的起始位自动提前 1/8 比特长度时间有效，令收发芯片提前做好准备发送数据；T2 表示 TXE 信号比数据的结尾自动提前 5/8 比特长度时间无效，使接收方有充分的时间进行数据同步。

13.5 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

13.6 编程指导

当前版本用户手册暂不提供详细编程指导。

14 I2C 控制器

14.1 概述

I2C 总线是一个两线的串行接口，包含数据线 SDA 和时钟线 SCL，SCL 和 SDA 线是双向的。GSC3281 作为 I2C 总线的 MASTER，控制总线上的数据通信。支持标准、快速和高速三种速度模式，并向下兼容。如果设为高速模式，则可以与支持标准、快速和高速的 SLAVE 设备进行通信。

14.2 引脚描述

表 14-1 I2C 控制器引脚描述

模块信号	类型	上拉 下拉	描述
i2cscl	B	上拉	I2C 时钟线
i2csda	B	上拉	I2C 数据线

14.3 功能说明

14.3.1 支持协议

支持 2.1 版本 I2C 规范。

14.3.2 功能特点

1. 支持 APB 接口
2. 在 I2C 通信中作为 MASTER 设备
3. 支持三种速度模式：标准(0-100kb/s)，快速(0-400kb/s)，高速(0-3.4Mb/s)
4. 发送 FIFO 与接收 FIFO 的地址深度均为 8
5. 支持 restart 功能
6. 支持滤波功能，即去毛刺功能
7. 支持 7bit 地址格式和 10bit 地址格式
8. 支持 DMA 功能，并支持硬件握手

14.3.3 功能描述

I2C 总线的 MASTER 由 APB 接口，I2C 接口和发送接收 FIFO 等部分组成，具体见图 14-1。

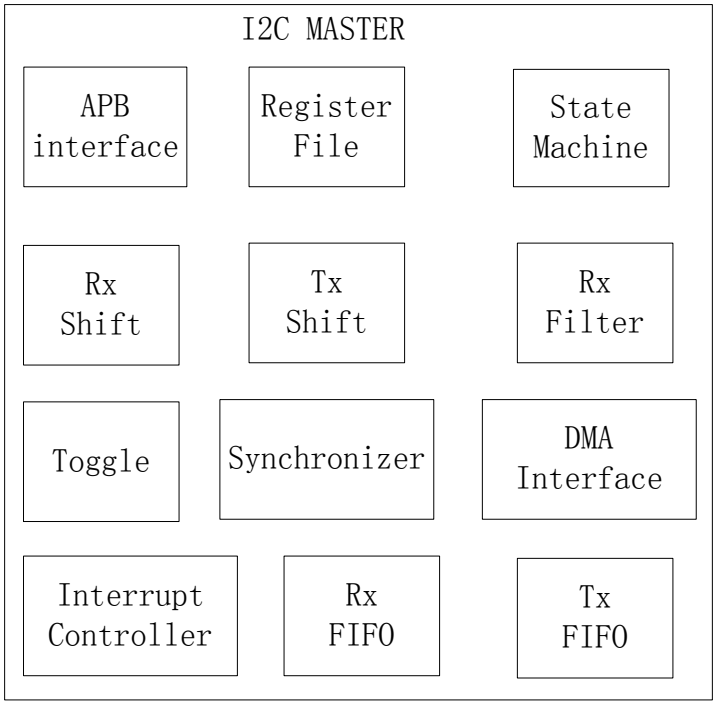


图 14-1 I2C MASTER 结构

APB Interface： 此模块将 APB 数据转换成 I2C MASTER 内部总线数据格式。

Register File： 包含一些配置寄存器，同时也是与软件的接口。

State Machine： 按照 I2C 的协议运作，控制其它模块。

Rx Shift： 将 I2C 总线上接收到的数据比特流转换成字节形式。

Tx Shift： 将发送的数据字节转换成比特流发送到 I2C 总线上。

Rx Filter： 检测 I2C 总线上的事件，例如 start 信号和 stop 信号。

Synchronizer： 处理跨时钟域的信号。

DMA Interface： 产生 DMA 的握手信号。

Interrupt Controller： 产生中断和中断标志位。

Tx FIFO： 需要发送到 I2C 总线的数据以字节形式存入此 FIFO。

Rx FIFO： 将 I2C 总线的数据以字节的形式存入此 FIFO。

14.3.4 I2C 时序

14.3.4.1 开始和停止条件

开始条件产生后，字节数据以串行行式在 SDA 线上进行传输，一个停止条件能结束该数据传输。MASTER 能一直产生开始和停止条件。当一个开始条件产生后，I2C 总线获得繁忙信号。停止条件将使 I2C 总线空闲。

当 MASTER 发起一个开始条件，它将发送一个 SLAVE 地址来通知 SLAVE 设备。一个字节的地址域包含 7 位地址和 1 位传输方向指示位（表示写或读）。如果位 8 是 0，表示写操作（发送操作）；如果位 8 是 1，表示请求读取数据（接收操作）。

MASTER 通过发出一个停止条件来完成传输操作。如果 MASTER 想继续将数据发送到 I2C 总线，它将产生另一个开始条件和一个 SLAVE 地址。通过这种方式，读写操作能在不同的格式下被执行。具体可见图 14-2。

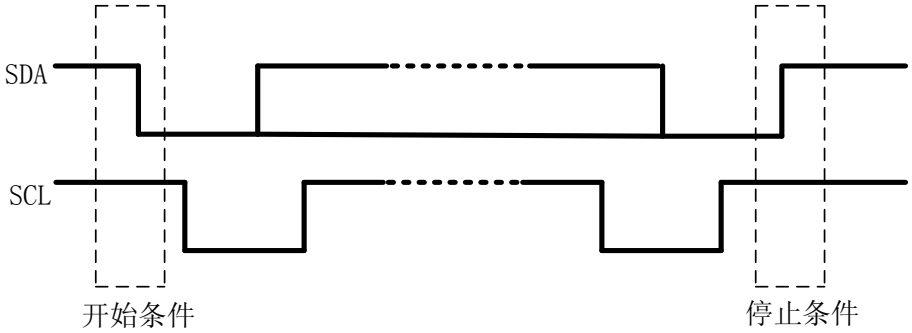


图 14-2 开始和停止条件

14.3.4.2 I2C 总线的数据传输

在 SDA 线上的每一个字节长度必须是 8 位。起始条件后的第一个字节有一个地址域，地址域通过 MASTER 被传输。每一个字节后面跟随一个 ACK（acknowledgement）位。MSB 位始终首先发送。I2C 总线数据传输的模块图，如图 14-3 所示。

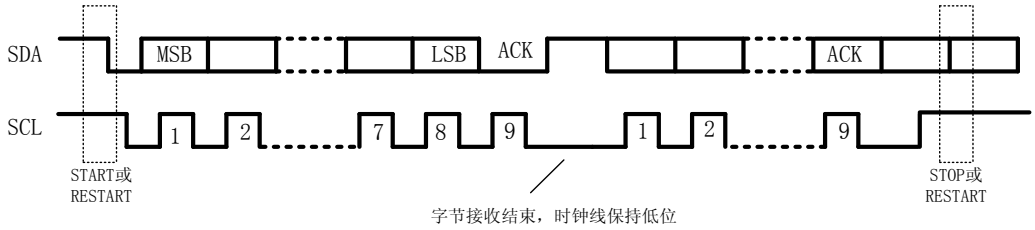


图 14-3 I2C 总线数据传输

14.3.5 I2C_CLK 与 SCL

在 I2C 主控制器中，I2C_CLK 为内部工作时钟。而 I2C 总线上的时钟信号 SCL 则是通过 I2C_CLK 计数分频产生的，即 I2C_CLK 频率是 SCL 频率的整倍数，而这个倍数可以通过配置寄存器 I2C_SS_SCL_HCNT，I2C_SS_SCL_LCNT，I2C_FS_SCL_HCNT，I2C_FS_SCL_LCNT，I2C_HS_SCL_HCNT 和 I2C_HS_SCL_LCNT 来实现，具体可参见相应寄存器的描述部分。在标准速度模式下，通过配置 I2C_SS_SCL_HCNT 和 I2C_SS_SCL_LCNT 两个寄存器来实现 SCL 的占空比和频率大小。同理，在快速速度模式下，通过配置 I2C_FS_SCL_HCNT 和 I2C_FS_SCL_LCNT 两个寄存器来实现 SCL 的占空比和频率大小，在高速速度模式下通过配置 I2C_HS_SCL_HCNT 和 I2C_HS_SCL_LCNT 两个寄存器来实现 SCL 的占空比和频率大小。

14.3.5.1 SCL 相关寄存器的最小值

在标准速度和快速速度模式下，I2C_SS_SCL_LCNT 和 I2C_FS_SCL_LCNT 的值必须大于(I2C_FS_SPKLEN+7)，I2C_SS_SCL_HCNT 和 I2C_FS_SCL_HCNT 的值必须大于(IC_FS_SPKLEN+5)。

在高速速度模式下，I2C_HS_SCL_LCNT 必须大于(I2C_HS_SPKLEN+7)，I2C_HS_SCL_HCNT 必须大于(I2C_HS_SPKLEN+5)。

其中 I2C_FS_SPKLEN 和 I2C_HS_SPKLEN 两个寄存器可查阅后文。这些寄存器的数值都是以 I2C_CLK 时钟周期为单位。

14.3.5.2 I2C_CLK 的频率最小值

在不同速度模式下，GSC3281 对 I2C_CLK 的最小值提出了要求，具体可参见表 14-2。需要注意的是，SCL 低电平的实际值是在 SCL 低电平寄存器的配置值基础上加 1，而 SCL 高电平的实际值是在 SCL 高电平寄存器的配置值基础上加 8.

表 14-2 I2C_CLK 最小频率与 SCL 的关系

速度模式	I2C_CLK 频率最小值（MHz）	SCL 低电平的值	SCL 低电平寄存器的配置值	SCL 低电平持续时间	SCL 高电平的值	SCL 高电平寄存器的配置值	SLC 高电平持续时间
标准	2.7	13	12	4.7μs	14	6	5.2 μs
快速	12.0	16	15	1.33 μs	14	6	1.16 μs
高速 (400pf)	60.2	22	21	365ns	14	6	232ns
高速 (100pf)	128.5	24	23	186ns	14	6	108ns

14.3.5.3 I2C_CLK 最小值的计算说明

本节将对 14.3.5.2 节中的结果进行计算说明，计算的前提是 I2C_FS_SPKLEN 寄存器和 I2C_HS_SPKLEN 寄存器都将配置成 2。

标准和快速模式下的计算方法相同，下面以快速模式进行举例说明。

- 快速模式下最快速率为 400kb/s，则 SCL_PERIOD=1/400khz=2.5μs
- 假设 I2C_HCNT_FS=14
- THE I^2C-BUS SPECIFICATION 中规定了 SCL 信号的高电平时间与低电平时间，MIN_SCL_LOWtime_FS = 1300ns，MIN_SCL_HIGHtime_FS = 600ns

推导等式：

SCL_PEIRIOD_FS / (I2C_HCNT_FS + I2C_LCNT_FS)=I2C_CLK_PERIOD

I2C_LCNT_FS x I2C_CLK_PERIOD = MIN_SCL_LOWtime_FS

将上两式推导一下得下式：

I2C_LCNT_FS x SCL_PERIOD_FS / (I2C_HCNT_FS+I2C_LCNT_FS)= MIN_SCL_LOWtime_FS

即：

I2C_LCNT_FS x 2.5μs / (I2C_LCNT_FS + 14) = 1.3μs

可以推出 I2C_LCNT_FS = 16（注意是 15.166 向上取整得 16）

所以 I2C_CLK_PERIOD = 2.5μs / (16+14) = 83.3ns，则 I2C_CLK 的频率为 12Mhz，符合 I2C 协议的要求。

在高速模式下，不能简单用上述方法进行计算，如果用上述方法将不能得到正确结果。假设高速模式下，I2C 总线的电容负载为 100pf，用上述方法计算的结果如下：

- I2C_LCNT_HS = 17
- I2C_HCNT_HS = 14
- I2C_CLK 为 105.4 Mhz，即 I2C_CLK_ PERIOD=9.48ns

在去毛刺的情况下，SDA 的数据保持时间 THD;dat 需要 9 个 I2C_CLK_ PERIOD，即 85.32ns，但 THE I^2C-BUS SPECIFICATION 中 TABLE 7 对 THD;dat 有规定，最大不能超过 70ns，所以此计算方法不能用于高速模式下，正确方法如下。

I2C_CLK_ PERIOD = 70ns/9 =7.77ns

I2C_LCNT_HS x I2C_CLK_PERIOD ≥ MIN_SCL_LOWtime_HS

I2C_LCNT_HS x 7.77ns ≥ 160ns

I2C_LCNT_HS ≥ 21

根据 THE I²C-BUS SPECIFICATION 中的 TABLE 7，可得 I2C_HCNT_HS ≥ 14

SCL_PERIOD= 1/3.4Mhz= 294ns

(I2C_LCNT_HS+I2C_HCNT_HS) = SCL_PERIOD / I2C_CLK_ PERIOD=294/7.77=38

综上所述，I2C_HCNT_HS 取 14，则 I2C_LCNT_HS=38-14=24。需要注意的是，I2C_HCNT_HS 寄存器的配置值为 14-8=6，I2C_LCNT_HS 寄存器的配置值为 24-1=23。

同样的方法也可以计算出总线电容负载为 400pf 时的各数值。

14.3.6 两种地址格式

I2C 总线的 MASTER 支持两种地址格式，即 7bit 地址格式和 10bit 地址格式，是指 SLAVE 设备的地址。通过配置 I2C_TAR 寄存器来设置 SLAVE 设备地址。

7bit 地址格式如图 14-4 所示，A6-A0 对应 I2C_TAR[6:0]，即为 SLAVE 设备的地址，R/W 位由 I2C_DATA_CMD 寄存器中的 CMD 决定，如果 R/W 值为 0，则 MASTER 要对 SLAVE 设备进行写操作，如果 R/W 值为 1，则 MASTER 要对 SLAVE 设备进行读操作。

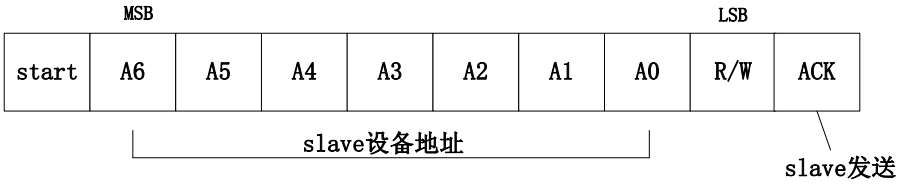


图 14-4 7bit 地址格式

10bit 地址格式如图 14-5 所示，A9-10 对应 I2C_TAR[9:0]，即为 SLAVE 设备的地址，并且以两个字节发送给 SLAVE 设备。R/W 位由 I2C_DATA_CMD 寄存器中的 CMD 决定，如果 R/W 值为 0，则 MASTER 要对 SLAVE 设备进行写操作，如果 R/W 值为 1，则 MASTER 要对 SLAVE 设备进行读操作。

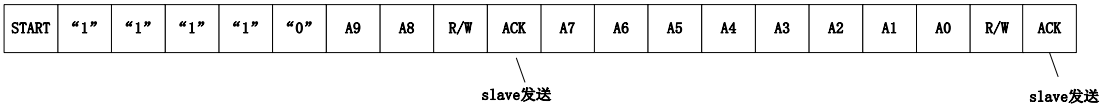


图 14-5 10bit 地址格式

14.3.7 去毛刺

I2C 协议中规定总线 SCL 和 SDA 在进入 I2C 设备内部时必须过滤掉一定长度的毛刺，对于标准和快速模式，要求能过滤掉最长为 50ns 的毛刺，而对于高速模式，要求能过滤掉最长为 10ns 的毛刺。具体可见 THE I²C-BUS SPECIFICATION 中的 TABLE 4 和 TABLE 6。

GSC3281 中的 I2C MASTER 通过 I2C_FS_SPKLEN 寄存器和 I2C_HS_SPKLEN 寄存器来设定毛刺的长度，其中 I2C_FS_SPKLEN 寄存器针设定标准和快速模式下的毛刺长度，I2C_HS_SPKLEN 寄存器设定高速模式下的毛刺长度。毛刺的长度是以 I2C_CLK 的周期为基本计数单元，如果 I2C_CLK 的周期为 10ns，将 I2C_FS_SPKLEN 寄存器设置为 5，那么标准和快速模式下，滤掉总线 SDA 和 SCL 上长度小于 50ns 的毛刺。下面以 SCL 线举例说明。

如图 14-6 所示，当外部总线 SCL 由低变高时，毛刺计数器开始计数，计数到 3 时，检

测到低电平，高电平持续时间为 30ns，由于没超过 50ns，即认为是毛刺，此时计数器清零。然后外部 SCL 再次从低变高，计数重新开始，当高电平持续 5 个时钟周期时，即 50ns，认为 SCL 信号稳定，即高电平，此时内部 SCL 由低变高，计数器同时清零。外部的 SCL 信号每次发生变化时，计数器开始清零，当状态持续时间超过 50ns 时，内部 SCL 信号才发生变化，这样可以不受毛刺影响，保证时序和逻辑的正确性。

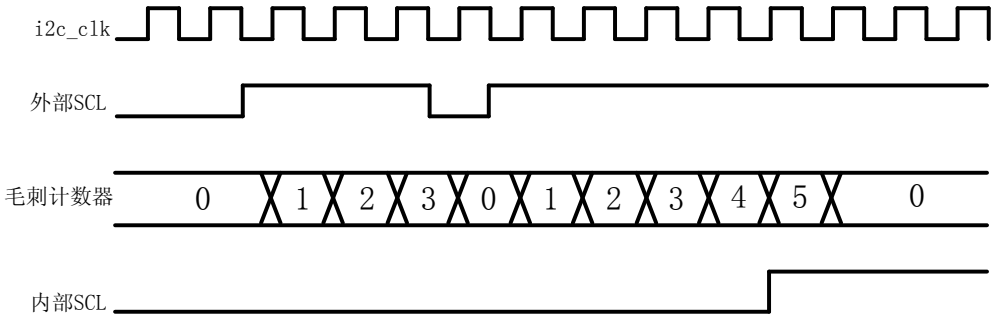


图 14-6 去毛刺示意图

14.3.8 SDA 保持时间

THE I^2C-BUS SPECIFICATION 中要求在标准速度和快速速度模式下，SDA 信号的保持时间 tHD;DAT 至少为 300ns，而在高速速度模式下要求必须 SDA 在 SCL 为下降沿时保持稳定的状态，0 或 1，而不能处于 0 和 1 之间的不稳定状态。SDA 的保持时间的定义如图 14-7 所示。

软件可以通过配置 I2C_SDA_HOLD 寄存器来使 SDA 的保持时间来满足实际要求，从而能够适应板级对 SCL 和 SDA 信号造成的延时。GSC3281 中，SDA 的保持时间至少为 1 个 I2C_CLK 时钟周期，即使配置 I2C_SDA_HOLD 为 0，MASTER 也将在 SCL 下降沿后的一个 I2C_CLK 时钟周期来驱动 SDA 信号。

在变更速度模式时，必须重新配置 I2C_SDA_HOLD 寄存器来满足对应速度模式下的 SDA 保持时间。

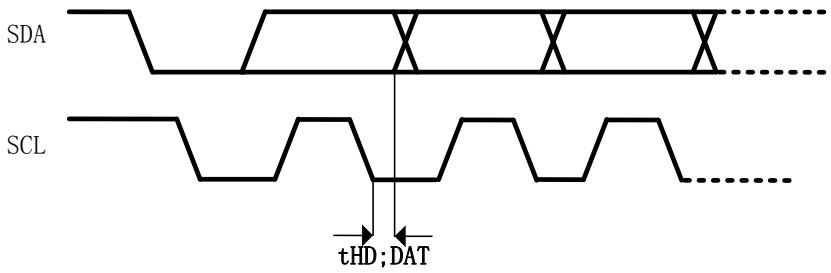


图 14-7 SDA 保持时间

14.3.9 FIFO 与 DMA 操作

在 I2C MASTER 设备中，有发送 FIFO 和接收 FIFO，地址深度均为 8，可以通过读取寄存器 I2C_TXFLR 和 I2C_RXFLR 寄存器得知发送 FIFO 和接收 FIFO 中的数据个数。CPU 可以直接通过写 I2C_DATA_CMD 寄存器向发送 FIFO 中写入数据，也可以直接通过读 I2C_DATA_CMD 寄存器从接收 FIFO 中读取数据。

通过外部的 DMA 控制器可以实现对数据的 DMA 传输。其中要对 DMA 控制器进行相应的配置，请查看用户手册的 DMA 控制器部分。下面以 DMA 发送和 DMA 接收进行举例说明。

I2C 进行 DMA 发送的过程如图 14-8 所示。假设 I2C MASTER 要向 SLAVE 设备发送 30 个数据，DMA 的触发阈值 DMATDL（见 I2C_DMA_TDLR 寄存器）设为 2，则将 DMA 每次 BURST 的数据量设为 6 时效率最高。FIFO 中有数时，MASTER 则将发送 FIFO 中的数据发送给 SLAVE 设备，当 FIFO 中的数据小于或等于 2 时，MASTER 则会向 DMA 控制器提出发送请求，然后 DMA 控制器将发送一个 BURST 的数据给 FIFO，填满整个 FIFO。这样 5 个 BURST 即可完成 30 个数据的发送。

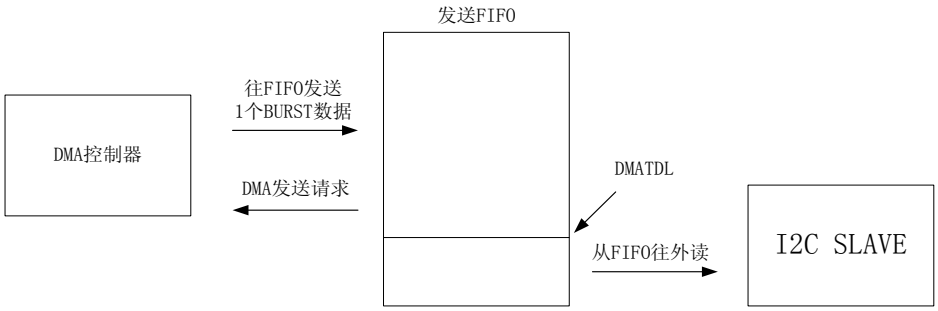


图 14-8 DMA 发送

I2C 进行 DMA 接收的过程如图 14-9 所示。假设 I2C MASTER 要从 SLAVE 设备接收 30 个数据，将 DMARDL（见 I2C_DMA_RDLR 寄存器）设为 5，即 DMA 的触发阈值为 6，则将 DMA 每次 BURST 的数据量设为 6 时效率最高。当接收 FIFO 中的数据量等于大于 6 时，MASTER 则会向 DMA 控制器提出接收请求，然后 DMA 控制器将从接收 FIFO 中读取一个 BURST 的数据。这样 5 个 BURST 即可完成 30 个数据的接收。

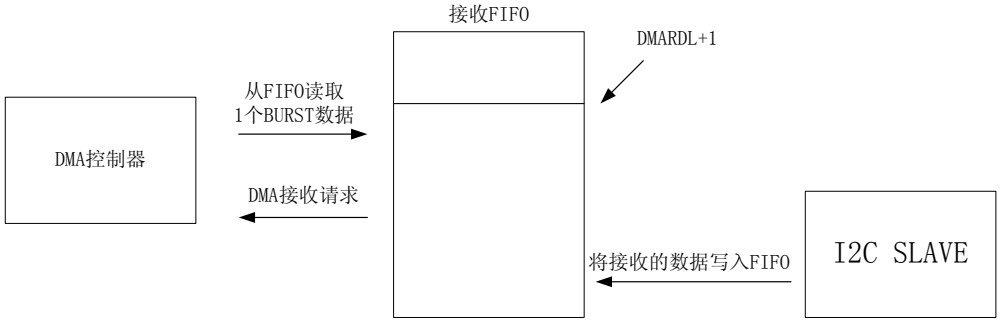


图 14-9 DMA 接收

14.3.10 发送 FIFO 与 I2C_DATA_CMD 寄存器

当发送 FIFO 变空时，MASTER 并不会产生 STOP 信号，而是将 SCL 信号保持成低电平，直到有新的数据写入到发送 FIFO 中。必须向 I2C_DATA_CMD 寄存器的 bit9 写 1，才会产生 STOP 信号。I2C_DATA_CMD 寄存器如图 14-10 所示，DATA 域可读可写，读操作可以得到 MASTER 接收到的数据，写操作可以向 SLAVE 设备发送数据。CMD 域只写，置 0 表示写操作，置 1 表示读操作。STOP 域只写，置 1 会使 MASTER 在发送或接收完一个字节后产生 STOP 信号。RESTART 域只写，置 1 会使 MASTER 在发送或接收完一个字节后产生 RESTART 信号。

