ICH2 软件接口手册

2017 年 10 月

成都申威科技有限责任公司

成都市华府大道四段电子科大科技园 D22 栋
Building D22, National University Science and technology park,
Section 4, Huafu Avenue, Chengdu

免责声明

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

成都申威科技有限责任公司

Chengdu Sunway Technology Corporation Limited

地址：成都市华府大道四段电子科大科技园 D22 栋

Building D22, National University Science and technology park,

Section 4, Huafu Avenue, Chengdu

Mail：sales@swcpu.cn

Tel ：028-68769016

Fax：028-68769019

阅读指南

《ICH2 软件接口手册》主要描述了国产第二代套片的芯片结构、空间编制和内部功能接口等内容。

文档修订

文档更新记录	文档名	ICH2 软件接口手册
	版本号	V1.0
	创建人	研发部
	创建日期	2017-10-8

版本更新

版本号	更新内容	更新日期
V1.0	初稿	2017-10-8

技术支持

可通过邮箱或问题反馈网站向我司提交产品使用的问题，并获取技术支持。

售后服务邮箱：sales@swcpu.cn

问题反馈网址：http://www.swcpu.cn/

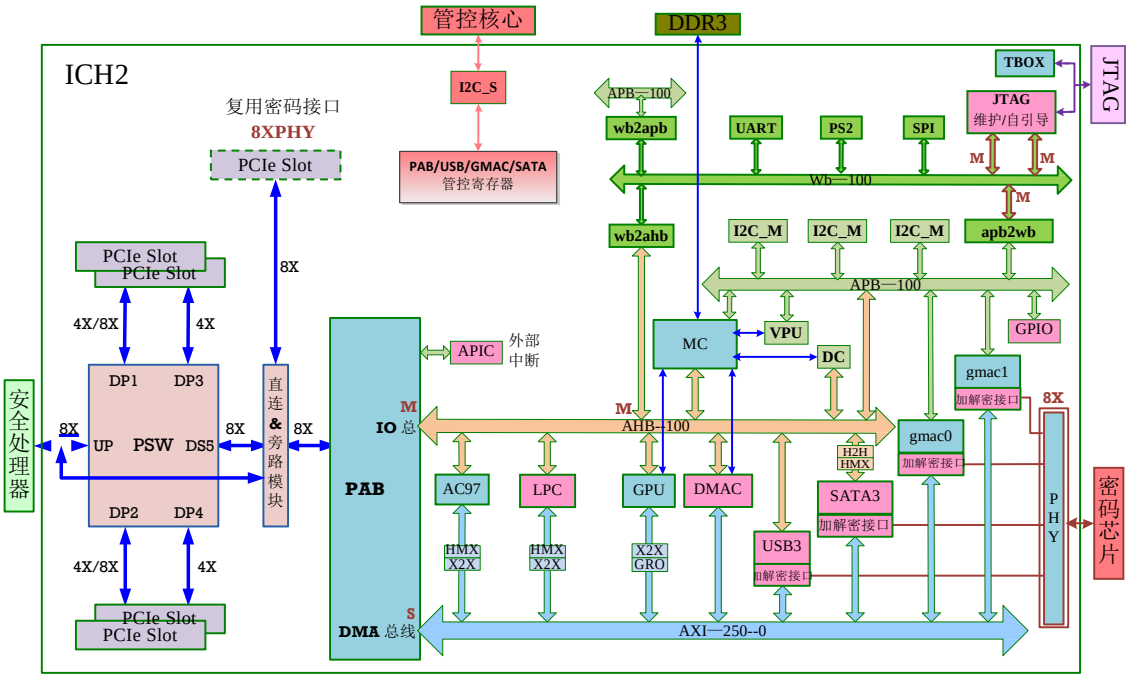
目　录

1 概述

本手册定义 ICH2 芯片同操作系统间的协议要求，由芯片设计组起草，由软件系统组确认。

1.1 ICH2 芯片结构

ICH2 芯片实现计算平台的外围 IO 扩展。相对第一代 ICH 芯片，要实现硬件的 IO 管控技 术，对 SATA 接口、USB 接口和以太网接口实现数据加解密。同时，对第一代 IO 芯片的结构、 接口、性能进行优化。