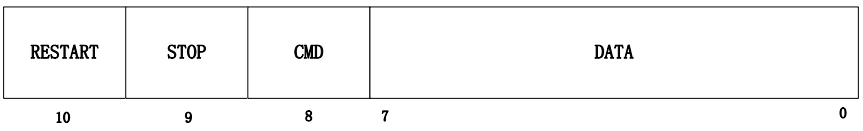


图 14-10 I2C_DATA_CMD 寄存器

I2C_DATA_CMD 寄存器的配置与发送 FIFO 的操作相配合，使 MASTER 能够正确的控制 I2C

总线的发送与接收，图 14-11 到图 14-18 对各种情况下的时序进行了图示说明，这对软件编程有重要作用。

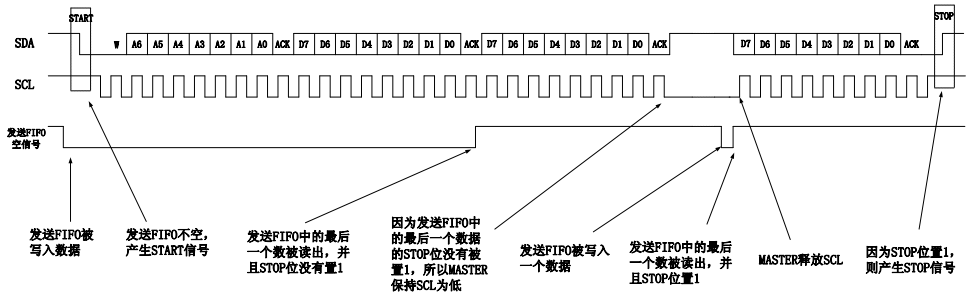


图 14-11 发送状态下，发送 FIFO 变空并且 STOP 位置 1 的时序图

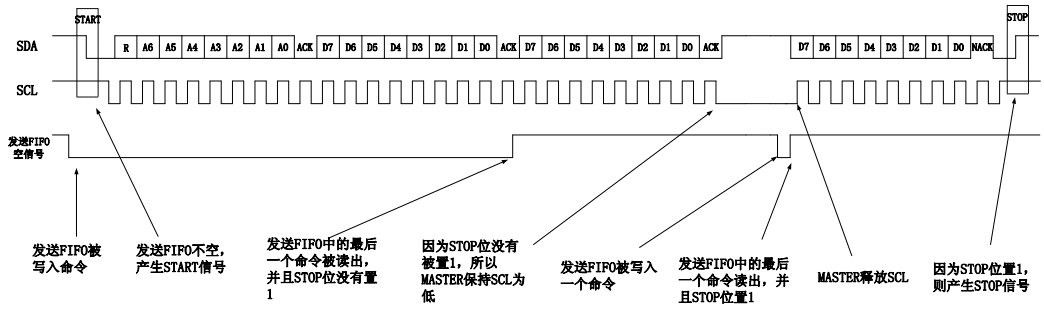


图 14-12 接收状态下，发送 FIFO 变空并且 STOP 位置 1 的时序图

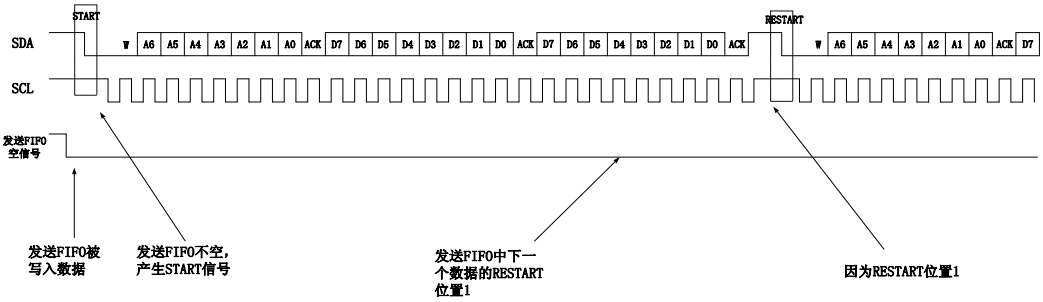


图 14-13 发送状态下，发送 FIFO 非空时并且 RESTART 位置 1 的时序图

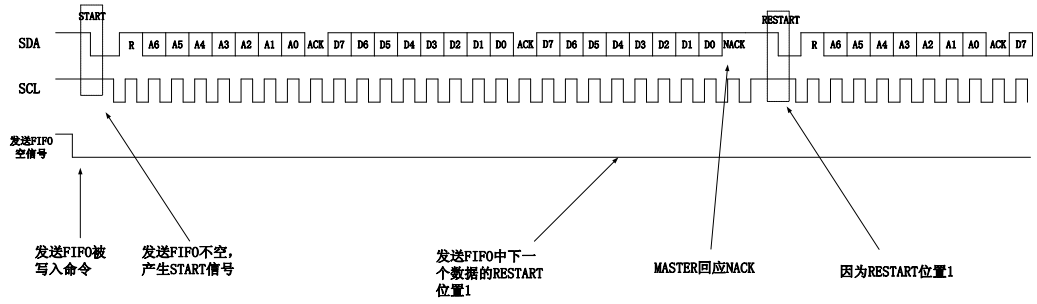


图 14-14 接收状态下，发送 FIFO 非空时并且 RESTART 位置 1 的时序图

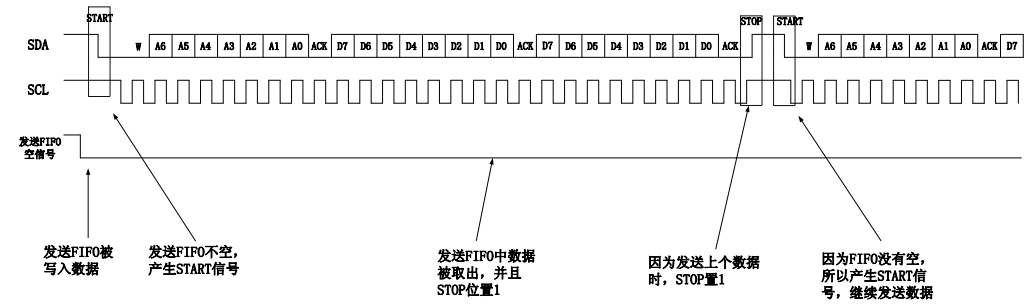


图 14-15 发送状态下，发送 FIFO 非空时并且 STOP 位置 1 的时序图

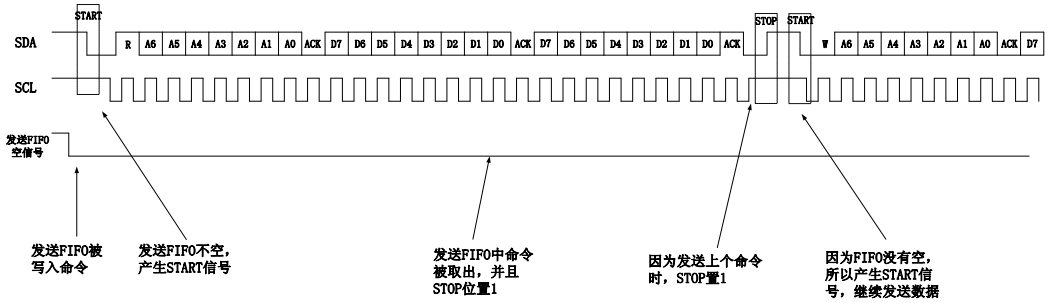


图 14-16 接收状态下，发送 FIFO 非空时并且 STOP 位置 1 的时序图

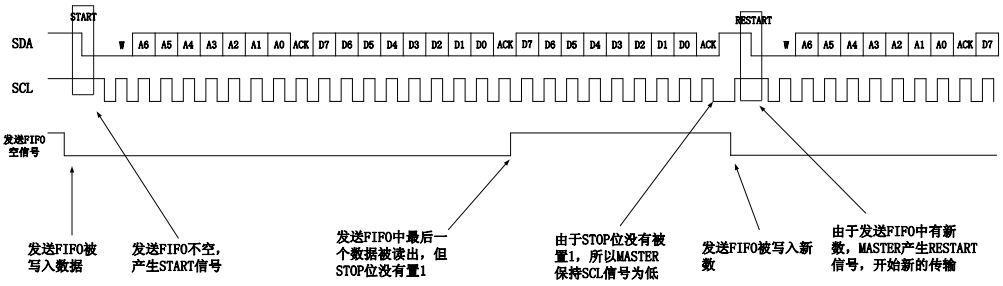


图 14-17 发送状态下，发送 FIFO 由空变非空时产生 RESTART 信号的时序图

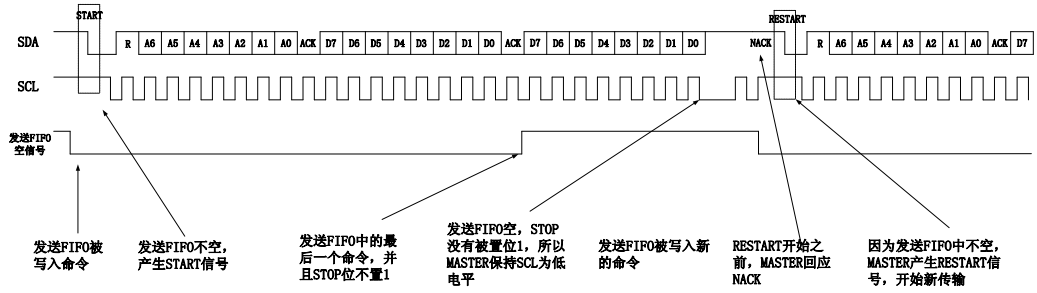


图 14-18 接收状态下，发送 FIFO 由空变非空时产生 RESTART 信号的时序图

14.4 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

14.5 编程指导

当前版本用户手册暂不提供详细编程指导。

15 I2S 控制器

15.1 概述

目前，大多数的数字音频系统对音响数据的采集、处理和传输符合 I2S 总线标准。

GSC3281 芯片内置 I2S 总线接口，一个发送、接收通道。支持中断与 DMA 两种传输方式进行数据传输，并为左，右声道数据各提供一个深度为 8 的 FIFO。

15.2 引脚描述

表 15-1 I2S 控制器引脚描述

名称	类型	上拉 下拉	功能描述
i2s_clk	O	-	I2S 位时钟
i2s_sdi	I	-	I2S 串行数据输入
i2s_sdo	O	-	I2S 串行数据输出
i2s_ws	O	-	I2S 声道选择信号(采样时钟)

15.3 结构框图

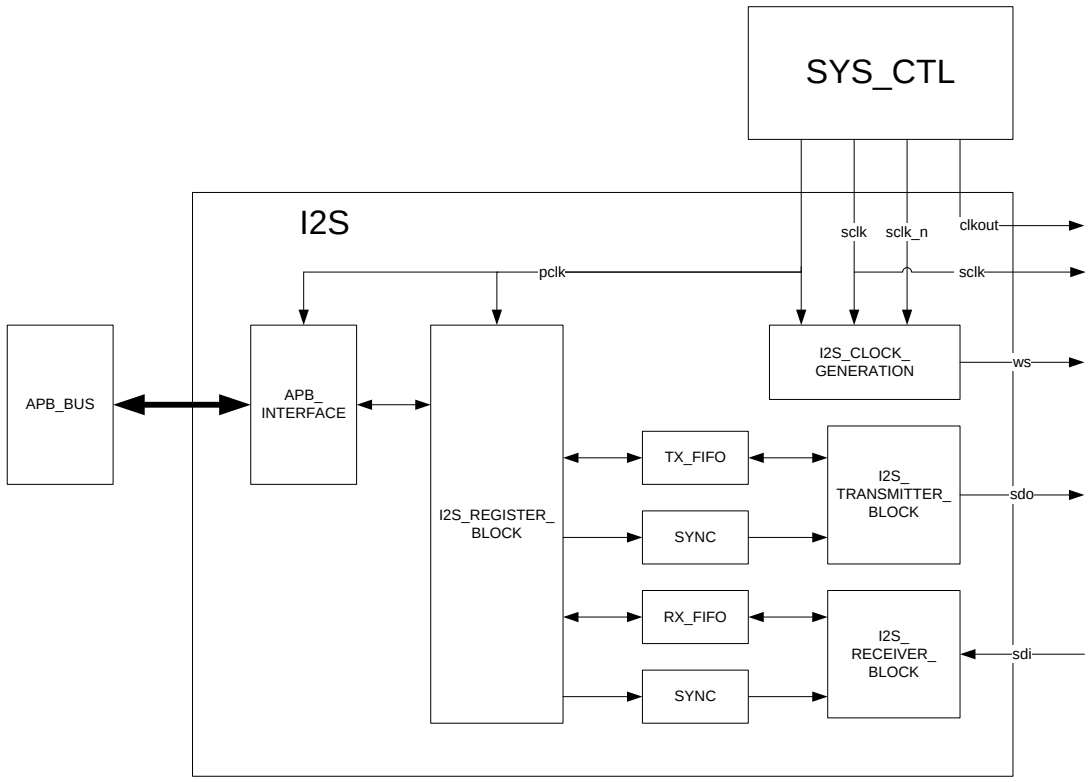


图 15-1 I2S 结构框图

15.4 功能说明

GSC3281 芯片内置 I2S 总线接口，为 I2S 主设备，与外设间的接口信号包括串行输出数据 sdo、串行输入数据 sdi、位时钟 sclk、采样时钟 ws。为了满足部分 I2S 音频设备的需要，可将 GSC3281 芯片的时钟输出信号 clkout 作为 I2S 主时钟输出信号(mclk)。

I2S 总线接口按照标准 I2S 总线结构进行数据的接收和发送，当 ws 信号变化后，等待一个 sclk 周期再开始传输一帧数据的最高位。

15.4.1 I2S 使能

在使用 I2S 接收数据和发送数据之前，必须先使能 I2S 控制器。可以将 I2S_IER 寄存器的 IEN 位置为 1 来使能 I2S 控制器。将 IER 寄存器的 IEN 位置为 0 将禁止 I2S 控制器。当禁止 I2S 控制器后，会出现以下情况：

1) TX 和 RX FIFO 清空，读写指针清零，正在传输的数据丢失。

2) 其他可编程的使能信号，如 I2S 接收通道使能、I2S 发送通道使能等均无效。

3) I2S 输出的 sclk、ws 信号均变为无效，保持低电平。

当 I2S 使能后，I2S 控制器开始左声道数据传输，并在 ws 信号变为 1 的一个 sclk 周期后传输右声道数据。必须要确保在这之前 TX FIFO 中有数据并且外设在 sclk 有效后不会丢失这第一帧数据。

15.4.2 I2S 发送数据

可通过软件向 I2S 控制器的 TX FIFO 中写入音频数据，使 I2S 控制器向 I2S 总线上发送音频数据。音频数据根据 sclk 和 ws 信号的时序按照 I2S 格式发送。

要使用发送功能，需要配置两个使能信号，发送模块使能(I2S_ITER[0])以及发送通道使能(I2S_TER[0])。

I2S 使能(I2S_IER)、发送模块使能(I2S_ITER[0])与发送通道使能(I2S_TER[0])的优先级为 I2S 使能最高，发送模块使能次之，发送通道使能最低。当 I2S 使能无效时，发送模块使能与发送通道使能均无效；发送模块使能无效时，发送通道使能无效。

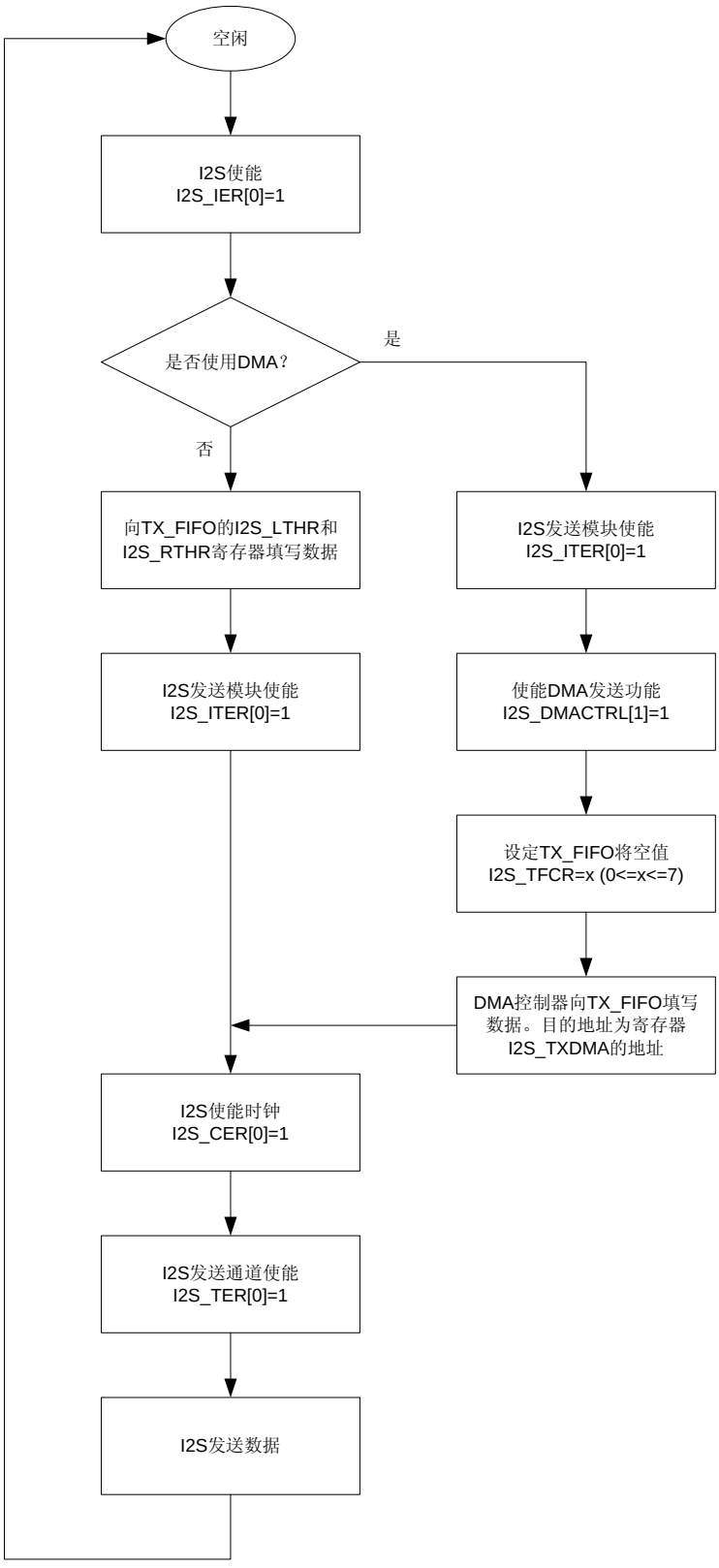


图 15-2 I2S 发送功能基本使用流程

15.4.2.1 发送模块使能

I2S 发送使能寄存器(I2S_ITER)的第 0 位是发送模块使能位(TX_EN)，它的作用是使能或者关闭 I2S 的发送功能。要使能 I2S 控制器的发送模块，设置 I2S 发送使能寄存器(I2S_ITER)的第 0 位即发送模块使能位(TX_EN)为 1；要关闭 I2S 控制器的发送模块，设置 I2S 发送使能寄存器(I2S_ITER)的第 0 位即发送模块使能位(TX_EN)为 0。

当关闭 I2S 控制器的发送模块时，I2S 会出现以下情况：

1) 正在发送的数据丢失，数据输出信号(sdo)变为低电平。
2) 在 TX_FIFO 中的数据不会丢失，并可向 FIFO 中写入新数据。
3) 对 I2S 控制器的设置会保存，不会回到初始值。
4) I2S 控制器的发送通道使能无效。

当 I2S 控制器的发送模块关闭时，可以使用清除发送 FIFO 寄存器(I2S_TXFFR[0]=1)或者清除通道 0 发送 FIFO 寄存器(I2S_TFF[0]=1)来清除 TX_FIFO 中的数据。

I2S 发送使能寄存器(I2S_ITER)的第 0 位在系统复位后的初始值为 0，也就是说发送模块初始状态为关闭状态。

15.4.2.2 发送通道使能

发送使能寄存器的第 0 位(I2S_TER[0])为发送通道使能位，为 1 时表示发送通道使能，为 0 时表示发送通道关闭。当数据发送正在进行时，可以通过发送通道使能位关闭发送通道，并对发送通道的设置进行改变，然后再重新使能发送通道。

当关闭 I2S 控制器的发送通道，I2S 会出现以下情况：

1) 正在发送的数据丢失，数据输出信号(sdo)变为低电平。
2) 在 TX_FIFO 中的数据不会丢失，并可向 FIFO 中写入新数据。
3) 对 I2S 控制器发送通道的设置会保存，不会回到初始值。

发送使能寄存器(I2S_TER)的第 0 位在系统复位后的初始值为 1，也就是说发送通道初始状态为使能状态。

15.4.2.3 发送音频数据分辨率

I2S 发送音频数据的分辨率最高为 32bit，可以设置为 12、16、20、24、或 32bit。例如，当设置音频数据的分辨率为 24bit，写入 TX_FIFO 中的数据只有低 24bit 有效，有效数据的最高位(TX_FIFO.bit23)最先发送，有效数据的最低位(bit0)保存在 TX_FIFO 的最低位(TX_FIFO.bit0)。

可通过 I2S_TCR[2:0](详见寄存器说明)来修改音频数据的分辨率，并且必须在通道关闭的情况下修改。

I2S 音频数据的分辨率在系统复位后的初始值为 32bit。

15.4.2.4 发送通道 FIFO

发送通道 FIFO 包含两个部分，左声道数据 FIFO 和右声道数据 FIFO，深度均为 8，宽度均为 32bit。有以下四种方式可以清空 FIFO 和使 FIFO 指针复位：

1) 系统复位。
2) 关闭 I2S 模块(I2S_IER[0]=0)。

3) 清空发送模块(I2S_TXFFR[0]=1)。
4) 清空发送通道(I2S_TFF[0]=1)。

15.4.2.5 发送通道写数据

有两种方式可以向 I2S 的发送通道写数据，一种是直接向 I2S 的左声道 TX_FIFO、右声道 TX_FIFO 中写数据。另一种是通过 DMA 的方式，将左声道、右声道数据按顺序循环写入 I2S_TXDMA 寄存器。

如果不使用 DMA 功能，可直接向 I2S 的左声道 TX_FIFO(I2S_LTHR)、右声道 TX_FIFO(I2S_RTHR)中写数据。注意一定要按照顺序，先写左声道数据，再写右声道数据，否则会使中断和状态位无效，并造成左右声道数据不同步。

如果 I2S 控制器在传输下一帧左声道数据前 FIFO 已空，则 I2S 会发送 0，直到 FIFO 中有新的有效数据。

只有 FIFO 不满时向 FIFO 中写入数据才有效，如果 FIFO 已满，再向 FIFO 中写入数据，数据会丢失并产生发送 FIFO 溢出中断。

15.4.2.6 DMA 发送功能

如果使用 I2S 的 DMA 发送功能，要先通过配置 DMA 使能寄存器(I2S_DMACTRL[1]=1)来使能 I2S 控制器的 TX_DMA 功能，然后配置 DMA 控制器，通过 DMA 控制器将左声道、右声道数据按顺序自动的写入 I2S_TXDMA 寄存器。

当 TX_FIFO 中数据个数达到发送 FIFO 空阈值(通过寄存器 I2S_TFCR 设置)，I2S 控制器会自动向 DMA 控制器请求数据，一次请求数据的个数由 DMA 控制器通过 M_size 设置，需要满足：

M_size <=(I2S 左右声道 FIFO 深度 8–发送 FIFO 空阈值)*2

I2S 控制器可通过复位发送 DMA 寄存器(I2S_RTXDMA)来复位发送 DMA FIFO 中的数据，不过，当在传输左、右声道数据的中间状态时，复位发送 DMA 寄存器(I2S_RTXDMA)是无效的。

15.4.3 I2S 接收数据

I2S 控制器可以通过输入信号 sdi 接收 I2S 总线上符合 I2S 格式的数据，将之存入 RX_FIFO 中，并可通过软件从 RX FIFO 中读出音频数据。

要使用接收功能，需要配置两个使能信号，接收模块使能(I2S_IRER[0])以及接收通道使能(I2S_RER[0])。

I2S 使能(I2S_IER)、接收模块使能(I2S_IRER[0])与接收通道使能(I2S_RER[0])的优先级为 I2S 使能最高，接收模块使能次之，接收通道使能最低。当 I2S 使能无效时，接收模块使能与接收通道使能均无效；接收模块使能无效时，接收通道使能无效。

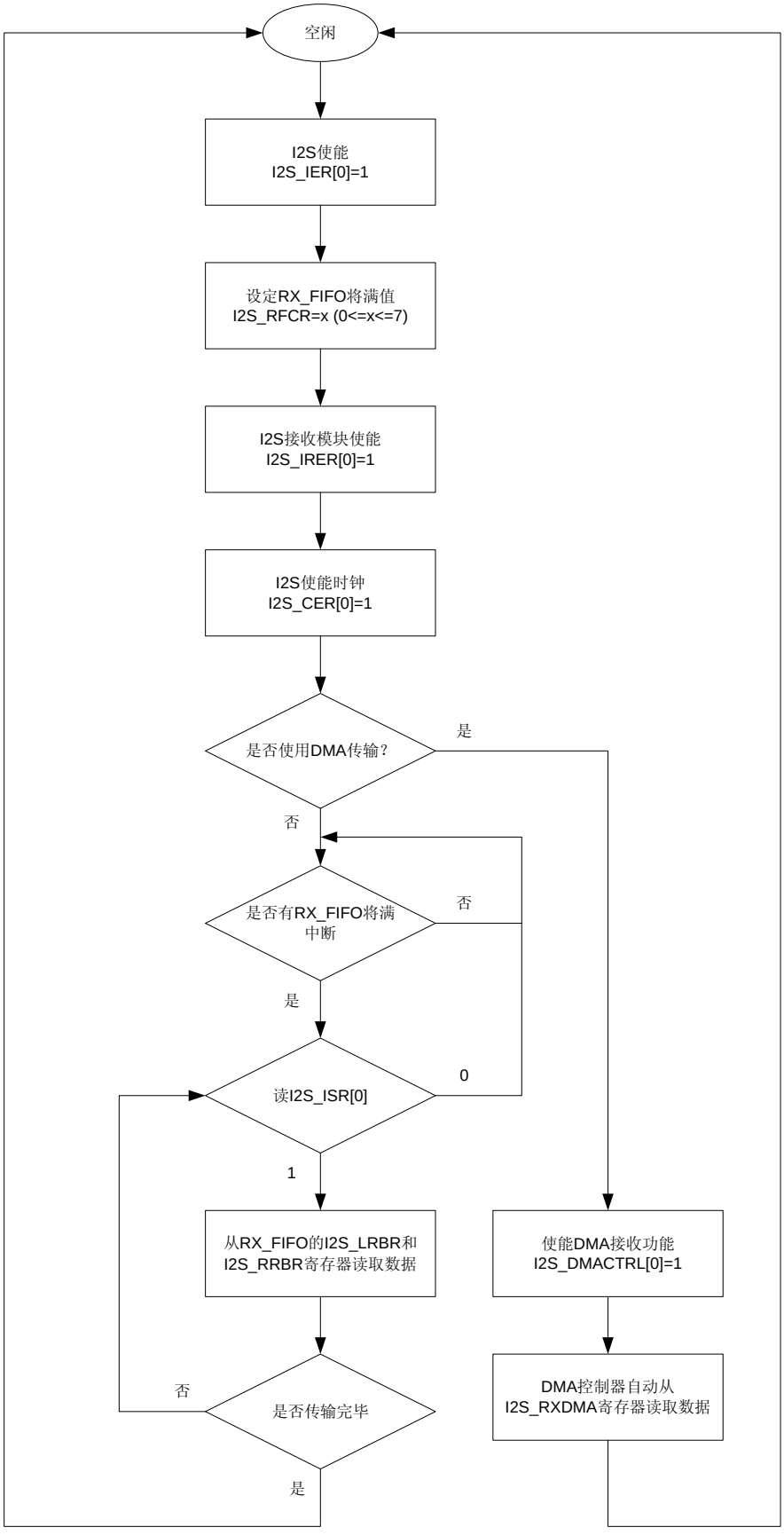


图 15-3 I2S 接收功能基本使用流程

15.4.3.1 接收模块使能

I2S 接收使能寄存器(I2S_IRER)的第 0 位是接收模块使能位(RX_EN)，它的作用是使能或者关闭 I2S 的接收功能。要使能 I2S 控制器的接收模块，设置 I2S 接收使能寄存器(I2S_IRER)的第 0 位即接收模块使能位(RX_EN)为 1；要关闭 I2S 控制器的接收模块，设置 I2S 接收使能寄存器(I2S_IRER)的第 0 位即接收模块使能位(RX_EN)为 0。

当关闭 I2S 控制器的接收模块，I2S 会出现以下情况：

1) 正在接收的数据丢失。
2) 在 RX_FIFO 中的数据不会丢失，并可从 RX_FIFO 中读出已有的数据。
3) 对 I2S 控制器接收模块的设置会保存，不会回到初始值。
4) I2S 控制器的接收通道使能会无效。

当 I2S 控制器的接收模块关闭时，可以使用清除接收 FIFO 寄存器(I2S_RXFFR[0]=1)或者清除通道 0 接收 FIFO 寄存器(I2S_RFF[0]=1)来清除 RX_FIFO 中的数据。

I2S 接收使能寄存器(I2S_IRER)的第 0 位在系统复位后的初始值为 0，也就是说接收模块初始状态为关闭状态。

15.4.3.2 接收通道使能

接收使能寄存器的第 0 位(I2S_RER[0])是接收通道使能位，为 1 时表示接收通道使能，为 0 时表示接收通道关闭。当数据接收正在进行时，可以通过接收通道使能位关闭接收通道，对接收通道的设置进行改变，然后再重新使能接收通道。

当关闭 I2S 控制器的接收通道，I2S 会出现以下情况：

1) 正在接收的数据丢失。
2) 在 RX_FIFO 中的数据不会丢失，并可从 FIFO 中读出已有数据。
3) 对 I2S 控制器接收通道的设置会保存，不会回到初始值。

接收使能寄存器(I2S_RER)的第 0 位在系统复位后的初始值为 1，也就是说接收通道初始状态为使能状态。

15.4.3.3 接收音频数据分辨率

I2S 接收音频数据的分辨率最高为 32bit，可以设置为 12、16、20、24、或 32bit。例如，当设置音频数据的分辨率为 24bit，RX_FIFO 中的数据只有高 24bit 有效，I2S 接收时先接收到的是音频数据的最高位(bit23)，将音频数据的最高位保存在 RX_FIFO 的最高位(RX_FIFO.bit31)。

可通过 I2S_RCR[2:0] (详见寄存器说明)来修改音频数据的分辨率，并且必须在通道关闭的情况下修改。

I2S 音频数据的分辨率在系统复位后的初始值为 32bit。

15.4.3.4 接收通道 FIFO

接收通道 FIFO 包含两个部分，左声道数据 FIFO 和右声道数据 FIFO，深度均为 8，宽度均为 32bit。有以下四种方式可以清空 FIFO 和使 FIFO 指针复位：

1) 系统复位。

2) 关闭 I2S 的 APB 总线接口(I2S_IER[0]=0)。
3) 清空接收模块(I2S_RXFFR[0]=1)。
4) 清空接收通道(I2S_RFF[0]=1)。

15.4.3.5 接收通道读数据

有两种方式可以从 I2S 的接收通道读数据，一种方式是直接从 I2S 的左声道 RX_FIFO、右声道 RX_FIFO 中读数据。另一种方式是通过 DMA 的方式，将左声道、右声道数据按顺序循环从 I2S_TXDMA 寄存器中读出。

如果不使用 DMA 功能，可直接从 I2S 的左声道 RX_FIFO(I2S_LRBR)、右声道 RX_FIFO(I2S_RRBR)中读数据。

15.4.3.6 DMA 接收功能

如果使用 DMA 功能，要先通过配置 DMA 使能寄存器(I2S_DMACTRL[0]=1)来使能 I2S 控制器的 RX_DMA 功能，然后配置 DMA 控制器，通过 DMA 控制器将左声道、右声道数据按顺序自动的从 I2S_RXDMA 寄存器的地址读出。

当 RX_FIFO 中数据达到接收 FIFO 满阈值(通过寄存器 I2S_RFCR 设置)，I2S 控制器会自动向 DMA 控制器请求读数据。一次请求数据的个数由 DMA 控制器通过 M_size 设置，需要满足：

M_size <=接收 FIFO 满阈值*2(包括左声道 FIFO 和右声道 FIFO)

I2S 控制器可通过复位接收 DMA 寄存器(I2S_RRXDMA)来复位接收 DMA FIFO 中的数据，不过，当在接收左、右声道数据的中间状态时，复位接收 DMA 寄存器(I2S_RRXDMA)是无效的。

15.4.4 中断与中断屏蔽

I2S 控制器共有 4 个中断，包括发送 FIFO 将空中断、发送 FIFO 溢出中断、接收 FIFO 将满中断和接收 FIFO 溢出中断。每一个中断都可通过寄存器读出状态以及通过寄存器设置是否屏蔽。

1) 发送 FIFO 将空中断：表示发送 FIFO 中的数据个数低于寄存器 I2S_TFCR 设置的发送 FIFO 空阈值。这个中断的状态可由寄存器 I2S_ISR[4]来查询。这个中断可将寄存器 I2S_IMR[4]设置为 1 来屏蔽。
2) 发送 FIFO 溢出中断：表示发送 FIFO 满但是仍然尝试向发送 FIFO 中写数据。这个中断的状态可由寄存器 I2S_ISR[5]来查询。这个中断可将寄存器 I2S_IMR[5]设置为 1 来屏蔽。
3) 接收 FIFO 将满中断：表示接收 FIFO 中的数据个数高于寄存器 I2S_RFCR 设置的接收 FIFO 满阈值。这个中断的状态可由寄存器 I2S_ISR[0]来查询。这个中断可将寄存器 I2S_IMR[0]设置为 1 来屏蔽。
4) 接收 FIFO 溢出中断：表示接收 FIFO 满但是仍然接收到音频数据。这个中断的状态可由寄存器 I2S_ISR[1]来查询。这个中断可将寄存器 I2S_IMR[1]设置为 1 来屏蔽。

15.4.5 时钟产生

I2S 控制器的工作时钟 sclk 由系统控制模块 SYSCTL 提供。具体配置请参见系统控制模块。I2S 控制器为主设备，为外围设备提供系统主时钟 clkout(可选)、位时钟 sclk 以及采样时钟 ws。

15.4.5.1 clkout 产生

为了满足部分 I2S 音频设备的需要，可将 GSC3281 芯片的时钟输出信号 clkout 作为 I2S 主时钟输出信号(mclk)。clkout 同时也是 sclk 时钟的源时钟。有些 I2S 音频芯片不需要使用这个时钟输出信号。

clkout 是 spll_clk 的分频时钟，可通过系统控制模块的 SYS_CLKDIV_CLKOUT 寄存器对 clkout 进行配置，可配置为 spll_clk 的 2 分频、3 分频、……、最高 512 分频。

15.4.5.2 sclk 产生

I2S 为外设提供位时钟 sclk。sclk 同时也作为 I2S 控制器的工作时钟，是由系统控制模块提供的。这个时钟的频率可以通过系统控制模块来配置。为了使 I2S 输出时钟更为灵活，I2S 控制器提供 sclk 使能信号和 sclk 门控信号。

可以通过 I2S 时钟使能寄存器(I2S_CER)使能和关闭 sclk。

可以通过 I2S 时钟配置寄存器的门控位(I2S_CCR[2:0])来配置 sclk 的门控信息，如图 15-4 所示。例如，当 I2S 时钟配置寄存器的门控位(I2S_CCR[2:0])设置为 1，表示每个声道传输时在 12 个周期后产生门控信号，停止 sclk 时钟。

sclk 是 clkout 的分频时钟，可通过系统控制模块 SYSCTL 的 SYS_CLKDIV_I2S 寄存器对 sclk 进行配置，可配置为 clkout 的 2 分频、3 分频、……、最高 32 分频。

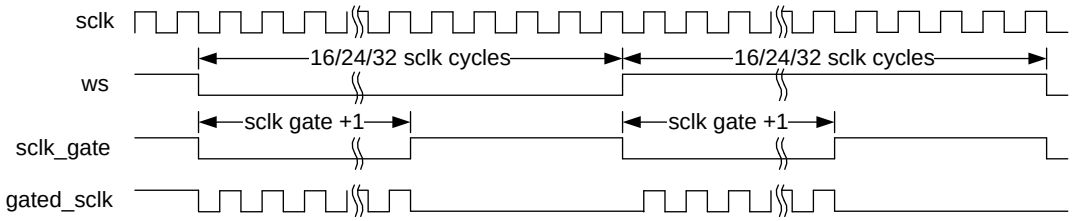


图 15-4 I2S sclk 门控信号波形示意图

15.4.5.3 word select 产生

I2S 为外设提供采样时钟 ws。ws 为低电平代表传输的为左声道数据，ws 为高电平代表传输的为右声道数据。每次传输都是先传输左声道数据，再传输右声道数据。ws 可通过 I2S 时钟配置寄存器的 ws 位(I2S_CCR[4:3])来设置，可以设置为 16bit 采样、24bit 采样和 32bit 采样。

15.4.5.4 时钟配置示例

假设在系统 spll_clk 频率 500MHZ、hclk 166.7MHZ 的情况下，采用 I2S 音频芯片 CS4334，实现 44.1KHZ 采样频率与 16bit 采样精度的声音数据传输，配置时钟过程如下：

由于 CS4334 芯片需要一个系统主时钟 mclk，因此需要 clkout 信号与之连接。查看 CS4334 芯片手册，44.1KHZ 采样频率下，mclk 可选 5.6448MHz、8.4672MHz、11.286MHz、169344MHz 或 22.5792MHz。我们选择 mclk 为 11.286MHz，为采样频率的 256 倍。由于 spll_clk 为 500MHz，clkout 为 spll_clk 的分频，分频系数为：

clkout 分频系数 = 500MHz/11.286MHz ≈ 44 = 0x2C

因此系统控制模块 SYSCTL 的 SYS_CLKDIV_CLKOUT 寄存器中应该写入值 0x2B，表示 44 分频。clkout 的实际频率为 11.3636MHz，误差为 0.66%

由 44.1KHZ 采样频率 16bit 采样精度，可计算出 sclk 的频率为：

sclk 频率 = 44.1KHz * 16 * 2 = 1.4112MHz

sclk 为 clkout 的分频，分频系数为：

sclk 分频系数 = 11.3636MHz/1.4112MHz ≈ 8 = 0x8

因此系统控制模块 SYSCTL 的 SYS_CLKDIV_I2S 寄存器中应该写入值 0x7，表示 8 分频。sclk 的实际频率为 1.42045MHz。

由于采样精度为 16bit，I2S_CCR[4:3]需要设置为 0，表示 ws 信号为 16bit 采样。

15.5 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

15.6 编程指导

当前版本用户手册暂不提供详细编程指导。

16 PS2 控制器

16.1 概述

PS2 通讯协议是一种双向同步串行通讯协议。PS2 控制器有两个输入输出信号：时钟输入输出信号 CLK 和数据输入输出信号 DATA。通讯的两端通过 CLK 同步，并通过 DATA 交换数据。

GSC3281 芯片和 PS2 设备间通讯时，GSC3281 芯片可以把 CLK 拉到低电平抑制 PS2 设备发送数据，而 PS2 设备则不会抑制 GSC3281 芯片发送数据。

GSC3281 芯片包含两个 PS2 控制器 PS2_0 和 PS2_1，可连接 PS2 键盘和 PS2 鼠标，两个 PS2 控制器功能上完全一样。下文中用 PS2_n 表示 PS2_0 和 PS2_1。n=0,1。

GSC3281 芯片 PS2 控制器未实现键盘扫描码与 ASCII 码的转换，也未实现几种键盘扫描码间的转换，这些转换需要软件进行处理。

16.2 引脚描述

表 16-1 PS2 控制器引脚描述

名称	类型	上拉 下拉	功能描述
ps_clk_0	B	-	PS2_0 时钟信号
ps_dat_0	B	-	PS2_0 数据信号
ps_clk_1	B	-	PS2_1 时钟信号
ps_dat_1	B	-	PS2_1 数据信号

16.3 功能框图

GSC3281 芯片包含两个 PS2 控制器，这两个控制器内部结构完全相同，如图 16-1 所示：

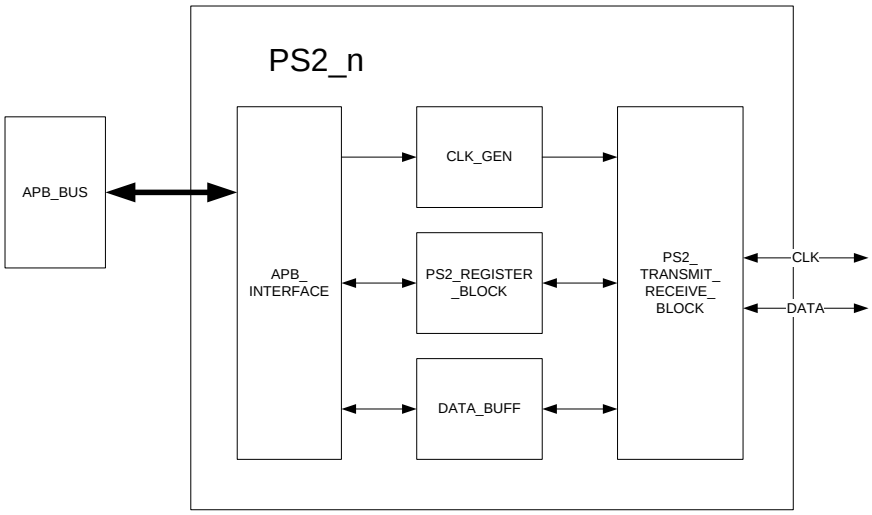


图 16-1 PS2 功能框图

16.4 功能说明

GSC3281 芯片内包含两个 PS2 控制器，每个 PS2 控制器都可以作为鼠标或者键盘使用。建议第一个 PS2 控制器 PS2_0 作为键盘，第二个 PS2 控制器 PS2_1 作为鼠标。两个控制器功能相同；支持 PS2 协议标准；控制器有自检测功能；支持 11 位数据帧格式，包括 1 位起始位，8 位数据位，1 位奇校验位，1 位停止位；独立的发送与接收模块；一个中断连接到中断控制器。

16.4.1 时钟产生

两个 PS2 控制器时钟结构类似，均有两个时钟：pclk 和 ps2_wk_clk。其中 pclk 是整个模块的同步时钟，ps2_wk_clk 是 pclk 的分频时钟，均可配置，由系统控制模块 SYSCTL 提供。ps2_wk_clk 为 pclk 的 2 分频、4 分频、6 分频……1024 分频，在系统控制模块 SYSCTL 中设置。

16.4.2 中断与中断使能

两个 PS2 控制器各有一个中断：输入缓冲器满中断。在输入缓冲器满且中断没有被屏蔽的情况下触发这个中断。

中断的使能通过命令寄存器 PS2_n_CR 的 bit0 和 bit1 设置。如果 PS2 控制器连接键盘，可置 PS2_n_CR.bit0 为 1 使能键盘中断；如果 PS2 控制器连接鼠标，可置 PS2_n_CR.bit1 为 1 使能鼠标中断。

输入缓冲器中的数据来源有两个：控制器的命令返回数据和设备发送过来的数据。

有三种方式可以清除中断：读输入缓冲器，将收到的数据读出；通过置命令寄存器 PS2_n_CR.bit4 和 bit5 为 0 禁止鼠标和键盘功能；通过置命令寄存器 PS2_n_CR.bit0 和 bit1 为 0 禁止鼠标和键盘中断。

16.4.3 设置命令字

向 PS2 控制器的命令寄存器 PS2_n_CR 写特定的命令字可以实现一些特定的命令，详细命令见表 16-2。

表 16-2 PS2 控制器命令字

值	命令	描述
0xAE	使能键盘接口	向命令寄存器 PS2_n_CR 写入命令字 0xAE，则命令寄存器 PS2_n_CR 第 4 位 EN_KB 置 1。
0xAD	禁止键盘接口	向命令寄存器 PS2_n_CR 写入命令字 0xAD，则命令寄存器 PS2_n_CR 第 4 位 EN_KB 置 0。
0xA8	使能鼠标接口	向命令寄存器 PS2_n_CR 写入命令字 0xA8，则命令寄存器 PS2_n_CR 第 5 位 EN_MS 置 1。
0xA7	禁止鼠标接口	向命令寄存器 PS2_n_CR 写入命令字 0xA7，则命令寄存器 PS2_n_CR 第 5 位 EN_MS 置 0。
0x20	读命令字	向命令寄存器 PS2_n_CR 写入命令字 0x20，则返回命令字，命令字可以通过读输入缓冲器 PS2_n_IBUF 读取。
0x60	写命令字	向命令寄存器 PS2_n_CR 写入命令字 0x60，则将输出缓冲器 PS2_n_OBUF 作为配置寄存器，可向输出缓冲器 PS2_n_OBUF 写入数据，用于配置命令寄存器 PS2_n_CR 相应位。例如：先向 PS2_n_CR 寄存器写入命令字 0x60，再向输出缓冲器 PS2_n_OBUF 写入数据 0x1，则将 PS2_n_CR 寄存器的 bit0 置为 1。
0xA1	控制器自测试	向命令寄存器 PS2_n_CR 写入命令字 0xA1，则输入缓冲器 PS2_n_IBUF 返回 0x0。
0xA4	控制器自测试	向命令寄存器 PS2_n_CR 写入命令字 0xA4，则输入缓冲器 PS2_n_IBUF 返回 0xF1。
0xAA	控制器自测试	向命令寄存器 PS2_n_CR 写入命令字 0xAA，则输入缓冲器 PS2_n_IBUF 返回 0x55。
0xAB	控制器自测试	向命令寄存器 PS2_n_CR 写入命令字 0xAB，则输入缓冲器 PS2_n_IBUF 返回 0x0。
0xAF	控制器自测试	向命令寄存器 PS2_n_CR 写入命令字 0xAF，则输入缓冲器 PS2_n_IBUF 返回 0x0。
0xC0	控制器自测试	向命令寄存器 PS2_n_CR 写入命令字 0xC0，则输入缓冲器 PS2_n_IBUF 返回 0xFF。
0xD0	控制器自测试	向命令寄存器 PS2_n_CR 写入命令字 0xD0，则输入缓冲器 PS2_n_IBUF 返回 0x01。
0xD2	写键盘缓冲器	向命令寄存器 PS2_n_CR 写入命令字 0xD2，则可将数据写到输出缓冲器 PS2_n_OBUF，随后这个数据会自动流入输入缓冲器 PS2_n_IBUF，产生键盘接收数据中断(使能的情况下)从而模拟 PS2 控制器从键盘收到数据。
0xD3	写鼠标缓冲器	向命令寄存器 PS2_n_CR 写入命令字 0Xd3，则可将数据写到输出缓冲器 PS2_n_OBUF，随后这个数据会自动流入输入缓冲器 PS2_n_IBUF，产生鼠标接收数据中断(使能的情况下)从而模拟 PS2

		控制器从鼠标收到数据。
0xE0	控制器自测试	向命令寄存器 PS2_n_CR 写入命令字 0xE0，则输入缓冲器 PS2_n_IBUF 返回 0xFF。

16.4.4 预定标和分频寄存器设置

为了使 PS2 控制器发送的数据稳定，当 PS2 控制器检测到时钟信号 CLK 的下降沿后会延迟 5us 发送数据。5us 的时间由预定标寄存器 PS2_n_CPSR 中的值来给定，具体值的计算方法为：

PS2_n_CPSR 值=5us*Fps2_wk_clk

Fps2_wk_clk 为 PS2 控制器工作时钟 ps2_wk_clk 的频率。5us 的时间并不要求十分精确，在 5us 到 10us 之间均可，不能超过 25us。由 PS2 设备决定。

PS2 控制器还通过预定标寄存器 PS2_n_CPSR 和分频寄存器 PS2_n_DVR 共同定义了一个 100us 时间。这段时间用于强制拉低时钟信号 CLK。根据 PS2 协议，当 PS2 控制器要想向 PS2 设备发送数据，先要强制拉低时钟信号 CLK 一段时间，至少需要 100us。PS2_n_DVR 具体值的计算方法为：

PS2_n_DVR 值=100us*Fps2_wk_clk / CPSR 值

100us 的时间并不要求十分精确，不能低于 100us 且不能超过 15ms，由 PS2 设备决定。

16.5 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

16.6 编程指导

当前版本用户手册暂不提供详细编程指导。

17 7816 控制器

17.1 概述

7816 接口控制器主要完成 GSC3281 芯片与 SIM 卡的接口功能，GSC3281 中有两组 7816 接口控制器，这两组 7816 控制器在功能上完全相同。它将 CPU 通过总线发来的数据保存在发送 FIFO 中，并以符合 ISO7816-3 标准的信号形式发送到外部端口，同时接收 SIM 卡返回的数据，存于内部的接收 FIFO 中，CPU 读取接收 FIFO 中的数据进行解析。同时，本模块还对 SIM 卡提供时钟、复位和电源的控制，用户可以通过此接口向 SIM 卡发出复位信号，接收 ATR 响应，规定接口速率，发送命令，接收应答。

17.2 引脚描述

两组 7816 接口的引脚如表 17-1 和表 17-2 所示。sim0_io 和 sim1_io 在芯片外部需要上拉。

表 17-1 7816 接口控制器 0 的引脚描述

名称	类型	上拉 下拉	描述
sim0_rst_n	O	-	7816 为 sim0 卡提供的复位信号
sim0_clk	O	-	7816 为 sim0 卡提供的时钟信号
sim0_vccen	O	-	7816 为 sim0 卡提供电源开关
sim0_io	B	上拉	7816 控制器与 sim0 卡之间的数据线

表 17-2 7816 接口控制器 1 的引脚描述

名称	类型	位宽	描述
sim1_rst_n	O	-	7816 为 sim1 卡提供的复位信号
sim1_clk	O	-	7816 为 sim1 卡提供的时钟信号
sim1_vccen	O	-	7816 为 sim1 卡提供电源开关
sim1_io	B	上拉	7816 控制器与 sim1 卡之间的数据线

17.3 功能说明

17.3.1 功能特点

1. 支持 ISO7816-3 协议
2. 异步半双工模式
3. 支持控制线电压为 3V
4. 支持 T=0 协议，不支持 T=1 协议
5. 卡激活（冷复位），热复位，卡释放

6. 为卡提供可控时钟
7. 提供可控波特率
8. 支持正向模式和反向模式
9. 支持奇偶校验
10. 支持自动重传

17.3.2 功能说明

7816 接口控制器的总体架构如图 17-1 所示。

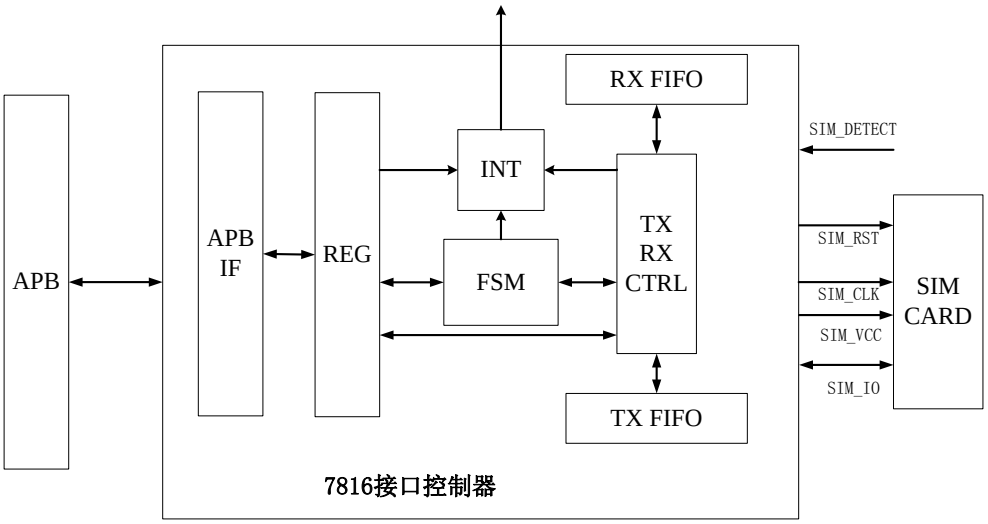


图 17-1 7816 接口总体架构

如图所示，7816 接口控制器为 SIM 卡提供复位信号 SIM_RST 和时钟信号 SIM_CLK，通过数据线 SIM_IO 与 SIM 卡进行数据通信，SIM_IO 是双向的数据线。接收到的数据存入接收 FIFO，而发送 FIFO 中的数据将发送给 SIM 卡。数据的发送接收将在下文中详细介绍。

17.3.3 字符帧

7816 接口控制器与 SIM 卡通信所采用的字符帧结构如图 17-2 所示，通过一根数据线来发送和接收数据。一个字符由 10 个连续的时间段组成。一个时间段即是一个 ETU，在 17.3.4 节中会对 ETU 进行说明。第一个 ETU 为开始位，处于低电平。b1~b8 这 8 个 ETU 运送一个字节，即 8 个比特位。最后一个 ETU 为奇偶校验位 b9。在每个字符中，如果在某个 ETU 结束时状态发生改变，允许有 0.2ETU 的延迟，详见 ISO7816-3 协议。

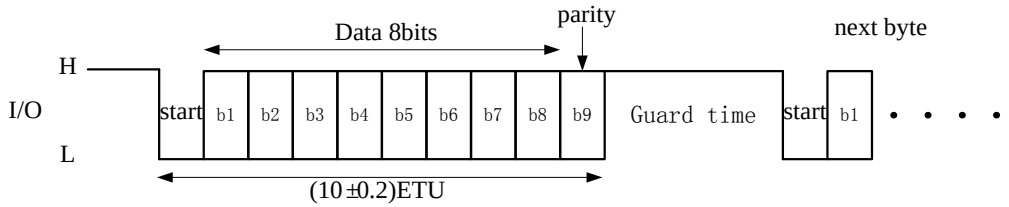


图 17-2 字符帧结构

7816 接口控制器从 SIM 卡接收到的第 1 个字符包含的信息对后续通信中的字符进行了

编码约定。包括 b1~b9 这 9 个比特的编码电平，高电平编码为 1，低电平编码为 0，或是高电平编码为 0，低电平编码为 1。同时对字符的方向也进行了约定，正向情况下，b1 为 LSB，b9 为 MSB，反向情况下，b1 为 MSB，b9 为 LSB。

解析第一个字符获知字符的编码约定后，软件通过 CTRL0 寄存器来进行相关配置，配置比特位 SENSE 可以设定编码电平，配置比特位 ORDER 来设置字符方向。还可以通过比特位 PARITY 来配置奇偶校验方式，不过一般情况下都采用偶校验。

更为详细的字符帧结构的相关介绍，请参见 ISO7816-3 协议。

17.3.4 波特率

7816 接口控制器的内部工作时钟为 sci_clk，而 7816 接口供给 SIM 卡的时钟信号为 sim_clk，它与 sci_clk 同频同相。图 17-2 中一个比特持续的时间即为 ETU，在默认情况下，1 ETU=(372*sci_clk_period)。从式中可以得知，ETU 持续 372 个时钟周期。

根据 ISO7816-3 协议，1 ETU=(F/D)* sci_clk_period，其中 F 和 D 分别是速率转换因子和波特率调整因子。(F/D)在复位应答期间为 372，而复位应答结束后，7816 控制器可以得知 F 和 D 的值，进而计算出(F/D)的值，将该值写入 BAUDR 寄存器即可。

17.3.5 保护间隔

如 17.3.3 节中的图 17-2 所示，连续两个字符间是有一段保护时间的，ISO7816-3 协议中规定保护时间最小为 2 个 ETU。如果在这 2 个 ETU 的保护间隔内，7816 接口接收到第二个字符的 start 位，意味着在保护时间内接收到一个低电平，表示字符帧结构出现错误，此时中断 FRAMERR 置位。

17.3.6 字符间额外保护时间(CGT)

在 7816 接口控制器与 SIM 卡的通信过程中，连续两个字符帧之间的最小保护间隔为 2 个 ETU，除此之外，还可以设置额外的保护时间。

ISO7816-3 协议中定义字符间的额外保护时间 CGT 如图 17-3 所示。寄存器 CGTR 与之相对应，软件可进行相关配置。

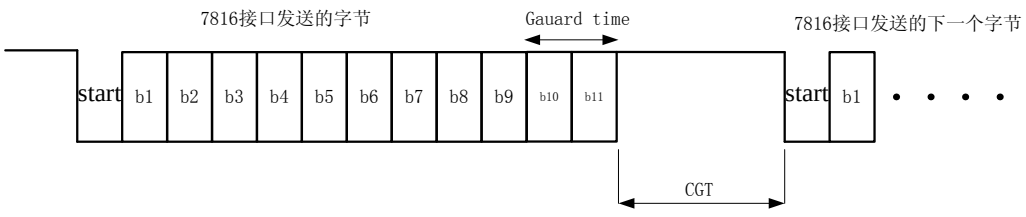


图 17-3 字符间额外保护时间

17.3.7 块等待时间(BWT)

ISO7816-3 协议中定义的块等待时间 BWT(ISO)：7816 接口控制器传给 SIM 卡的最后一个字符的起始沿到 SIM 卡回复 7816 接口的第一个字符的起始沿之间的最大延迟长度，如图 17-4

所示。

而在 GSC3281 中，将块等待时间的开始时刻向后推移 12 个 ETU，即 BWT(GSC3281)，如图 17-4 所示。

需要注意的是，寄存器 BWTR 与图中 BWT(GSC3281)相对应，软件可进行相应配置。如果块等待时间超过了 BWTR 寄存器的配置值，则会引起中断 BWTOUT 置位。

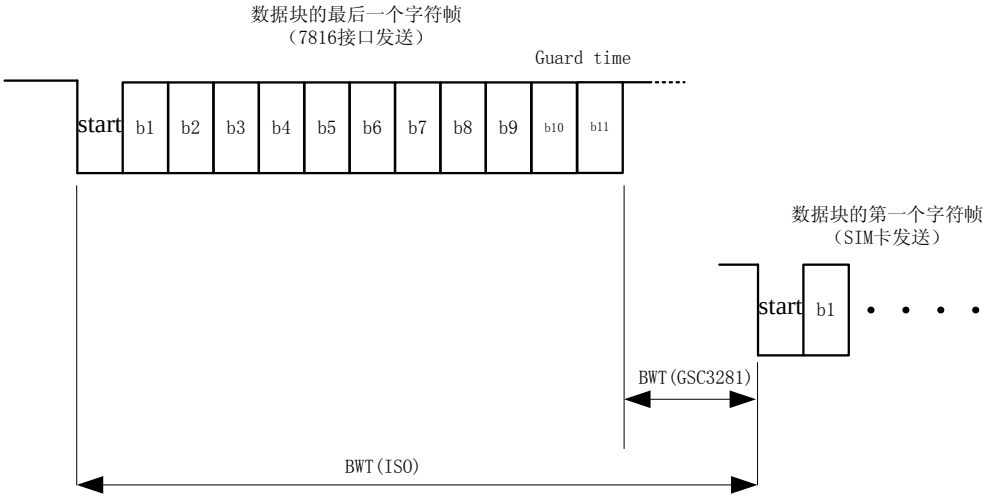


图 17-4 块保护时间

17.3.8 字符间等待时间(CWT)

ISO7816-3 协议中定义的字符等待时间 CWT(ISO)：SIM 卡发送的两个连续字符之间的时间间隔，如图 17-5 所示。

而在 GSC3281 中，将字符等待时间的起始沿向后推移 12 个 ETU，即 CWT(GSC3281)，如图 17-5 所示。

需要注意的是，寄存器 CWTR 与图中 CWT(GSC3281)相对应，软件可以进行相应配置。如果字符等待时间超过了寄存器 CWTR 的配置值，中断 CWTOUT 置位。

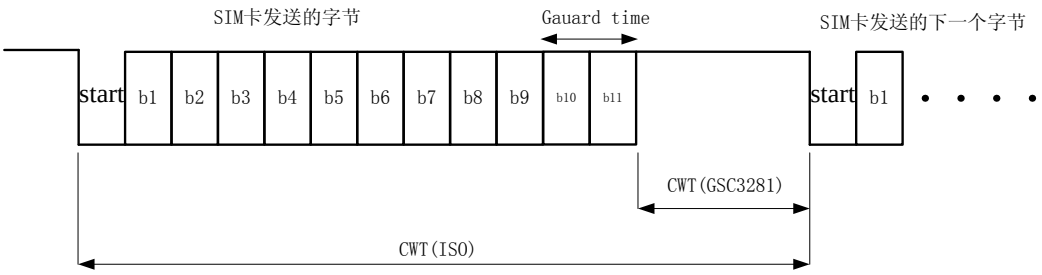


图 17-5 字符间等待时间

17.3.9 块保护时间(BGT)

ISO7816-3 协议中定义的块保护时间 BGT(ISO)：7816 接口控制器接收到的最后一个字符的起始沿到发送给 SIM 卡的第一个字符的起始沿之间的时间间隔，如图 17-6 所示。

而在 GSC3281 中，将块保护时间的起始沿向后推移 12 个 ETU，即 BGT(GSC3281)，如图

17-6 所示。

需要注意的是，寄存器 BGTR 与 BGT(GSC3281)相对应，软件可以进行相应配置。

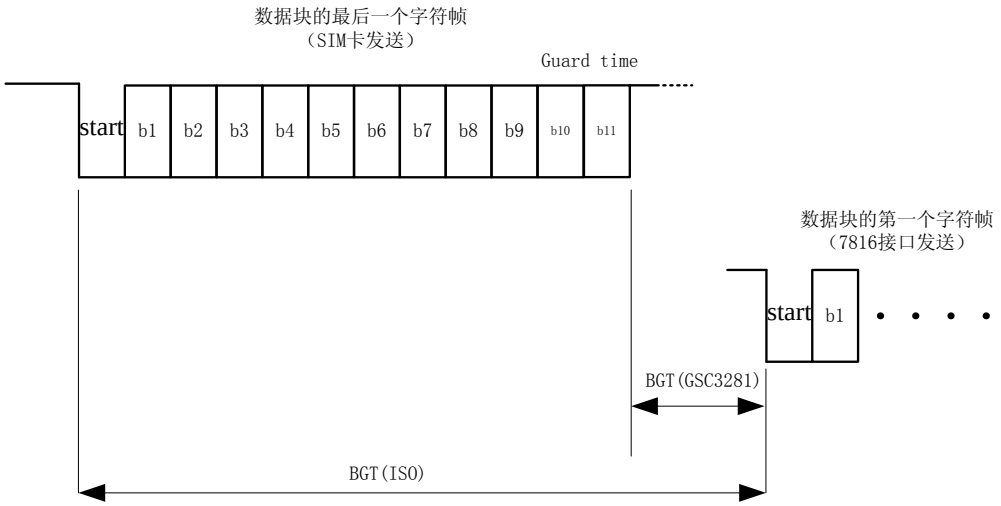


图 17-6 块保护时间

17.3.10 差错信号与字符重传

ISO7816-3 协议中规定，当奇偶校验不正确时，在(10.5±0.2)ETU（接收方时间）时，接收方应通过将 I/O 置为低，持续最小 1ETU，最大为 2ETU，这就是接收方的差错信号。然后，发送方应在检测到差错信号后的至少 2 个 ETU 延迟之后，重新发送该字符，接收方也应做好接收准备。

如图 17-7 所示，当发送完一个字符后，如果接收方在字符保护时间内发送差错信号，则发送方需要重新发送该字符。所以在 7816 接口与 SIM 卡的通信中，当 7816 接口控制器作为发送方时，接收到差错信号，需要重新发送字符。而 7816 接口控制器作为接收方时，当接收到的字符出现奇偶校验错误时，则需要重新接收 SIM 卡发送的重复字符。在 GSC3281 中，可以通过配置寄存器 CTRL0 中的 TX_RPTL 和 RX_RPTL 来设置重发次数和重收次数。

如果将 TX_RPTL 设置为 1，7816 控制器只支持一次重发，即当 7816 控制器发送字符后，接收到 SIM 卡发送的差错信号后，会重新发送上次的字符，如果 SIM 卡仍然发送差错信号，则中断 TXERR 置位。如果将 TX_RPTL 设置为 0，则意味着不允许重传，即 7816 控制器发送字符后，如果检测到差错信号，则立即会将中断 TXERR 置位。

如果将 RX_RPTL 设置为 1，7816 控制器只支持一次重收，即当 7816 控制器接收到的字符有奇偶校验错误时，会发送差错信号，这时 SIM 卡会重新发送字符，如果 7816 控制器接收到的字符仍有差错，则会将中断 RXERR 置位。如果将 RX_RPTL 设置为 0，则意味着不允许重收，即 7816 控制器接收到的字符有奇偶校验错误时，立即会将中断 RXERR 置位。

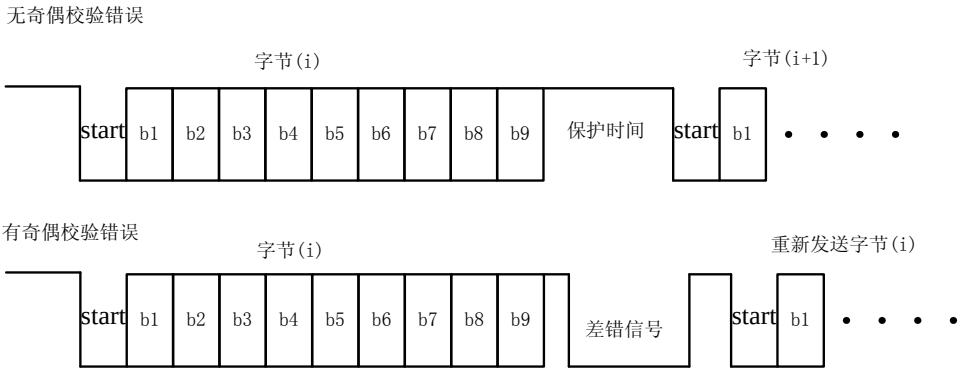


图 17-7 字符重传

17.3.11 FIFO 操作

如图 17-1 所示，7816 接口控制器中有发送 FIFO 和接收 FIFO，地址深度均为 16，位宽均为 8 位。

当 CPU 向发送 FIFO 中写入数据时，控制器会将 FIFO 中的字节以比特形式串行发送给 SIM 卡。只要发送 FIFO 不空，控制器就会向 SIM 卡发送数据。如果控制器正处于接收状态时，不要向发送 FIFO 写入数据，否则会产生错误。通过配置寄存器 FSR 中的 TX_FTL 来设置发送 FIFO 的将空阈值，如果发送 FIFO 中的数少于或等于该阈值，中断 TXTIDE 置位。如果发送 FIFO 空了，则中断 TEMPT 置位。

当控制器处于接收状态时，控制器会将接收的数据以字节的形式存入接收 FIFO，CPU 可以直接读取。通过配置寄存器 FSR 中的 RX_FTL 来设置接收 FIFO 的将满阈值，如果接收 FIFO 中的数等于或多于该阈值 ，中断 RXTIDE 置位。如果接收 FIFO 满，则中断 RFULL 置位。

17.3.12 SIM 卡的热插拔

在具体的产品应用和设计中，如果系统只有在断电后才能插拔 SIM 卡，这种情况 7816 控制器可以不支持 SIM 卡的热插拔功能，否则的话，就必须支持热插拔功能，因为系统在没有断电的情况下就直接插拔 SIM 卡可能会导致 SIM 卡的烧毁。GSC3281 中的 7816 控制器支持 SIM 卡的热插拔，这一功能是通过软件来实现。

当插入 SIM 卡时，产生 GPIO 中断，此时软件配置 CTRL0 寄存器和 CTRL1 寄存器的相关比特位使 7816 控制器激活 SIM 卡。

当拔出 SIM 卡时，产生 GPIO 中断，此时软件配置 CTRL0 寄存器和 CTRL1 寄存器的相关比特位使 7816 控制器释放 SIM 卡。

由于 SIM 卡的热插拔是通过软件来实现，会在软件编程部分对热插拔有更为详尽的说明。

17.3.13 7816 的接口时序

SIM 卡的时钟和复位信号由 7816 接口模块提供。当 SIM 卡的时钟在 1MHz 和 5MHz 之间，SIM 卡能正常工作。

SIM 卡的冷复位时序如图 17-8 所示。

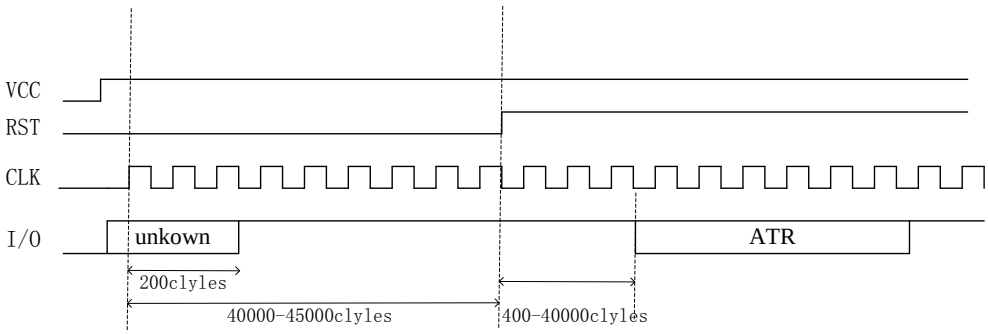


图 17-8 SIM 卡的冷复位时序

触点接通后，7816 接口控制器将发出一个冷复位信号，并从 SIM 卡得到一个复位应答(Answer To Reset, ATR)，过程如下（可参考图 17-8）：

1. VCC 电平为高且稳定后，终端给卡时钟信号 CLK
2. CLK 产生后的 200 cycles 内，终端将 I/O 线拉高置为高电平，则 I/O 线变为接收状态
3. 终端将 RST 置为低电平，时钟产生后的 40000~45000 cycles 内将其置为高电平
4. SIM 卡应在 RST 变为高电平后的 400~40000 cycles 内发送 ATR 信号
5. 如果 SIM 卡未返回 ATR 信号，终端则会进行热复位

SIM 卡的热复位时序如图 17-9 所示。

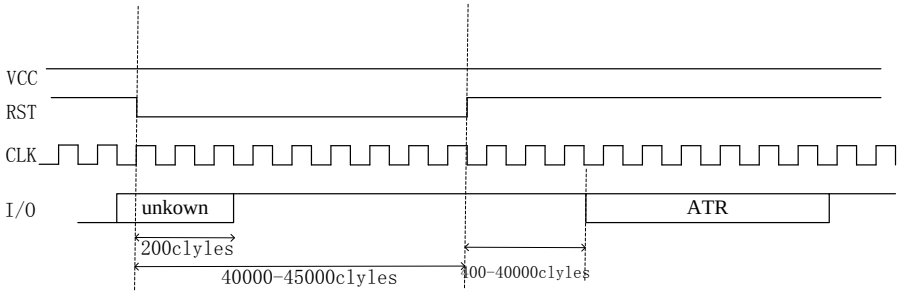


图 17-9 SIM 卡热复位时序

如果 7816 接口控制器收到的 ATR 不符合要求或是未收到 ATR 信号时，终端将启动一个热复位，并试图从 SIM 卡获得 ATR 信号，过程如下（可参考图 17-9）：

1. 终端保持 VCC 一直为高，保持 CLK
2. 将 RST 信号置低
3. 在 RST 置低后的 200 个 cycles 内，终端必须将 I/O 置高，即设为接收状态
4. 在 RST 置低后的 40000~45000 个 cycles 内，将 RST 置高
5. SIM 卡需在 RST 置高后的 400~40000 cycles 内，给终端返回 ATR 信号
6. 如果卡未返回 ATR 信号，则需要对 SIM 卡进行释放

SIM 卡的释放时序如图 17-10 所示。

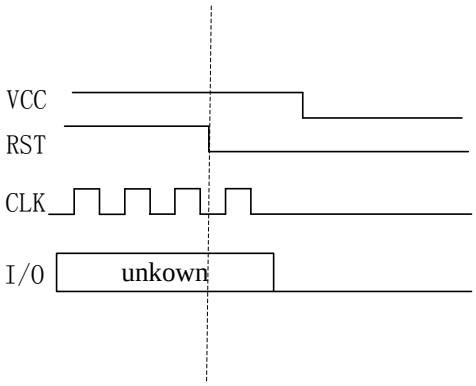


图 17-10 SIM 卡释放时序

7816 接口控制器与 SIM 卡的通信无论是正常结束，还是异常结束，都要对 SIM 卡进行释放。卡的释放过程如下（可参考图 17-10）：

1. 首先将 RST 置为低电平
2. 在 VCC 断电之前将 CLK 和 I/O 置为低电平
3. 物理断开 SIM 卡时，必须将 VCC 断电

17.3.14 复位应答

如图 17-8 和图 17-9 所示，7816 接口控制器对卡进行冷复位或是热复位之后，等待 SIM 卡发送的复位应答（ATR），如果长时间没有接收到 ATR，中断 ATRSTOUT 置位。如果接收到的 ATR 中的字节数超过 32 个，则中断 ATRSTOUT 置位。

17.4 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

17.5 编程指导

当前版本用户手册暂不提供详细编程指导。

18 矩阵键盘接口

本章节从使用角度介绍芯片 GSC3281 中集成的 4*4 矩阵式键盘控制器的功能特性和工作原理，从编程角度说明其内部寄存器的含义和配置方法。

18.1 概述

本矩阵键盘接口用于连接 4*4 的矩阵式键盘。矩阵式键盘由列线和行线组成，列线上接有上拉电阻（本接口内部已经接有上拉电阻，外部可以不接），按键位于行列线交叉点上，4*4 的行列线可以构成一个有 16 个按键的键盘，如图 18-1 所示。行列线分别连接到按键开关的两端。无按键动作时，所有列线均处于高电平状态；当有键按下时，按键所在的列线与行线导通，其电平状态将由此行线电平决定：行线电平如果为低，列线电平为低，行线电平如果为高，则列线电平亦为高。

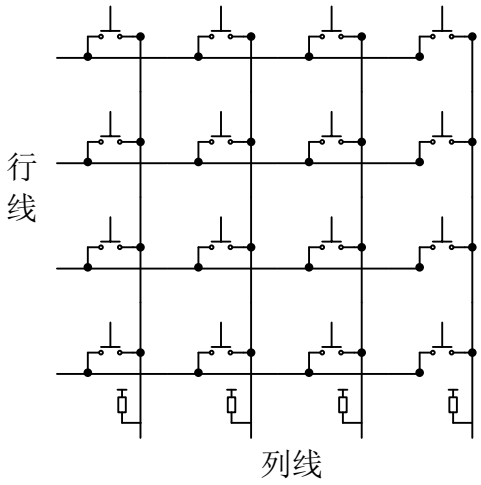


图 18-1 矩阵键盘原理图

本矩阵键盘接口对按键的识别步骤如下：

1．将所有行线均置为低电平，然后检查各列线电平是否为低，如果有列线为低，则说明该列有键被按下，否则认为无键被按下。

2．如果某列有键被按下，通过扫描确定哪一行的键被按下。逐行置低电平，并置其余各行为高电平，同时检查各列线电平的变化，如果有列线变为低电平，则可确定此列此行交叉点处有键被按下。

18.2 引脚描述

表 18-1 矩阵键盘接口引脚描述

名称	类型	上拉 下拉	功能描述
col0	I	U[注 1]	矩阵键盘列输入第 0 位
col1	I	U	矩阵键盘列输入第 1 位
col2	I	U	矩阵键盘列输入第 2 位

col3	I	U	矩阵键盘列输入第 3 位
row0	O		矩阵键盘行输出第 0 位
row1	O		矩阵键盘行输出第 1 位
row2	O		矩阵键盘行输出第 2 位
row3	O		矩阵键盘行输出第 3 位

[注 1]：I 表示输入，O 表示输出，U 表示内部上拉。

18.3 功能与原理说明

矩阵键盘接口模块用于连接 4*4 的矩阵键盘，通过对矩阵键盘的扫描，去抖动后获得键值，并在按键按下，改变和释放时以中断形式通知 CPU 来获取按键状态。该矩阵键盘接口能够检测任意一个或两个同时按下的按键值，对三个或三个以上的同时按键，则产生多键按键错误中断（如不屏蔽）；支持低功耗模式。低功耗模式下，任意按键都可以产生系统中断来通知 CPU。所有矩阵键盘的功能必须在寄存器 KP_EN 为 1 即使能矩阵键盘时才有效。

本矩阵键盘接口的特点如下：

- 支持硬件扫描键盘
- 支持 4*4, 3*4, 3*3 的矩阵情形
- 低功耗模式下任意按键都可以产生系统中断来通知 CPU
- 支持去抖动功能
- 能够检测一个或两个同时按下的按键

本矩阵接盘接口支持 4*4, 3*4, 3*3 等矩阵情形，在使用 3*4 或 3*3 的矩阵键盘时，将多余的行列线悬空即可。

矩阵键盘接口模块主要包括键盘扫描控制（keypad_scan_ctrl）、键盘扫描（keypad_scanner）和寄存器逻辑（keypad_register_logic）三个部分，如图 18-2 所示。其中，寄存器逻辑包括本模块的寄存器和 APB 总线访问寄存器的逻辑，寄存器的描述在后面的章节会详细阐述。键盘扫描控制模块用于完成按键的去抖动，产生键盘扫描使能信号，将键盘扫描结果编码后写到寄存器并产生键盘接口模块中断。键盘扫描模块完成键盘的即时扫描功能。

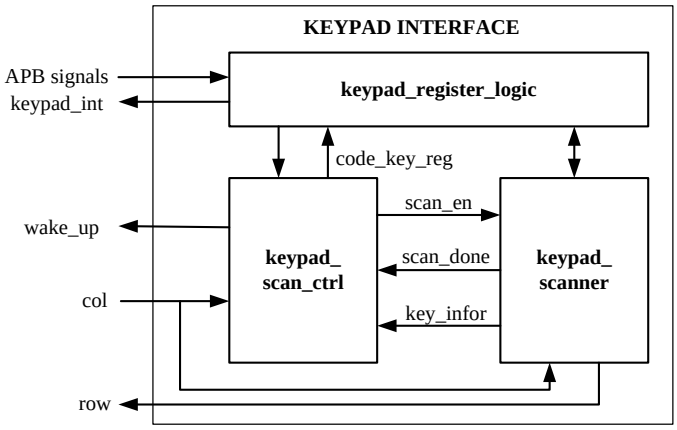


图 18-2 矩阵键盘接口模块结构

在介绍键盘扫描工作原理之前，先插入介绍本接口对 4*4 矩阵键盘的按键编码。键盘的 16 个按键采用 4-bit 键值编码，用十六进制数 0x0-0xF 来表示每个按键的键值（KEY_VALUE）。按键的键值与行列的对应关系如表 18-2 所示。

表 18-2 键值与行列对应关系表

	col0	col1	col2	col3
row3	0x0	0x1	0x2	0x3
row2	0x4	0x5	0x6	0x7
row1	0x8	0x9	0xA	0xB
row0	0xC	0xD	0xE	0xF

每个按键有 2-bit 状态标志信息：“01”表示按下；“10”表示松开；其它表示无动作。

键盘扫描模块完成即时扫描功能。起始状态下，输出全低电平的行信号，通过输入的列信号是否有低电平来判断是否有按键按下。当检测到有按键按下时，触发扫描动作，即按照一定的步长（寄存器 KP_CTRL 中字段 SCAN_STEP）周期性地产生一组行输出，然后对列输入信号进行高低电平检测，从而判断有无按键被按下及哪个键被按下。每完成一轮扫描，键盘扫描模块都将本次扫描到的按键信息传递给键盘扫描控制模块。

键盘扫描控制模块内部有一个计数器，每次收到按键信息后都将新的按键信息与上一次的（保存在此模块内部的按键信息寄存器中）作比较，如果相同，就将计数器加 1；而一旦读到不同的按键信息则立即将计数器清 0，同时更新内部按键信息寄存器。如果计数器的值达到寄存器 KP_CTRL 中字段 DEBOUNCE_LIMIT 的值时，表示连续 DEBOUNCE_LIMIT 次读到同一个按键信息，此时认为已经读到了有效的按键信息，同时产生一个有效按键中断（如不屏蔽）。当有三个或以上的按键被检测到同时按下后，键盘扫描模块会立即产生一个多键按下错误中断（如不屏蔽）。若当前中断是所有按键都释放引起，则通知键盘扫描模块停止扫描。

按键扫描所采用的时钟由 APB 总线时钟分频而得，时钟的配置请参考系统控制（SYS_CTRL）模块中的相关寄存器。配置好扫描时钟的频率后，可以根据寄存器 KP_CTRL 中的描述对扫描步长做配置。

矩阵键盘接口模块支持低功耗工作模式。在正常模式下，要使矩阵键盘接口进入低功耗模式，必须先配置 KP_CTRL 寄存器的 LPM 位为 1，然后再通过配置系统控制模块内的寄存器来关闭矩阵键盘接口的工作时钟，使之进入低功耗模式；如果关闭时钟之前，LPM 位没有被置 1，那么键盘接口模块将无法向系统产生 wake_up 中断而被唤醒。系统处于低功耗模式时，按动矩阵键盘的任意按键都能使本模块向系统产生 wake_up 中断而被唤醒。

本模块有一个中断寄存器，用以保存两个中断：多键被按下的错误中断，有效按键中断。这两个中断都是通过写 1 清除，都有独立的屏蔽位来屏蔽。除此之外，本模块还可以在低功耗模式下向系统中断控制器产生前文所述的 wake_up 中断。

18.4 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

18.5 编程指导

当前版本用户手册暂不提供详细编程指导。

19 ADC 控制器

19.1 概述

ADC 控制器通过 SPI 接口挂在 GSC3281 的 SPI0 总线上。

19.2 引脚描述

表 19-1 ADC 控制器引脚描述

ADC 控制器接口信号			
名称	类型	上拉 下拉	功能描述
XP	I		外部 SAR-ADC 输入通道 0
YP	I		外部 SAR-ADC 输入通道 2
XN	I		外部 SAR-ADC 输入通道 1
YN	I		外部 SAR-ADC 输入通道 3
PBAT	I		电池输入(3.3V-5V)
avdd1	I		模拟电源 1(3.3V)
avss1	I		模拟地 1
avdd2	I		模拟电源 2(3.3V)
avss2	I		模拟地 2

19.3 功能特性

GSC3281 片上集成的 ADC 控制器主要有以下特性：

- 通过标准 SPI 接口传输命令和数据；
- 最大分辨率为 12 位；
- 输入最大的 SPI 时钟是 6MHz，对应最大采样率 120Ksps；
- 可以当 4 路 ADC 输入；
- 支持低功耗模式；
- 支持软件复位；

在使用 ADC 控制器时，需要首先使能系统控制模块中的 ADC 使能寄存器，然后才能通过 SPI0 接口操作 ADC 控制器。使能 ADC 时，外部 SPI 接口仍然可以接其它设备，外部设备片选通过 GPIO 实现。ADC 控制器通过 SPI0 连接到系统中，数据传输模式是 SPI 模式 3。

管脚 PBAT 用于测量电源电压，测量命令是 0x3（0011）。

控制器支持低功耗模式，也就是挂起模式，当控制器接收到命令 0xE（1110）时，控制器将进入挂起状态，此时，控制器数字部分功耗大约是 16uA（1.2V），模拟部分功耗大约是 18uA（3.3V），而正常工作状态下功耗为数字部分 200uA（1.2V），模拟部分功耗大约是 3mA（3.3V）。软件复位控制器也能使控制器进入挂起状态。

ADC 控制器通过 SPI0 集成在 GSC3281 中，软件上可以通过编程系统控制单元中复位管

理逻辑来复位 SPI0 控制器及 ADC 控制器。

19.3.1 ADC 操作

当 ADC 控制器四根引脚连接到外部设备上时，CPU 可以直接发出相关的通道测量命令进行测量，也可以配置 SPI0 控制器让 ADC 自动采样每个通道都有各自的测量命令，并且一个时刻只能对一个通道进行测量，也就是当一个通道在测量，其它通道只能等待正在被测通道测量结束，CPU 才能发其它通道的测量命令开始下一个通道的测量。用户不能在某个通道正在测量时，又发出其它通道的测量命令。

19.3.2 VBAT 测量

用户可以通过发命令 0x3 测量管脚 PBAT 上的电压值，这个功能可以用作监测电池电压值，PBAT 的电压通过内部分压之后经过 ADC 转换，得到有效位数为 8 位的电压值。

如图 19-1 所示，当 PBAT 上的电压值 VBAT 为 4.2V 时，则 ADC 测量得到的值应该为((4.2/3)/2.5)*256，用户通过测量值可以计算出 VBAT 的值。PBAT 管脚也可以当做一路模拟输入使用，但和其它的四路模拟输入有区别，PBAT 的分辨率为 8 位，参考电压为 2.5V，而 SADCIN0~3 的分辨率为 12 位，参考电压为 3.3V。

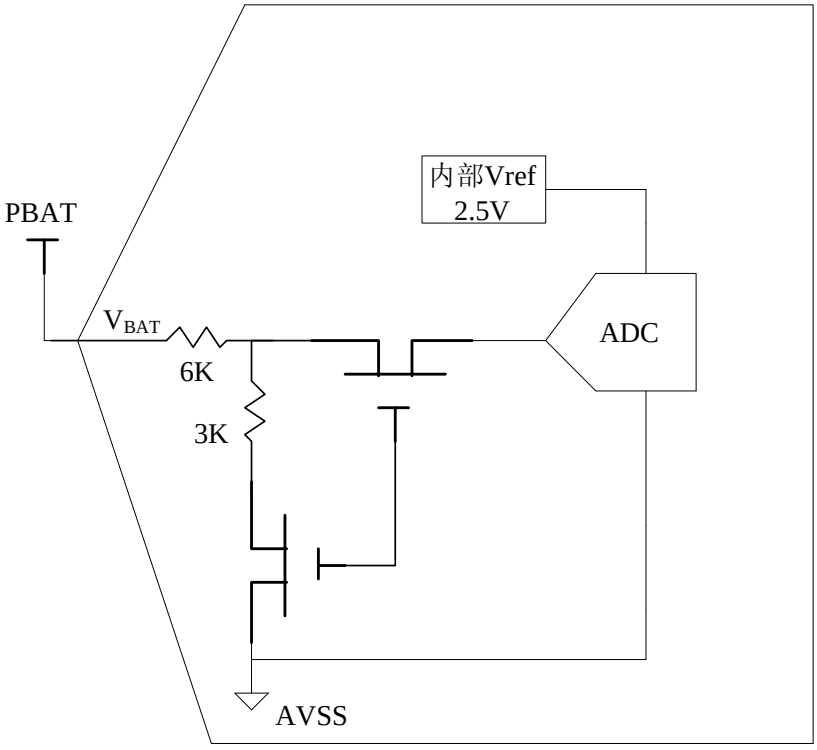


图 19-1 VBAT 测量功能图

19.3.3 命令周期

当 CPU 发出 ADC 测量命令时，在命令发到 SPI_DIN 线上的同时，控制器在 SPI_DOUT 线上返回了状态。

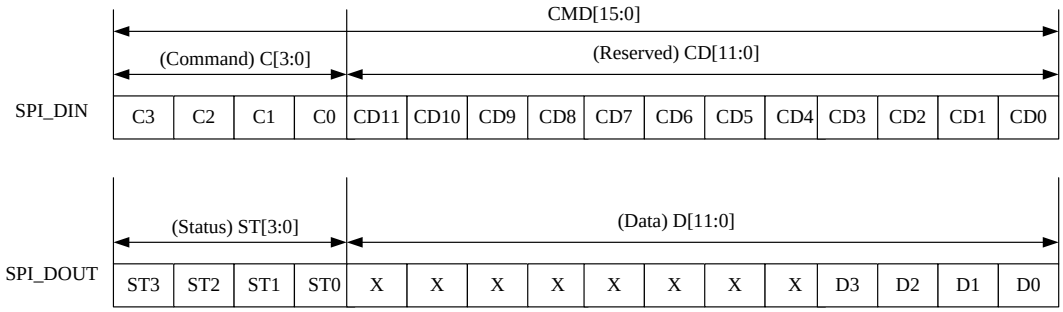


图 19-2 ADC 命令周期

图 19-2 中 SPI_DIN 数据线上发的是命令，命令 CMD[15:0]的高 4 位即 C[3:0]表示是 ADC 的测量命令，后面 12 位是参数，如果命令不带参数，后面 12 位全部为 0。

如图 19-2 所示，CPU 往 SPI_DIN 线上发送命令的同时，SPI_DOUT 线返回了控制器的状态。ST[3:0]表示当前控制器的状态。状态 ST 为 0x0 时，表示控制器正在传数据，状态 ST 为 0x15 时表示控制器正忙，状态 ST 为 0x8 时，表示控制器处于空闲状态（此时中断产生是使能的，模数转换也是使能的），状态 ST 为 0x9 时，表示控制器处于挂起状态（中断产生是使能的，但模数转换不使能）。除了这四种状态之外的任何其它状态，都表示控制器是处于不正常状态，需要进行软件复位。

图 19-2 中 SPI_DOUT 上的低 4 位即 D[3:0]是命令响应，应该与命令 C[3:0]相等，如果 D[3:0]和 C[3:0]不相等，表明控制器接受命令出现了错误，需要复位且重发命令。SPI_DOUT 上的 D[11:4]为无意义位，用户不用关心。

在空闲状态时，所有命令都能接受，在挂起状态时，仅仅只有 0x0（0000）和 0xE（1110）命令能接受。

19.3.4 数据周期

控制器通过 SPI_DOUT 线将数据返回给 CPU，返回的数据一共 16 位，低 12 位是数据，高 4 位是标志位。CPU 首先发测量命令，当命令正确接收后（命令返回状态正确），CPU 需要通过 SPI_DIN 线发 NOP 命令，在发 NOP 命令的同时，在 SPI_DOUT 线上取回数据，每次取回的数据，CPU 都要判断标志位来决定低 12 位是否是有效数据。

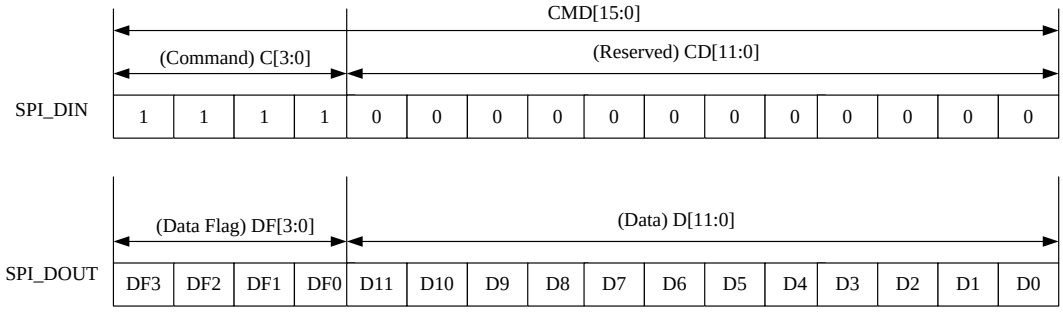


图 19-3 ADC 数据周期

数据线 SPI_DOUT 上的高 4 位是数据标志位 DF[3:0]，表示当前数据即数据线上后面 12 位的数据是否是有效的数据。DF 等于 0x0 时，是有效数据，DF 等于 0x15 时，表示控制器正忙，还在处理上个测量命令，此时数据线上的数据是无效的，可忽略。DF 等于其他数值时，控制器处于非正常状态，需要软件复位。

19.4 时钟及采样率

ADC 控制器是通过 SPI 接口集成到 GSC3281 中的，SPI 接口的时钟 SPI_CLK 是由 SPI0 控制器产生的。ADC 控制器的输入 SPI 时钟 SPI_CLK 的最大频率为 6MHz，当 SPI_CLK 不被分频时，每次测量得到有效数据的时间大约为 50 个 SPI 时钟周期，所以最大的采样率大约是 120Ksps（SPI_CLK/50=6MHz/50）。

ADC 控制器提供一个分频命令（1000），在使用过程中如果不需要以 SPI_CLK 的频率进行操作时，可以通过分频命令（1000）对 SPI_CLK 进行分频，最多能进行 4 分频，详细请参考命令说明。ADC 控制器如果被软件复位过后，控制器进入挂起状态，分频配置也将被复位，软件需要重新发分频命令进行分频操作。如果软件通过命令（1110）让控制器进入挂起状态时，不需要重新进行分频。GSC3281 控制器的 SPI0 控制器的输出时钟也是可配置的，实际使用过程中，软件也许不必对 ADC 控制器的输入时钟 SPI_CLK 进行分频，而是直接配置 SPI0 控制器，让 SPI0 控制器输出一个想要的时钟 SPI_CLK 给 ADC 控制器。

19.5 ADC 自动采样

当 ADC 控制器用于测量 ADC 模拟通道时，可以通过软件发测量命令进行采样，也可以配置 SPI0 控制器的 ADC 自动采样逻辑进行 ADC 自动采样。自动采样时，控制器的初始化及测量操作全部都由 SPI0 控制器的自动采样逻辑完成，不需要 CPU 参与操作 ADC 控制器，详细过程可以参考 SPI 控制器的用户手册。

19.6 软件操作流程

图 19-4 为 ADC 控制器的软件操作流程，控制器复位过后进入挂起状态，挂起状态只能接受 0x0 和 0xE 两种命令。

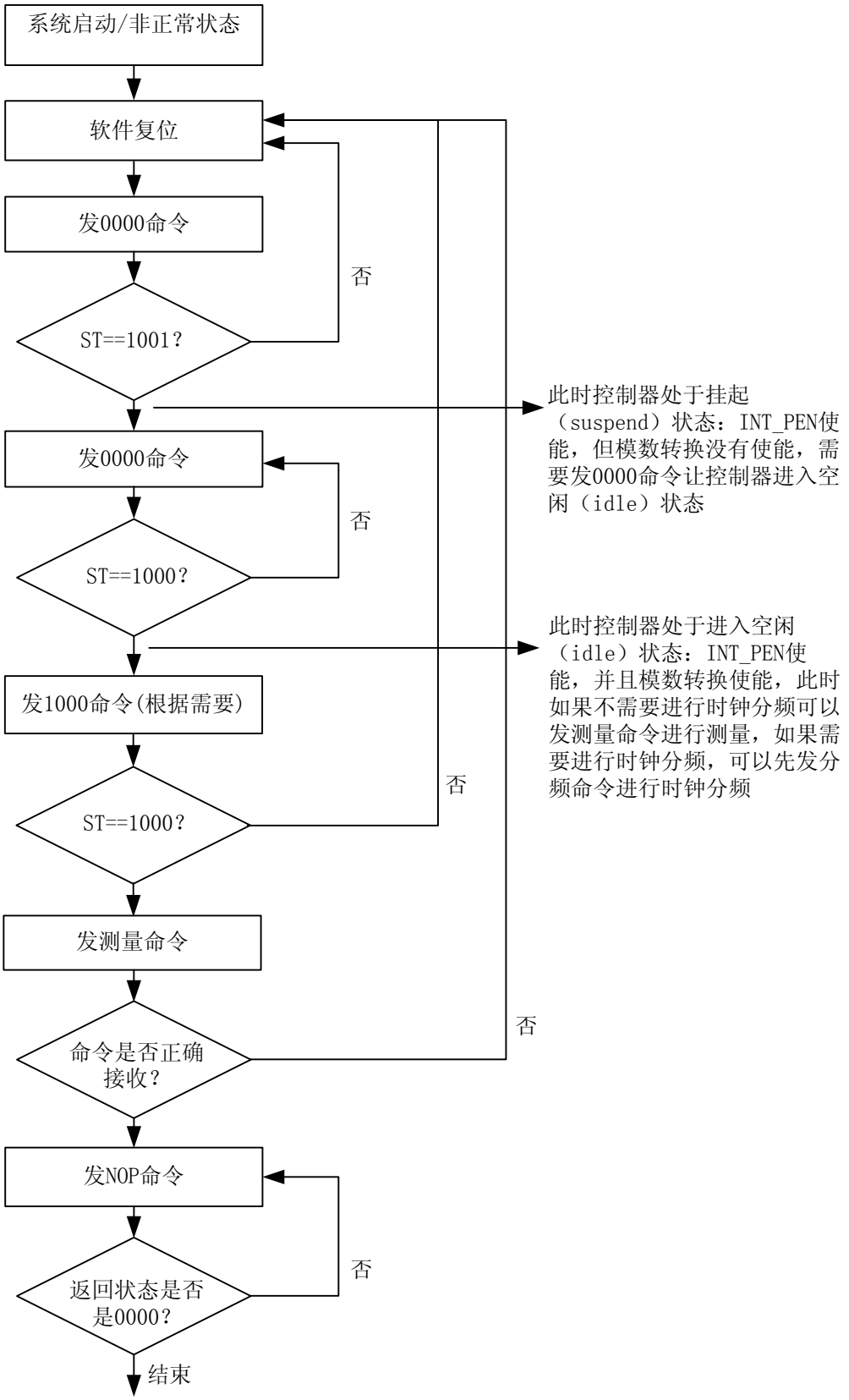


图 19-4 ADC 控制器操作流程图

19.7 控制器命令

表 19-2 ADC 控制器命令

C[3:0][1]	ADC 输入	功能描述
0x0(0000)	-	使控制器进入空闲状态，如果上个状态是挂起状态，需要延迟 100us。 测量命令结束后，控制器也是处于空闲状态。
0x1(0001)	YP	单端测量 X 的位置
0x2(0010)	XP	单端测量 Y 的位置
0x3(0011)	PBAT	单端测量 PBAT 的输入
0x4(0100)	YP	差分格式测量 X 的位置
0x5(0101)	XP	差分格式测量 Y 的位置
0x7(0111)	组合	组合测量相当于发（0110,0100,0101,0110）命令
0x8(1000)	-	分频命令，分频因子需要通过 CMD[1:0]给出 0x0：1 分频 0x1: 2 分频 0x2：3 分频 0x3：4 分频
0x9(1001)	SADCIN0	测量 ADC 通道 0 模拟输入
0xA(1010)	SADCIN1	测量 ADC 通道 1 模拟输入
0xB(1011)	SADCIN2	测量 ADC 通道 2 模拟输入
0xC(1100)	SADCIN3	测量 ADC 通道 3 模拟输入
0xD(1101)	-	读控制器状态
0xE(1110)	-	使控制器进入挂起状态（这也是复位后的默认状态）
0xF(1111)	-	NOP 命令，发完测量命令后，并且命令正确接收，CPU 需要发 NOP 命令，并且判断返回的状态是否是 0x0（0000），是 0x0 即得到有效的数值

注[1]：命令是 4 位，括号里是 4 位完整二进制命令格式。

19.8 典型应用

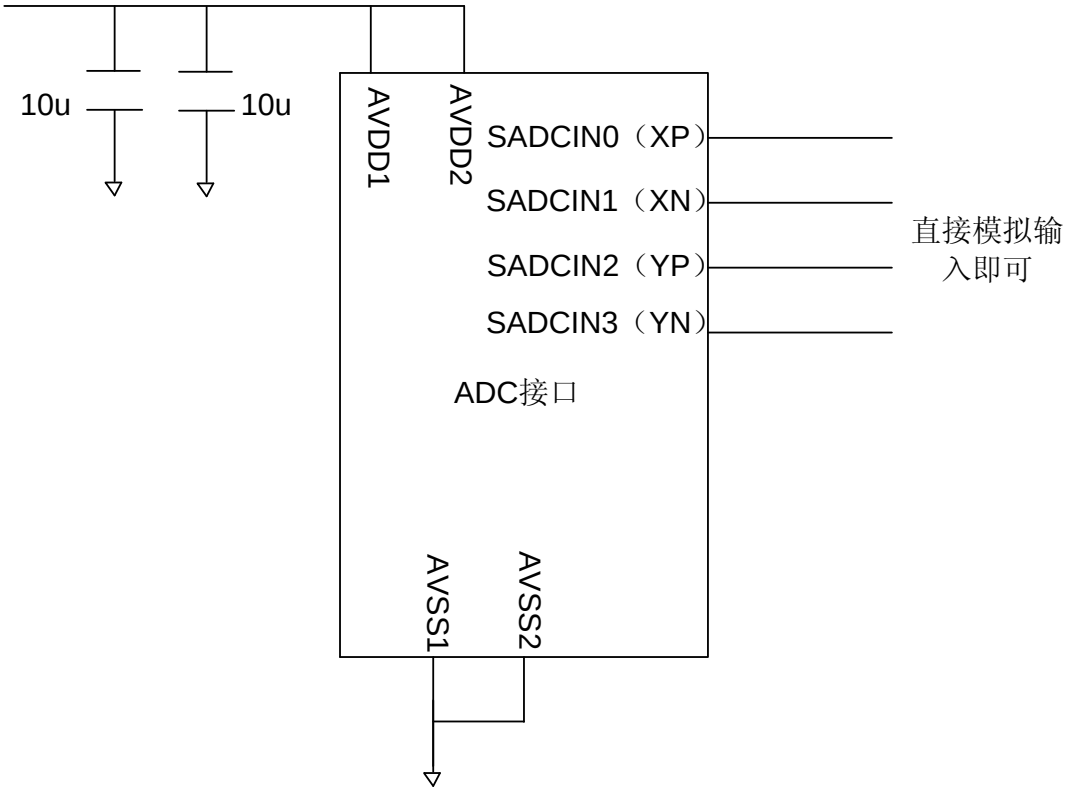


图 19-5 ADC 控制器典型应用

19.9 编程指导

当前版本用户手册暂不提供详细编程指导。

20 PWM 与旋转编码器接口

20.1 概述

脉冲宽度调制(PWM)是一种对模拟信号电平进行数字编码的方法。通过高分辨率计数器的使用，方波的占空比被调制用来对一个具体模拟信号的电平进行编码。PWM 信号仍然是数字的，因为在给定的任何时刻，满幅值的直流供电要么完全有(ON)，要么完全无(OFF)。电压或电流源是以一种通(ON)或断(OFF)的重复脉冲序列被加到模拟负载上去的。通的时候即是直流供电被加到负载上的时候，断的时候即是供电被断开的时候。只要带宽足够，任何模拟值都可以使用 PWM 进行编码。

多数负载(无论是电感性负载还是电容性负载)需要的调制频率高于 10Hz，通常调制频率为 1kHz 到 200kHz 之间。

20.2 引脚描述

表 20-1 PWM 与旋转编码器接口的引脚描述

名称	类型	上拉 下拉	功能描述
pwm_abort	I	-	PWM 快速终止输入
cap0	I	-	PWM 捕获输入 1
cap1	I	-	PWM 捕获输入 2
cap2	I	-	PWM 捕获输入 3
pwmout_0	O	-	PWM 输出信号 0
pwmout_1	O	-	PWM 输出信号 1
pwmout_2	O	-	PWM 输出信号 2
pwmout_3	O	-	PWM 输出信号 3
pwmout_4	O	-	PWM 输出信号 4
pwmout_5	O	-	PWM 输出信号 5

20.3 功能框图

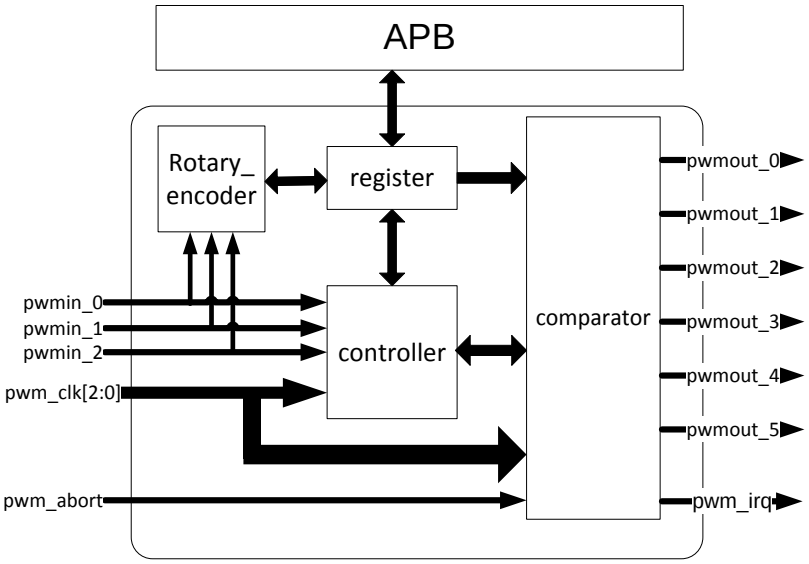


图 20-1 PWM 功能框图

20.4 功能说明

PWM 具有 3 路独立通道。每个通道包含一个 16bit 计数器。每个通道有两路输出，通道 0 有输出 pwmout_0 和 pwmout_1；通道 1 有输出 pwmout_2 和 pwmout_3；通道 2 有输出 pwmout_4 和 pwmout_5。PWM 有 3 路捕获输入信号，可用于计数或者采样。中断采用电平触发，高电平有效，写 1 清零的方式。PWM 可工作在两种模式，PWM 模式和 MCPWM 模式。PWM 控制器还支持增量式旋转编码器，能够由硬件判断旋转方向，并由硬件提供计算转速的参数，提供实时高精度旋转位置信息。

20.4.1 PWM 模式

PWM 模块可作为计数器或者定时器操作。当 PWM 作为计数器工作时，会使用相应的时钟输入对相应的外部输入信号进行采样，每采到上升、下降、上升或者下降时，计数器加 1。但是必须满足输入信号频率要小于工作时钟频率的条件。3 个独立通道实现 6 个单边沿控制或 3 个双边沿控制的 PWM 输出，或两种类型的混合输出。单边沿每次匹配时，输出由无效变为有效。双边沿每次匹配时，同一通道的两个输出均在第一次匹配由无效变为有效，第二次匹配由有效变为无效。每个输出信号的有效、无效代表的电平均可设置。在定时器匹配时，可选择匹配时定时器继续工作、匹配时停止定时器或匹配时复位定时器。定时器和计数器为 16bit，可在 PWM 工作过程中重新设置定时器和计数器。

计数器 PWM_TC 到达界限后复位值为 0，因此实际输出信号的周期拍数为界限寄存器 PWM_LIM 中值加 1。例如，在界限寄存器 PWM_LIM 中写 5，实际输出信号的周期为 5+1 个工作时钟周期。

PWM 模块具有三个捕获输入，可以选择在捕获的上升沿或者下降沿发生捕获事件。输入也可以用作计数器计数。

20.4.1.1 单边沿模式

PWM 模式下的单边沿输出模式，每个通道的输出信号 pwmout 的波形需要一个界限寄存器 PWM_LIM 和一个匹配寄存器 PWM_MR 来确定：

pwmout_0 由 PWM_LIM0 和 PWM_MR0_1 确定；

pwmout_1 由 PWM_LIM0 和 PWM_MR0_2 确定；

pwmout_2 由 PWM_LIM1 和 PWM_MR1_1 确定；

pwmout_3 由 PWM_LIM1 和 PWM_MR1_2 确定；

pwmout_4 由 PWM_LIM2 和 PWM_MR2_1 确定；

pwmout_5 由 PWM_LIM2 和 PWM_MR2_2 确定。

以通道 0 的 pwmout_0 为例，设定 PWM_LIM0 为 2，PWM_MR0_1 为 1，pwmout_0 输出设为高有效，通道 0 作为定时器。波形如图 20-2 所示：

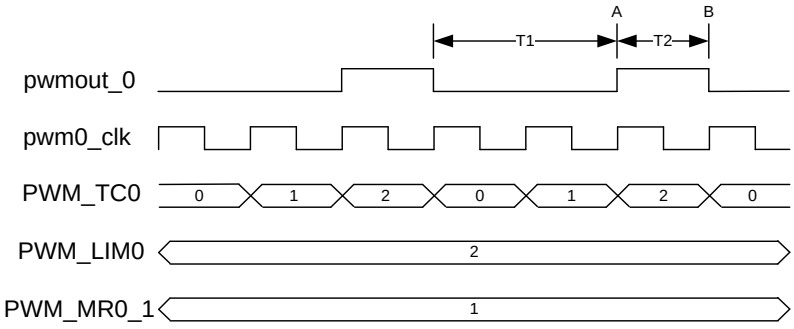


图 20-2 PWM 模式单边沿输出示意图

PWM 开始工作后，计数器 PWM_TC0 递增计数。当计数器 PWM_TC0 中的值等于匹配寄存器 PWM_MR0_1 中的值时 (图中 A 时刻)，发生匹配事件，输出信号 pwmout_0 由无效变为有效。计数器 PWM_TC0 继续递增，当计数器 PWM_TC0 中的值等于界限寄存器 PWM_LIM0 时(图中 B 时刻)，发生到达界限事件，输出信号 pwmout_0 由有效变为无效。计数器 PWM_TC0 复位，重新计数。

pwmout_0 无效持续时间 T1=Tpwm0_clk*(PWM_MR0_1+1)

pwmout_0 有效持续时间 T2= Tpwm0_clk*(PWM_LIM0-PWM_MR0_1)

其中 Tpwm0_clk 为通道 0 的工作时钟 pwm0_clk 周期长度。

20.4.1.2 双边沿模式

PWM 模式下的双边沿输出模式，每个通道的输出信号 pwmout 的波形需要一个界限寄存器 PWM_LIM 和两个匹配寄存器 PWM_MR 共同来确定：

pwmout_0 由 PWM_LIM0 和 PWM_MR0_1、PWM_MR0_2 确定；

pwmout_1 由 PWM_LIM0 和 PWM_MR0_1、PWM_MR0_2 确定；

pwmout_2 由 PWM_LIM1 和 PWM_MR1_1、PWM_MR1_2 确定；

pwmout_3 由 PWM_LIM1 和 PWM_MR1_1、PWM_MR1_2 确定；

pwmout_4 由 PWM_LIM2 和 PWM_MR2_1、PWM_MR2_2 确定；

pwmout_5 由 PWM_LIM2 和 PWM_MR2_1、PWM_MR2_2 确定。

以通道 0 的 pwmout_0、pwmout_1 为例，设定 PWM_LIM0 为 3，PWM_MR0_1 为 1，

PWM_MR0_2 为 2，pwmout_0 输出设为高有效，pwmout_1 输出设为低有效，通道 0 作为定时器。波形如图 20-3 所示：

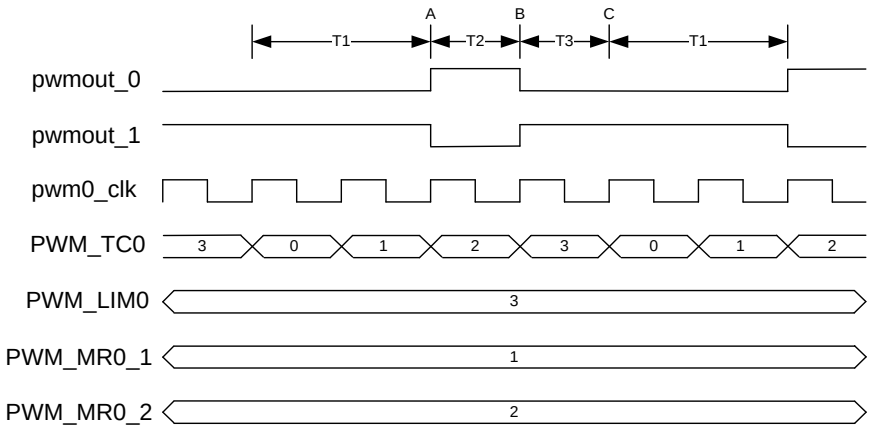


图 20-3 PWM 模式双边沿输出示意图

PWM 开始工作后，计数器 PWM_TC0 递增计数。当计数器 PWM_TC0 中的值等于匹配寄存器 PWM_MR0_1 中的值时 (图中 A 时刻)，发生匹配 1 事件，输出信号 pwmout_0、pwmout_1 由无效变为有效。计数器 PWM_TC0 继续递增，当计数器 PWM_TC0 中的值等于匹配寄存器 PWM_MR0_1 中的值时，界限寄存器 PWM_LIM0 时(图中 B 时刻)，发生匹配 2 事件，输出信号 pwmout_0、pwmout_1 由有效变为无效。计数器 PWM_TC0 继续递增，当计数器 PWM_TC0 中的值等于界限寄存器 PWM_LIM0 时(图中 C 时刻)，发生到达界限事件，输出信号 pwmout_0、pwmout_1 保持无效，计数器 PWM_TC0 复位，重新计数。

pwmout_0 无效持续时间 T1+T3=Tpwm0_clk*(PWM_LIM0-PWM_MR0_2+PWM_MR0_1+1)

pwmout_0 有效持续时间 T2= Tpwm0_clk*(PWM_MR0_2-PWM_MR0_1)

其中 Tpwm0_clk 为通道 0 的工作时钟 pwm0_clk 周期长度。

界限寄存器、匹配寄存器中的值需要满足：PWM_LIM0 > PWM_MR0_2 >PWM_MR0_1。

20.4.2 MCPWM 模式

MCPWM 模式为电机控制模式。在 MCPWM 模式下，3 个独立通道中每个通道的 2 个输出保持极性相反，支持边沿对齐与中心对齐模式，每个通道都有 1 个 10 位死区时间寄存器 (PWM_DT)支持死区模式，支持三相 DC 模式与三相 AC 模式。

20.4.2.1 边沿对齐模式

MCPWM 模式下的边沿对齐输出模式，每个通道的输出信号 pwmout 的波形需要一个界限寄存器 PWM_LIM 和一个匹配寄存器 PWM_MR 来确定：

pwmout_0 由 PWM_LIM0 和 PWM_MR0_1 确定；

pwmout_1 与 pwmout_0 极性相反；

pwmout_2 由 PWM_LIM1 和 PWM_MR1_1 确定；

pwmout_3 与 pwmout_2 极性相反；

pwmout_4 由 PWM_LIM2 和 PWM_MR2_1 确定；

pwmout_5 与 pwmout_4 极性相反。

以通道 0 的 pwmout_0、pwmout_1 为例，设定 PWM_LIM0 为 2，PWM_MR0_1 为 1，

pwmout_0 输出设为高有效，通道 0 作为定时器。波形如图 20-4 所示：

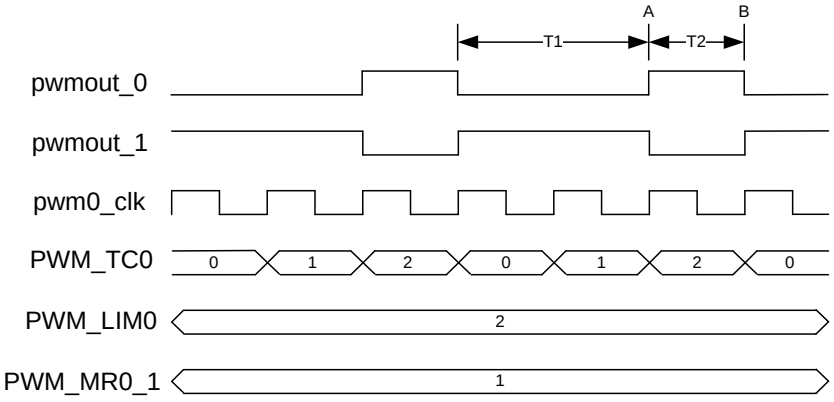


图 20-4 MCPWM 模式边沿对齐输出示意图

PWM 开始工作后，计数器 PWM_TC0 递增计数。当计数器 PWM_TC0 中的值等于匹配寄存器 PWM_MR0_1 中的值时 (图中 A 时刻)，发生匹配事件，输出信号 pwmout_0 由无效变为有效。计数器 PWM_TC0 继续递增，当计数器 PWM_TC0 中的值等于界限寄存器 PWM_LIM0 时(图中 B 时刻)，发生到达界限事件，输出信号 pwmout_0 由有效变为无效。计数器 PWM_TC0 复位，重新计数。pwmout_1 与 pwmout_0 保持极性相反。

pwmout_0 无效持续时间 T1=Tpwm0_clk*(PWM_MR0_1+1)

pwmout_0 有效持续时间 T2= Tpwm0_clk*(PWM_LIM0-PWM_MR0_1)

其中 Tpwm0_clk 为通道 0 的工作时钟 pwm0_clk 周期长度。

界限寄存器中值的范围：0x2~0xFFFF；匹配寄存器中值的范围：0x1~0xFFFE。

20.4.2.2 中心对齐模式

MCPWM 模式下的边沿对齐输出模式，每个通道的输出信号 pwmout 的波形需要一个界限寄存器 PWM_LIM 和一个匹配寄存器 PWM_MR 来确定：

pwmout_0 由 PWM_LIM0 和 PWM_MR0_1 确定；

pwmout_1 与 pwmout_0 极性相反；

pwmout_2 由 PWM_LIM1 和 PWM_MR1_1 确定；

pwmout_3 与 pwmout_2 极性相反；

pwmout_4 由 PWM_LIM2 和 PWM_MR2_1 确定；

pwmout_5 与 pwmout_4 极性相反。

以通道 0 的 pwmout_0、pwmout_1 为例，设定 PWM_LIM0 为 2，PWM_MR0_1 为 1，pwmout_0 输出设为高有效，通道 0 作为定时器。波形如图 20-5 所示：

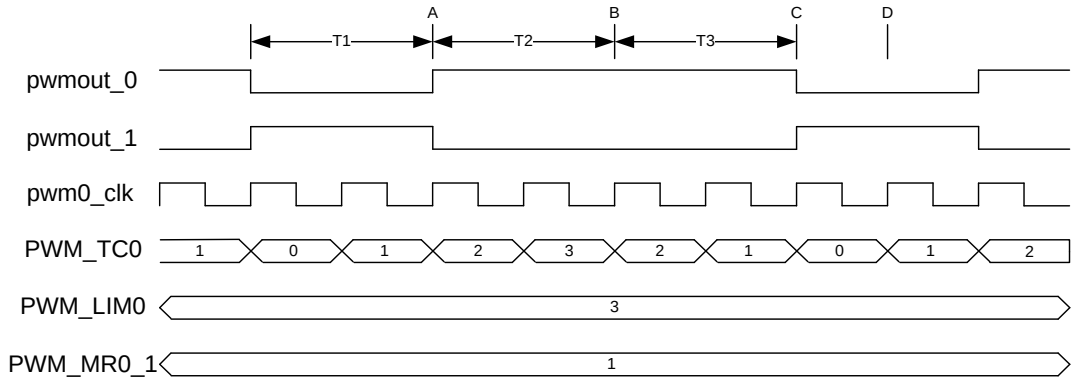


图 20-5 MCPWM 模式中心对齐输出示意图

PWM 开始工作后，计数器 PWM_TC0 递增计数。当计数器 PWM_TC0 中的值等于匹配寄存器 PWM_MR0_1 中的值时 (图中 A 时刻)，发生匹配 1 事件，输出信号 pwmout_0 由无效变为有效。计数器 PWM_TC0 继续递增，当计数器 PWM_TC0 中的值等于界限寄存器 PWM_LIM0 时(图中 B 时刻)，发生到达界限事件，输出信号 pwmout_0 保持有效，计数器 PWM_TC0 开始递减计数。当计数器 PWM_TC0 中的值再次等于匹配寄存器 PWM_MR0_1 中的值时 (图中 C 时刻)，发生匹配 2 事件，输出信号 pwmout_0 由有效变为无效。计数器 PWM_TC0 继续递减，当计数器 PWM_TC0 中的值减为 0(图中 D 时刻)，计数器 PWM_TC0 开始新一轮的递增计数。pwmout_1 与 pwmout_0 保持极性相反。

pwmout_0 无效持续时间 T1=Tpwm0_clk*(2*PWM_MR0_1)

pwmout_0 有效持续时间 T2= Tpwm0_clk*(2* PWM_LIM0-2*PWM_MR0_1)

其中 Tpwm0_clk 为通道 0 的工作时钟 pwm0_clk 周期长度。

20.4.2.3 死区模式

PWM 控制器三个通道每个通道都有 1 个 10 位死区时间寄存器(PWM_DT)支持死区模式。在 MCPWM 模式下可见，在普通 PWM 模式下不起作用。当使能通道的死区模式，这个通道的两个输出信号应该由无效变为有效的时候，都延迟一个死区时间再变为有效；应该由有效变为无效的时候，不延迟死区时间。

死区时间的操作特性是功率晶体管(例如：在电机控制应用中由 A 和 B 输出驱动的功率晶体管)完全断开所需的时间比导通的时间更长。如果 A 和 B 晶体管同时打开，那么浪费且破坏性的电流将通过晶体管在电源导轨之间流动。在这些应用中，用工作时钟周期的数目来编程死区时间寄存器，工作时钟周期数大于或等于晶体管的最大关断时间减去其最小的导通时间。

每个通道的输出信号 pwmout 的波形需要一个界限寄存器 PWM_LIM、一个匹配寄存器 PWM_MR 和一个死区时间寄存器 PWM_DT 来确定：

pwmout_0 由 PWM_LIM0、PWM_MR0_1 和 PWM_DT 确定；

pwmout_1 由 PWM_LIM0、PWM_MR0_1 和 PWM_DT 确定；

pwmout_2 由 PWM_LIM1、PWM_MR1_1 和 PWM_DT 确定；

pwmout_3 由 PWM_LIM1、PWM_MR1_1 和 PWM_DT 确定；

pwmout_4 由 PWM_LIM2、PWM_MR2_1 和 PWM_DT 确定；

pwmout_5 由 PWM_LIM2、PWM_MR2_1 和 PWM_DT 确定。

20.4.2.3.1 边沿对齐死区模式

以通道 0 的 pwmout_0、pwmout_1 为例，设定 PWM 工作 MCPWM 模式下，通道 0 使能死区，为边沿对齐模式，设置 PWM_LIM0 为 4，PWM_MR0_1 为 2，设置死区 PWM_DT[9:0] 为 1，pwmout_0 输出设为高有效，通道 0 作为定时器。波形如图 20-6 所示：

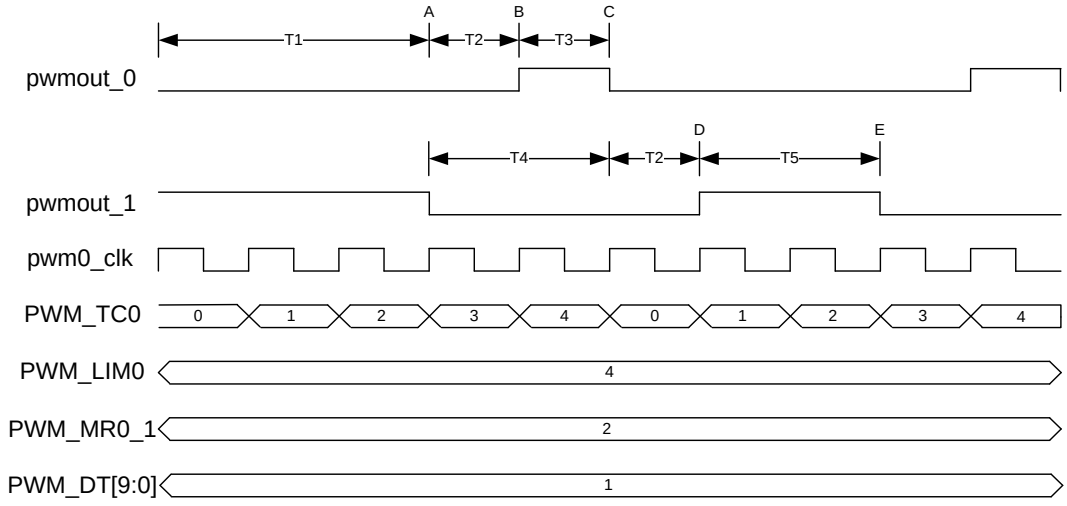


图 20-6 MCPWM 模式边沿对齐死区模式输出示意图

PWM 开始工作后，计数器 PWM_TC0 递增计数。当计数器 PWM_TC0 中的值等于匹配寄存器 PWM_MR0_1 中的值时 (图中 A 时刻)，发生匹配 1 事件，按照边沿对齐模式规则，输出信号 pwmout_0 应该此时由无效变为有效，但是由于使能了死区模式，因此延迟一个死区时间再变为有效(图中 B 时刻)。而 pwmout_1 按照边沿对齐模式规则，在 A 时刻由有效变为无效，不延迟死区时间。

计数器 PWM_TC0 继续递增，当计数器 PWM_TC0 中的值等于界限寄存器 PWM_LIM0 时 (图中 C 时刻)，发生到达界限事件，按照边沿对齐模式规则，输出信号 pwmout_0 应该此时由有效变为无效，不延迟死区时间。而 pwmout_1 按照边沿对齐模式规则，在 C 时刻由无效变为有效，但是由于使能了死区模式，因此延迟一个死区时间再变为有效(图中 D 时刻)。

计数器 PWM_TC0 开始新一轮的递增计数。

死区持续时间 T2 = PWM_DT[9:0]

pwmout_0 无效持续时间 T1+T2=Tpwm0_clk*(PWM_MR0_1+PWM_DT[9:0]+1)

pwmout_0 有效持续时间 T3=Tpwm0_clk*(PWM_LIM0-PWM_MR0_1-DT[9:0])

pwmout_1 无效持续时间 T4+T2=Tpwm0_clk*(PWM_LIM0-PWM_MR0_1+DT[9:0])

pwmout_1 有效持续时间 T3=Tpwm0_clk*(PWM_MR0_1-DT[9:0]+1)

其中 Tpwm0_clk 为通道 0 的工作时钟 pwm0_clk 周期长度。

界限寄存器、匹配寄存器、死区寄存器的值需要满足：PWM_LIM0 – PWM_MR0_1 > PWM_DT[9:0]、PWM_MR0_1 > PWM_DT[9:0]。

20.4.2.3.2 中心对齐死区模式

以通道 0 的 pwmout_0、pwmout_1 为例，设定 PWM 工作 MCPWM 模式下，通道 0 使能死区，为中心对齐模式，设置 PWM_LIM0 为 4，PWM_MR0_1 为 2，设置死区 PWM_DT[9:0]

为 1，pwmout_0 输出设为高有效，通道 0 作为定时器。波形如图 20-7 所示：

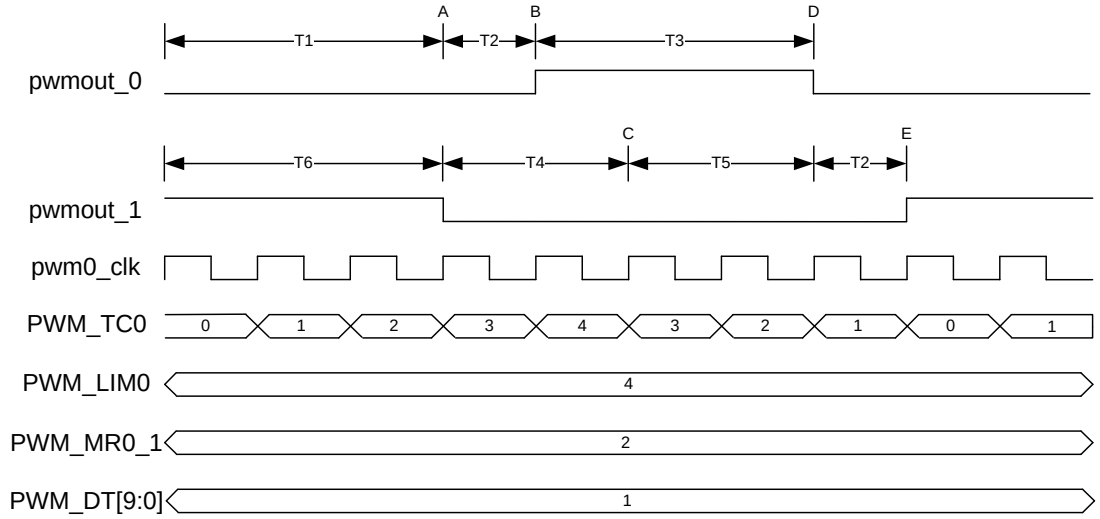


图 20-7 MCPWM 模式中心对齐死区模式输出示意图

PWM 开始工作后，计数器 PWM_TC0 递增计数。当计数器 PWM_TC0 中的值等于匹配寄存器 PWM_MR0_1 中的值时 (图中 A 时刻)，发生匹配 1 事件，按照边沿对齐模式规则，输出信号 pwmout_0 应该此时由无效变为有效，但是由于使能了死区模式，因此延迟一个死区时间再变为有效(图中 B 时刻)。而 pwmout_1 按照边沿对齐模式规则，在 A 时刻由有效变为无效，不延迟死区时间。

计数器 PWM_TC0 继续递增，当计数器 PWM_TC0 中的值等于界限寄存器 PWM_LIM0 时 (图中 C 时刻)，发生到达界限事件，输出信号 pwmout_0 保持有效，计数器 PWM_TC0 开始递减计数。

当计数器 PWM_TC0 中的值再次等于匹配寄存器 PWM_MR0_1 中的值时 (图中 D 时刻)，发生匹配 2 事件，输出信号 pwmout_0 由有效变为无效，不延迟死区时间。而 pwmout_1 按照边沿对齐模式规则，在 D 时刻由无效变为有效，但是由于使能了死区模式，因此延迟一个死区时间再变为有效(图中 E 时刻)。

计数器 PWM_TC0 继续递减，当计数器 PWM_TC0 中的值减为 0，计数器 PWM_TC0 开始新一轮的递增计数。

死区持续时间 T2 = PWM_DT[9:0]

pwmout_0 无效持续时间 T1+T2=Tpwm0_clk*(2*PWM_MR0_1+PWM_DT[9:0])

pwmout_0 有效持续时间 T3=Tpwm0_clk*(2*PWM_LIM0-2*PWM_MR0_1-PWM_DT[9:0])

pwmout_1 无效持续时间 T4 +T5+T2=Tpwm0_clk*(2*PWM_LIM0-2*PWM_MR0_1+ PWM_DT[9:0])

pwmout_1 有效持续时间 T6=Tpwm0_clk*(2*PWM_MR0_1-PWM_DT[9:0])

其中 Tpwm0_clk 为通道 0 的工作时钟 pwm0_clk 周期长度。

界限寄存器、匹配寄存器、死区寄存器的值需要满足：PWM_LIM0 – PWM_MR0_1 > PWM_DT[9:0]、PWM_MR0_1 > PWM_DT[9:0]。

20.4.2.4 三相 DC 模式

MCPWM 模式下支持三相 DC 模式，可通过 PWM_MCCTRL.bit14 设置。在三相 DC 模式下，通道 0 的 pwmout_0 信号可以被连接到任意的 pwmout 输出。具体每个 PWMOUT 输出

是否与 pwmout_0 连接，通过 PWM_CPR 设置。

可通过设置 PWM_MCCTRL.bit12 设置每个通道的两个输出极性相同还是极性相反。DC 模式下不支持死区。三相 DC 模式和三相 AC 模式不可同时置位。

例：设 PWM 工作在 MCPWM 模式下；PWM 输出设为高有效；使能三相 DC 模式；PWM_MCCTRL.bit12=1 设置每个通道的两个输出极性相同；PWM_CPR 设为 0xF，pwmout_4 和 pwmout_5 断开，其它输出激活。输出波形如图 20-8 所示：

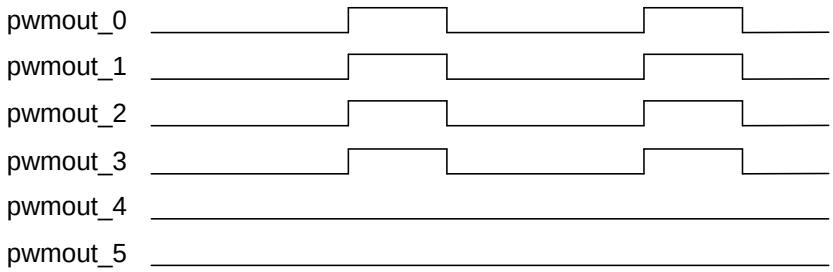


图 20-8 三相 DC 模式输出示意图

20.4.2.5 三相 AC 模式

MCPWM 模式下支持三相 AC 模式，可通过 PWM_MCCTRL.bit13 设置。在三相 AC 模式下，每个 PWM 通道均使用通道 0 的定时计数器和界限寄存器，使用自身的匹配寄存器。

例：设 PWM 工作在 MCPWM 模式下；PWM 输出设为高有效；使能 AC 模式；设置 PWM_LIM0 为 4，PWM_MR0_1 为 1，PWM_MR1_1 为 2，PWM_MR2_1 为 3 输出波形如图 20-9 所示：

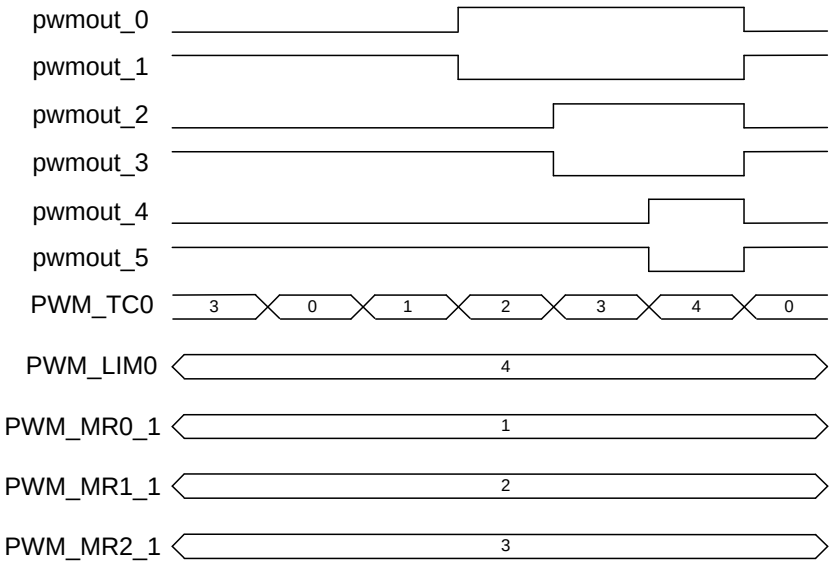


图 20-9 三相 AC 模式输出示意图

20.4.3 定时器模式与计数器模式

PWM 控制器的三个通道可工作在定时器模式下或者工作在计数器模式下。可通过 PWM_CNTCTRL[2:0]对三个通道分别设置。

当 PWM 控制器的某个通道工作在定时器模式下，则这个通道的计数器值根据这个通道的工作时钟递增或递减计数；当 PWM 控制器的某个通道工作在计数器模式下，则这个通道的计数器值根据相应输入信号的上升沿、下降沿或者上升和下降沿递增或递减计数。

20.4.4 增量式旋转编码器鉴相模块

PWM 控制器包括一个增量式旋转编码器鉴相模块，能够由硬件判断外部旋转编码器的旋转方向，并由硬件提供计算转速的参数，提供实时高精度旋转位置信息。

增量式旋转编码器是一种用来测量转速的装置，它根据转轴的旋转方向与速度输出三组方波脉冲 A、B 和 Z，A、B 两组脉冲相位差 90°，可通过 PWM 控制器的增量式旋转编码器鉴相模块判断出旋转方向，而 Z 相为转轴每转一个周期输出一个脉冲，用于定位转轴旋转角度。

PWM 对增量式旋转编码器进行鉴相并计算转速，需要将增量式旋转编码器的 3 个输出信号 A，B，Z 依次接到 PWM 的输入信号 pwmin_0，pwmin_1，pwmin_2 上，使能旋转编码器鉴相功能后，可通过读寄存器 PWM_RE_STATE 得到旋转编码器的旋转方向，并可通过寄存器 PWM_RE_LIM 设定计数时间 Tc，当产生旋转编码器匹配中断(IR_RE_MAT)后读 PWM_RE_M1，PWM_RE_M2 寄存器的值，根据 M/T 法计算转速(rpm)：

$$n=(15*M1*fclk)/(N*M2)$$

其中 N 为旋转编码器的分辨率，表示增量式旋转编码器旋转一周的时间内 A 或 B 信号发出的脉冲信号个数。参数 M1 为寄存器 PWM_RE_M1 中的值，表示一定时间 Tc 内 A，B 信号发生翻转的次数。参数 M2 为寄存器 PWM_RE_M2 中的值，表示一定时间 Tc 内旋转编码器鉴相模块工作时钟(pwmr_clk)周期数。

可根据参数 M3 和参数 RE_TC3 算出旋转的实时位置：

$$d=(RE\_TC3/M3)*360°$$

其中参数 M3 为寄存器 PWM_RE_M3 中的值，表示增量式旋转编码器旋转一周的时间内鉴相模块的工作时钟周期数。参数 RE_TC3 为寄存器 PWM_RE_TC3 中的值，表示当前时刻鉴相模块的工作时钟周期数。

为了使旋转编码器鉴相模块计算的精度更高，应使旋转编码器鉴相模块的工作时钟(pwmr_clk)频率高于旋转编码器设备最高转速(每秒转速)与分辨率(参数 N)乘积的 8 倍，为了避免计数器溢出，也不要设置 pwmr_clk 频率过高，pwmr_clk 频率为旋转编码器设备最高转速(每秒转速)与分辨率(参数 N)乘积的 8 到 10 倍为宜。

使用旋转编码码器鉴相模块不影响 PWM 其他功能的使用。

20.4.5 中断与中断屏蔽

PWM 每个通道都可能有 1 个到达界限中断和 2 个匹配中断， PWM 模块还有 3 个捕获中断和 1 个快速终止中断，一共 13 个中断。采用写 1 清零的方式，写 1 清零中断寄存器时，不用加载使能寄存器，直接写中断寄存器就可生效。

每个中断都可以通过设置中断屏蔽寄存器 PWM_IER 屏蔽或者打开。复位后的初始状态均为屏蔽状态。

20.4.6 时钟产生

PWM 模块包含三个通道，每个通道都有一个时钟输入，以及旋转编码鉴相器有一个时钟输入，共四个时钟输入：pwm0_clk，pwm1_clk，pwm2_clk 和 pwmr_clk，均由 APB 时钟整数分频而得。如果不需要某个通道输出，可以通过软件关闭这个通道的时钟输入以节省功耗。这三个时钟的频率由系统控制模块 SYSCTL 提供，具体配置请参见系统控制模块。

20.5 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

20.6 编程指导

当前版本用户手册暂不提供详细编程指导。

21 可编程定时器

21.1 概述

GSC3281 芯片包含一个 TIMER 定时器模块，TIMER 定时器模块包含 TIMER0~TIMER3 共 4 个相对独立的 32bit 定时器，每一个定时器都有自己独立的时钟源，均支持循环定时模式和单次定时模式两种工作模式，每个定时器都有各自的中断，TIMER 定时器模块将 4 个中断合并为一个中断输出给中断控制器，并额外提供一个 int_for_adc 中断。

下文中 TIMER0~TIMER3 用 TIMERn 代替。n=0，1，2，3。

21.2 结构框图

TIMER 定时器模块的时钟与复位信号均由系统控制模块 SYSCTL 提供，TIMER0~TIMER3 均有各自的时钟与复位信号，时钟可由 SYSCTL 系统控制模块分别配置，为 PCLK 时钟的分频，分频系数为 2、4、8、16、….、1024 可选。

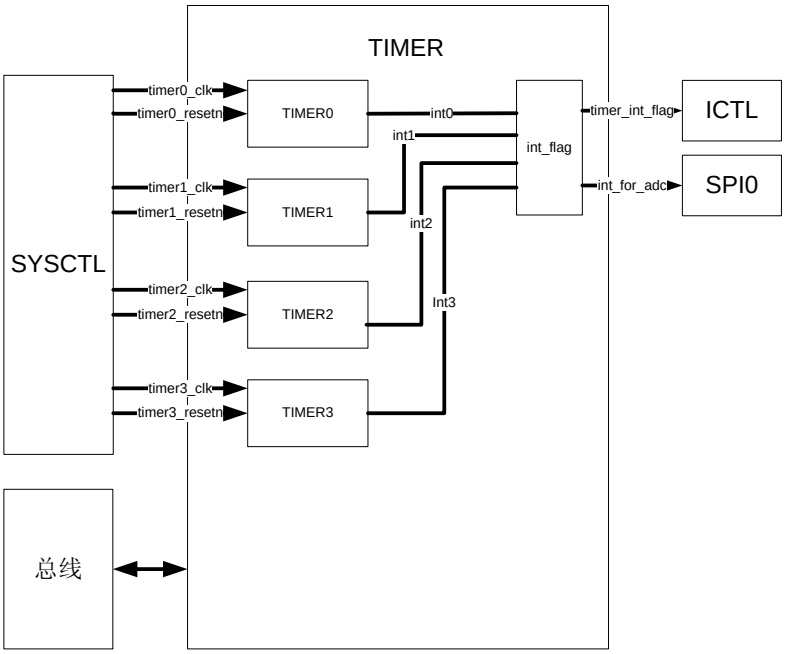


图 21-1 TIMER 结构框图

int_for_adc 信号提供给 SPI0 模块用于 ADC 采样的中断信号，由 TIMER0 的中断信号经过处理得到。

21.3 功能描述

TIMER 定时器模块包含 TIMER0~TIMER3 共 4 个 TIMER，每个 TIMER 功能相同，各自独立。TIMER 的计数器宽度为 32bit。

当系统复位后 TIMER 开启、关闭或 TIMER 的计数器递减至 0 时，计数器自动重载。

TIMER 定时器模块支持循环定时模式和单次定时模式两种工作模式。

TIMER 定时器模块具有可屏蔽读清零中断。

21.3.1 使能与禁止

TIMER 定时器包含的 4 个定时器 TIMER0~TIMER3 可分别使能和禁止。4 个定时器均有各自的设置寄存器 TIMERn_TCR(n 表示 TIMER0、TIMER1、TIMER2 或 TIMER3，下同)，可通过 TIMERn_TCR.bit0 来使能和禁止相应的 TIMER，置 1 为使能，置 0 为禁止。当 TIMER 使能后，计数器载入值寄存器(TIMERn_TLC)中的值载入计数器，计数器开始在之后的每个工作时钟信号上升沿递减计数。TIMER 的计数器宽度为 32bit。

当某个定时器状态由开启变为关闭时，此定时器停止计数，同时包括计数器在内的所有寄存器被异步复位。

21.3.2 定时器重载

当定时器的状态由关闭转变为开启时会引起计数器重载，在下一个时钟上升沿时计数器载入值寄存器(TIMERn_TLC)的值载入计数器。

当定时器工作在循环定时模式，每当计数器递减为 0，在下一个时钟上升沿时计数器载入值寄存器(TIMERn_TLC)的值重新载入计数器。

当定时器工作在单次定时模式，当计数器递减为 0，在下一个时钟上升沿时计数器载入最大值 0xFFFFFFFF。用户有足够的时间来完成中断处理及定时器配置相关操作。

21.3.3 中断屏蔽与中断清除

TIMER 定时器中 4 个单独的定时器 TIMER0~TIMER3 均有各自的中断状态寄存器 TIMERn_TIS 和中断清除寄存器 TIMERn_TEOI。除此之外 TIMER 定时器还有全局中断状态寄存器 TIMER_TIS，全局中断清除寄存器 TIMER_TEOI 以及全局中断原始状态寄存器 TIMER_TRIS，可对中断进行全局处理。

读中断状态寄存器 TIMERn_TIS 或全局中断状态寄存器 TIMER_TIS 可查询 4 个 TIMER 是否有中断产生。中断状态寄存器 TIMERn_TIS 为 1 表示 TIMERn 有中断，为 0 表示没有中断。全局中断状态寄存器 TIMER_TIS.bit0 为 1 表示 TIMER0 有中断，TIMER_TIS.bit1 为 1 表示 TIMER1 有中断，TIMER_TIS.bit2 为 1 表示 TIMER2 有中断，TIMER_TIS.bit3 为 1 表示 TIMER3 有中断。

读中断清除寄存器 TIMERn_TEOI 或全局中断清除寄存器 TIMER_TEOI 可清除 4 个 TIMER 的中断。读中断清除寄存器 TIMERn_TEOI 可清除 TIMERn 的中断。读全局中断清除寄存器 TIMER_TEOI 可清除所有中断。

每个 TIMER 定时器的寄存器 TIMERn_TCR.bit2 可设置是否屏蔽中断，设为 1 表示屏蔽中断，设为 0 表示允许中断。当屏蔽中断后，TIMER 不会产生中断。

TIMER 定时器提供一个全局中断原始状态寄存器 TIMER_TRIS，无论 TIMER 是否屏蔽中断，只要达到中断产生条件，都会使 TIMER_TRIS 的相应位被置为 1。TIMER_TRIS.bit0 置为 1 表示 TIMER0 达到产生中断的条件，TIMER_TRIS.bit1 置为 1 表示 TIMER1 达到产生中断的条件，TIMER_TRIS.bit2 置为 1 表示 TIMER2 达到产生中断的条件，TIMER_TRIS.bit3 置为 1 表示 TIMER3 达到产生中断的条件。

21.3.4 工作模式

4 个独立的定时器均支持循环定时模式和单次定时两种工作模式。

21.3.4.1 循环定时模式

定时器工作在循环定时模式时，每当计数器递减为 0，在下一个时钟上升沿时计数器载入值寄存器(TIMERn_TLC)的值重新载入计数器，并产生中断(中断允许的情况下)。期望周期性的定时可以采用该模式。

21.3.4.2 单次定时模式

定时器工作在单次定时模式时，当计数器递减为 0，在下一个时钟上升沿时计数器载入最大值 0xFFFFFFFF，并产生中断(中断允许的情况下)。用户有足够的时间来完成中断处理及定时器配置相关操作。在只需要单次定时的场合建议使用该模式。

21.4 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

21.5 编程指导

当前版本用户手册暂不提供详细编程指导。

22 看门狗定时器

22.1 概述

GSC3281 芯片内部包含一个 WatchDogTimer 看门狗定时器模块(WDT)，可以用来防止可能引起的系统锁死。WDT 内部有个 32 位计数器进行递减计数，软件需要每隔一段时间对计数器进行重启。当系统锁死，则软件无法对计数器进行重启，会导致计数器减为 0，这时 WDT 模块可以产生一个超时中断，如果中断长时间没有得到处理则 WDT 判断系统锁死从而对系统进行复位，或者不产生超时中断，WDT 直接对系统进行复位。

22.2 结构框图

WDT 模块的输入信号由系统控制模块 SYSCTL 给出，包括控制暂停以及使 WDT 复位的控制信号、WDT 工作时钟信号 pclk_wdt。

pclk_wdt 工作频率等于 APB 总线时钟 pclk 的工作频率。

WDT 产生的超时中断信号提供给中断控制器。

WDT 生成的系统复位信号 wdt_sys_rst 提供给系统控制模块 SYSCTL，再由系统控制模块 SYSCTL 生成芯片全局复位信号 sys_rstn。

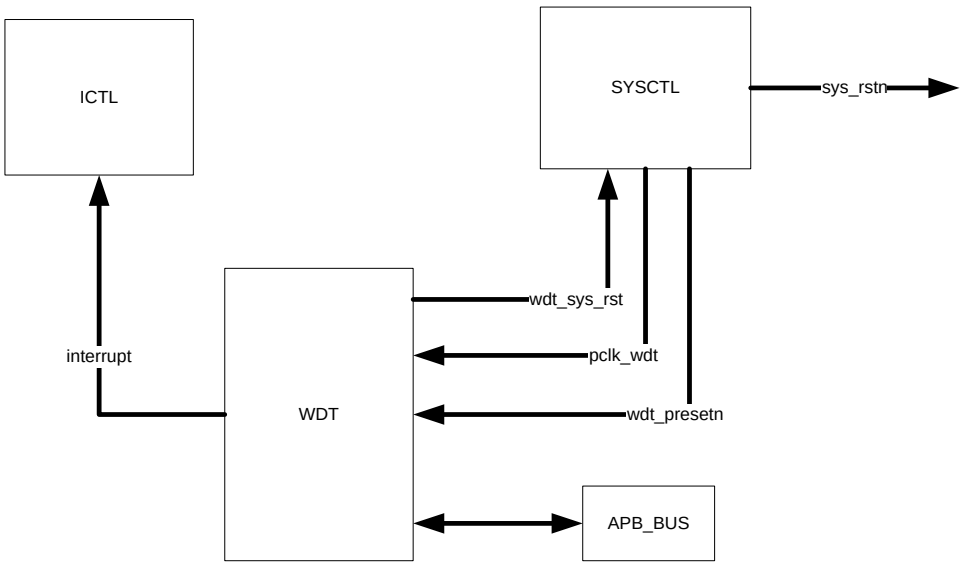


图 22-1 WDT 结构框图

22.3 功能描述

WDT 内部包含 32 位计数器。WDT 运行时计数器递减计数。可配置当计数器减为 0 时 WDT 发出超时中断还是直接系统复位。发出超时中断后 WDT 等待中断处理的时间长度可配置，超过等待中断处理的时间还没有重启计数器则系统复位。支持暂停功能。

22.3.1 计数器

WDT 包含一个 32bit 计数器 couner。WDT 从初始值 0xFFFF 开始递减计数，当计数值递减至 0 时，如果控制寄存器 WDT_CR 的复位模式选择位(RMOD)为 1，则产生超时中断信号并从软件配置的新的初始值(由 WDT_TORR 的 TOP 位设置)重新开始递减计数，如果控制寄存器 WDT_CR 的复位模式选择位(RMOD)为 0，直接产生系统复位。

可通过 SYSCTL 系统控制模块 SYSCTL_WDT_CFG 寄存器的 bit[3:0]，设置计数器的计数频率，每 1 个 pclk_wdt 计数一次、每 2 个 pclk_wdt、4 个 pclk_wdt、8 个 pclk_wdt 计数一次……最高支持每 32768 个 pclk_wdt 周期计数一次。

例如，SYSCTL_WDT_CFG 寄存器的 bit[3:0]设为 0，表示 WDT 计数器每 1 个 pclk_wdt 周期计数一次，SYSCTL_WDT_CFG 寄存器的 bit[3:0]设为 0x1，表示 WDT 计数器每 2 个 pclk_wdt 周期计数一次，SYSCTL_WDT_CFG 寄存器的 bit[3:0]设为 0x2，表示 WDT 计数器每 4 个 pclk_wdt 周期计数一次……

用户发起的“喂狗”操作(写 0x76 至 WDT_CRR 寄存器)可以重启 WDT，并自动重载计数器值(重载值由 WDT_TORR 的 TOP 位设置)，开始新一轮的定时操作。

“喂狗”操作也可以清除 WDT 超时中断。

22.3.2 中断与中断清除

WDT 只有一个中断，为超时中断。

WDT 超时中断产生后必须在下一次 WDT 计数器减为 0 之前清除，否则下一次定时器减为 0 时 WDT 会令系统复位。

读取 WDT_EOI 寄存器可以清除超时中断，此外“喂狗”操作也可以清除超时中断。读取 WDT_EOI 寄存器清除超时中断并不会使 WDT 计数器重启，计数器会继续递减操作，如果直到递减为 0 仍不进行“喂狗”操作的话会再次产生 WDT 超时中断。“喂狗”操作会使 WDT 计数器重启。

22.3.3 暂停

WDT 支持暂停功能，当 WDT 被暂停时计数器停止计数。如果暂停 WDT 时计数器值刚好为 0，则不会产生超时中断或系统复位，直到取消暂停后的下一个时钟周期产生超时中断或系统复位。

暂停功能可通过 SYSCTL 系统控制模块的 SYSCTL_WDT_CFG 寄存器 bit4 位设置。置 1 为暂停，置 0 为取消暂停。

22.3.4 生成复位信号

WDT 生成系统复位信号 wdt_sys_rst 提供给系统控制模块，再由系统控制模块生成芯片全局复位信号 sys_rstn。

WDT 计数器复位和系统复位信号时序如图 22-2 所示。

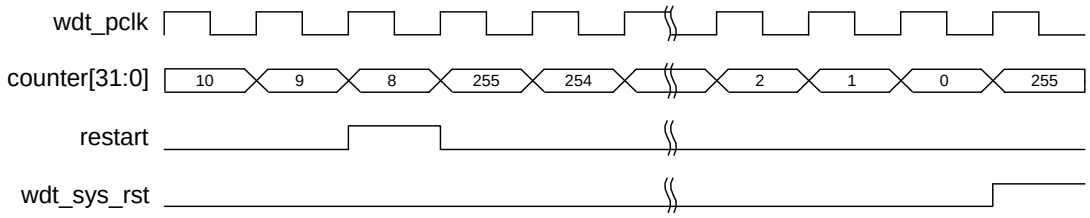


图 22-2 WDT 计数器复位和系统复位信号时序图

图 22-2 中 restart 信号为“喂狗”操作产生的 WDT 内部信号，用于 WDT 计数器重启。

WDT 控制寄存器 WDT_CR 的 RPL 位(bit2~bit4)值决定 WDT 输出系统复位信号 wdt_sys_rst 有效宽度。可选的宽度为 2、4、8、16、32、64、128 或 256 个 pclk_wdt 时钟周期。

22.3.5 CPU 等待模式下 WDT 使能

CPU 可通过 WAIT 命令进入等待模式。可配置 WDT 在 CPU 等待模式下是否工作，通过 SYSCTL 系统控制模块的 SYSCTL_WDT_CFG 寄存器 bit5 的 wait_ena 位来设置。置 1 表示等待模式下 WDT 工作，置 0 表示等待模式下 WDT 不工作。

22.4 WDT 工作流程图

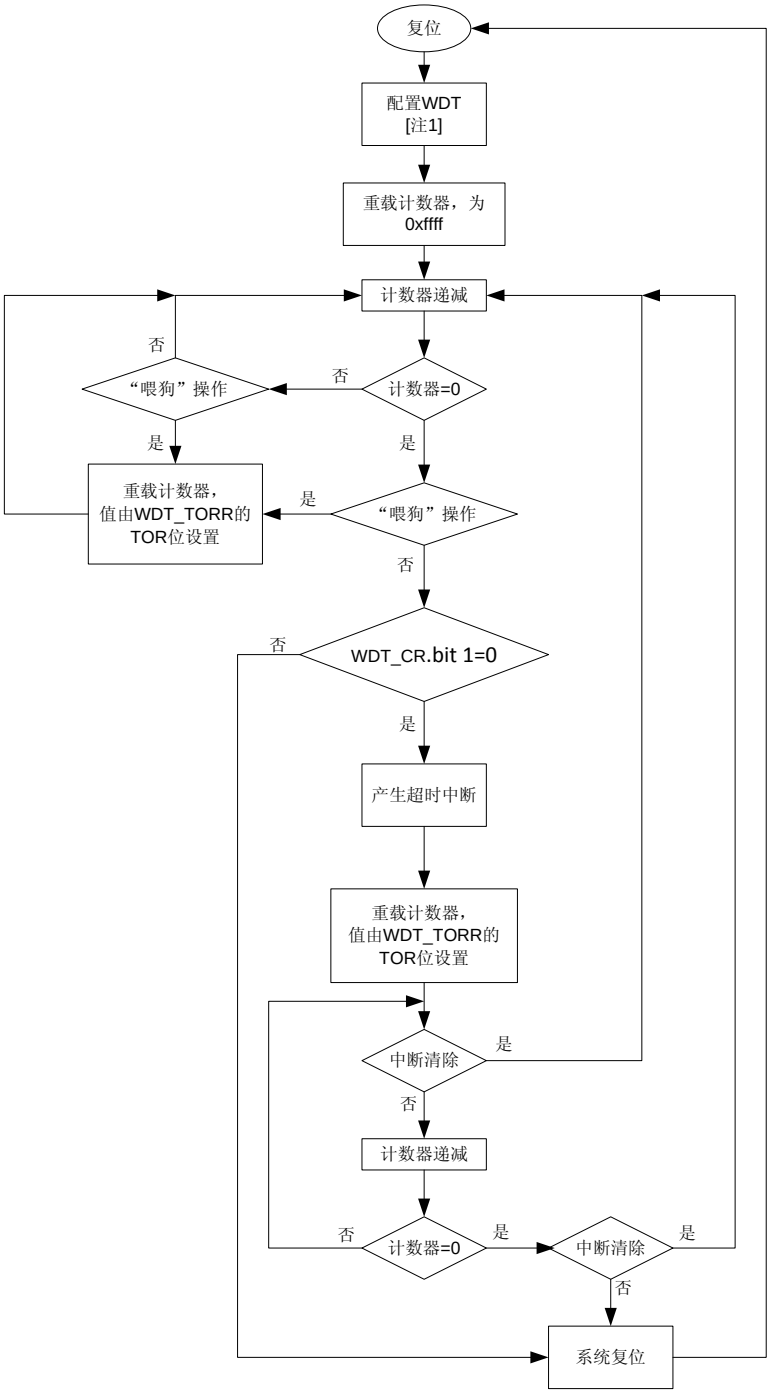


图 22-3 WDT 工作流程图

[注 1]：配置 WDT 包括配置 WDT_TORR 寄存器和 WDT_CR 寄存器。

22.5 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

22.6 编程指导

当前版本用户手册暂不提供详细编程指导。

23 可编程输入输出接口

23.1 概述

GSC3281 中除了专用引脚之外的所有引脚都复用为 GPIO 引脚，CPU 通过 GPIO 控制器来访问 GPIO 引脚。GSC3281 包含 87 个多功能输入/输出端口管脚，GSC3281 的 87 个 GPIO 由 A，B，C 三组组成，A 组有 31 个 GPIO，B 组有 32 个 GPIO，C 组有 24 个 GPIO。其中 A 组的 31 个 GPIO 具有中断功能。

三组 GPIO 与 GSC3281 芯片引脚的对应关系如表 23-1 所示，类型为 GPI 时，说明该端口只能作可编程输入使用，如果配置成输出则无效。类型为 GPO，说明该端口只能作可编程输出使用，如果配置成输入则无效。类型为 GPIO，说明该端口作可编程输入输出使用。

表 23-1 GPIO 与引脚的对应关系

引脚名称	对应的 GPIO	类型
u6rtsn_u3txe	gpioa[00]	GPIO
u6ctsn_u4rxd	gpioa[01]	GPIO
u6dtrn_u4txd	gpioa[02]	GPIO
u6dcdn_u4txe	gpioa[03]	GPIO
u6ri	gpioa[04]	GPIO
u6dsrn	gpioa[05]	GPIO
u5rxd_sim1clk	gpioa[06]	GPIO
u5txd_sim1io	gpioa[07]	GPIO
gpio8	gpioa[08]	GPIO
emia3	gpioa[09]	GPIO
emia4	gpioa[10]	GPIO
emia5	gpioa[11]	GPIO
emia6	gpioa[12]	GPIO
emia7	gpioa[13]	GPIO
emia8	gpioa[14]	GPIO
sim0rstn	gpioa[16]	GPIO
sim0clk_row3_u3txe	gpioa[17]	GPIO
sim0io_col3_u4txe	gpioa[18]	GPIO
cap0_psdat1	gpioa[19]	GPIO
pwmabort_psclk1	gpioa[20]	GPIO
pwmout3_bootmode1_psclk0	gpioa[21]	GPIO
pwmout4_psdat0	gpioa[22]	GPIO
pwmout5_i2sclk	gpioa[23]	GPIO
spi0miso_col1	gpioa[24]	GPIO
spi0mosi_row1	gpioa[25]	GPIO
spi0sck_col0_emia0	gpioa[26]	GPIO
jtck	gpioa[27]	GPI

jrstn	gpioa[28]	GPI
jtms_testcmprn_cap2	gpioa[29]	GPI
sim0vccen_row0_emia1	gpioa[30]	GPIO
emia2	gpioa[31]	GPIO
nfcsn_bootmode0	gpiob[00]	GPO
nfcle_clkcfg0	gpiob[01]	GPO
nfale_clkcfg1	gpiob[02]	GPO
nfren_clkcfg2_emioen	gpiob[03]	GPO
nfwen_emiwen	gpiob[04]	GPIO
nfrnb_emirdy	gpiob[05]	GPIO
nfdat0_emid0	gpiob[06]	GPIO
nfdat1_emid1	gpiob[07]	GPIO
nfdat2_emid2	gpiob[08]	GPIO
nfdat3_emid3	gpiob[09]	GPIO
nfdat4_emid4	gpiob[10]	GPIO
nfdat5_emid5	gpiob[11]	GPIO
nfdat6_emid6	gpiob[12]	GPIO
nfdat7_emid7	gpiob[13]	GPIO
u6rxd_u3rxd	gpiob[14]	GPIO
u6txd_u3txd	gpiob[15]	GPIO
u0rxd_u5txe	gpiob[16]	GPIO
u0txd_sim1rstn	gpiob[17]	GPIO
u2rxd	gpiob[18]	GPIO
u2txd	gpiob[19]	GPIO
u3rxd	gpiob[20]	GPIO
u3txd	gpiob[21]	GPIO
u4rxd	gpiob[22]	GPIO
u4txd	gpiob[23]	GPIO
u7rxd	gpiob[24]	GPIO
u7txd	gpiob[25]	GPIO
sim1rstn	gpiob[26]	GPIO
sim1clk	gpiob[27]	GPIO
sim1io	gpiob[28]	GPIO
spi1csn	gpiob[29]	GPIO
gpio62	gpiob[30]	GPIO
clkout	gpiob[31]	GPIO
spi1miso	gpioc[00]	GPIO
spi1sck	gpioc[01]	GPIO
spi1mosi	gpioc[02]	GPIO
rmclk	gpioc[03]	GPIO
rmtxen	gpioc[04]	GPIO
rmtxd0	gpioc[05]	GPIO
rmtxd1	gpioc[06]	GPIO

rmrxdv	gpioc[07]	GPIO
rmrxd0	gpioc[08]	GPIO
rmrxd1	gpioc[09]	GPIO
mdc	gpioc[10]	GPIO
mdio	gpioc[11]	GPIO
i2cscl	gpioc[12]	GPIO
i2csda	gpioc[13]	GPIO
emicsn0_i2sws	gpioc[14]	GPIO
row2	gpioc[15]	GPIO
col2	gpioc[16]	GPIO
jtdi_bistmoden_u1rxd_cap1	gpioc[17]	GPI
jtdo_u1txd_pwmout2	gpioc[18]	GPO
emicsn1_i2ssdi_pwmout1	gpioc[19]	GPIO
sim1vccen_i2ssdo	gpioc[20]	GPIO
emicsn2_utmidrvvbus_u5txe	gpioc[21]	GPIO
pwmout0	gpioc[22]	GPIO
gpio87	gpioc[23]	GPIO

23.2 功能说明

23.2.1 功能特点

1. 87 个 GPIO 软件可配，其中 A 组 31 个 GPIO 具有中断功能。
2. 中断可配置四种方式：高电平触发，低电平触发，上升沿触发和下降沿触发。
3. 通过软件配置，GPIO 中断可具有防抖去毛刺功能。
4. 可通过读寄存器读取端口外部信号。

23.2.2 功能描述

GPIO 结构如图 23-1 所示，每一位 GPIO 都是由图中所示的结构组成。输出数据是由数据寄存器（GPIO_SWPORT_DR）决定，方向由方向寄存器（GPIO_SWPORT_DDR）决定。CPU 通过读取 GPIO_EXT_PORT 寄存器可以得到 GPIO 的值。

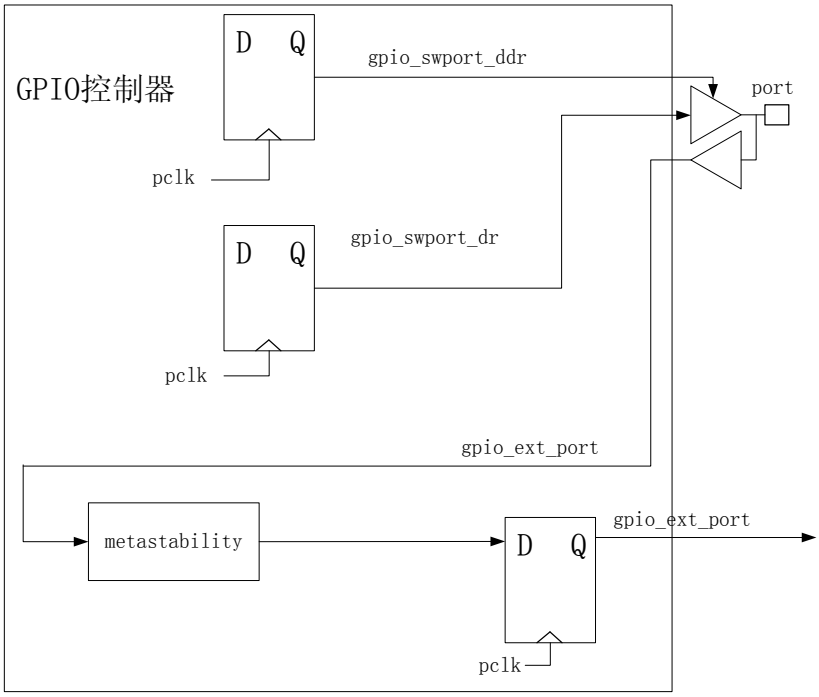


图 23-1 GPIO 控制器结构图

23.2.3 中断

A 组的 31 个 GPIO 可作为中断使用，外部信号为中断源，中断类型软件可配，通过配置 GPIO_INTTYPE_LEVEL 寄存器和 GPIO_INT_POLARITY 寄存器可以实现下面四种中断触发类型：

- ✧ 高电平触发
- ✧ 低电平触发
- ✧ 上升沿触发
- ✧ 下降沿触发

GPIO 作为中断使用时，可以通过配置 GPIO_INTMASK 寄存器来屏蔽这些中断。如果中断被屏蔽，也可以通过读取 GPIO_RAW_INTSTATUS 来获知中断的原始状态。当 GPIO 需要作为中断使用时，对应的数据方向必须设为输入，否则将检测不到中断。数据方向通过寄存器 GPIO_SWPORTA_DDR 来配置。

当 GPIO 检测到沿触发的中断时，可以通过向寄存器 GPIO_PORTA_EOI 的对应比特位写 1 来清掉中断。但是如果检测电平触发的中断时，则无法通过上述方法清掉中断。

23.2.4 消抖功能

GPIO 作为中断使用时，可以通过配置 GPIO_DEBOUNCE 寄存器的相应比特位，使对应的 GPIO 具有消抖去毛刺功能。GSC3281 中的系统控制模块提供消抖时钟 dbclk，它的频率大小根据消抖需求可进行配置。如果使能某一位的 GPIO 具有消抖功能，任何小于一个 dbclk 时钟周期的信号就将被视为毛刺而被滤掉，如图 23-2 所示，只有被 dbclk 的上升沿采样两次以上，信号才会被传递到 GPIO 控制器内部。

消抖时钟 dbclk 的频率大小通过系统控制器模块中的 SYSCTL_CLKDIV_GPIODB 寄存器来进行配置。

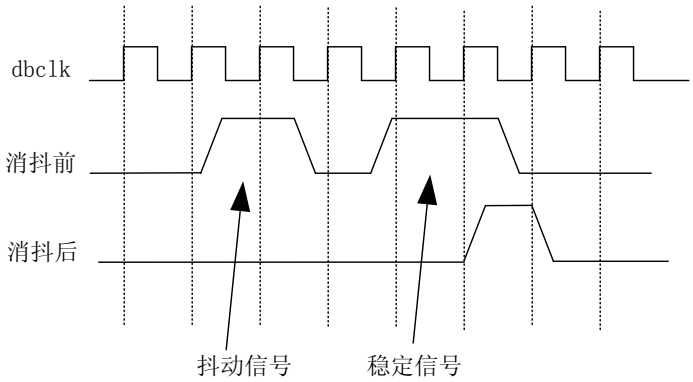


图 23-2 消抖时序图

23.3 寄存器描述

当前版本用户手册暂不提供详细寄存器描述。

23.4 编程指导

当前版本用户手册暂不提供详细编程指导。

24 订购信息

产品描述

表 24-1 GSC3281 产品描述

编号	版本号	封装形式	温度范围
GSC3281	-	LFBGA256	工业级-40℃~85℃

产品用途

客户信息

公司名称：_____电话：_____

个人签名：_____职务：_____

25 修订历史

表 25-1 GSC3281 用户手册修订历史

序号	芯片版本	修订内容	修订时间
1	GSC3281	发布正式版本	2012-01-11
2	GSC3281	发布公开版本	2013-06-24