1.2 ICH2 芯片主要功能

1) 实现所有设备在不同安全策略下的启闭控制。实现通过管控核心和自引导固件两种途径进 行工作模式配置

2) 实现 USB 和 SATA 双向明通、双向密通、单向明通和禁用四种工作模式；实现 GMAC 双 向

明通、双向密通和禁用三种工作模式。

3) 实现独立的高速串行链路为密码调用接口。
4) 采用标准的 PCIE2.0-X8 接口实现与安全处理器对接。
5) 集成高性能独立显卡模式的 GPU/VPU，配备专用的 DDR3 显存和 DMA 控制器。
6) 集成可旁路的 PCIE2.0 交叉开关，下游扩展最多 3 个 X8，或者 1 个 X8+4 个 X4。
7) 设计 LPC 控制器实现 LPC 总线扩展。
8) 支持显示、声音、USB、SATA、千兆以太网等常用 IO 接口。

1.3 ICH2 芯片技术指标

1) 与处理器采用 PCI-E 2.0 x8 标准接口对接，链路传输率为 5Gbps，双向有效带宽 8GB/s；
2) 集成显卡支持最大分辨率 1920*1080（1080P）的高清显示；
3) 支持 2GB，64 位宽，接口传输率为 1066~2133Mbps 的 DDR3 存储器作为独立显存；
4) 实现 1 个 PCI-E 2.0 x8 高速串行链路作为密码调用接口；
5) 实现 USB3.0、SATA3.0 的应用，支持 2 个 USB2.0 接口、3 个 SATA3.0 接口、1 个千兆以 太网接口双向明通、双向密通等多种工作模式；
6) 采用 250MHz，128 位宽的片内数据总线；GPU 核心工作频率 800MHz；
7) 应用于桌面系统的 ICH2 芯片满配功耗控制在 8W 以内；应用于便携系统的 ICH2 芯片关闭 部分 PCIE 扩展端口，整芯片功耗控制在 6W 以内；
8) ESD 保护电压：>2000V；
9) 工作温度：0℃～+70℃；
10) 存储温度：-25℃～+85℃。

1.4 工作模式说明

USB、SATA、GMAC 三种控制器可以通过自引导固件进行工作模式配置。其中，USB 和 SATA 支持双向明通、双向密通、单向明通和禁用四种工作模式，GMAC 支持双向明通、双向 密通和禁用三种工作模式。

由自引导固件静态配置 USB、SATA、GMAC 的工作模式，三种设备均不支持运行过程中 的模式切换。

1.5 禁用模式说明

由于 USB 支持运行过程中的模式切换，因此处于禁用模式时 USB 控制器既要禁用数据传 输，又要支持计算核心的驱动加载。因此 USB 控制器在禁用模式下应该支持以下行为：

1) 寄存器通路：保持 USB 控制器和 PHY 的寄存器通路，可以正常进行读写，以支持正常的 USB 驱动流程。
2) DMA 通路：要按设备初始化的要求，正常发起描述符的 DMA 传输。但并不发起针对 USB 设备的数据 DMA 传输。
3) 中断通路：硬件需要支持中断。
4) 硬件不发起 DMA 传输，只保证 AXI 总线上请求和响应的完整性，禁用模式下软件层次的 握手完整性需要软件自己保证。 总结：在禁用模式下需要完全支持通用模式下的设备初始化流程，但要管住发向/来自设备

的数据 DMA 传输。

而 SATA 和 GMAC 硬件不支持从禁用模式切换到其他模式，也不保证运行过程中其他模式（明/密/单）之间切换时通路和缓冲的干净，如果一定要切换需要由软件流程保证正确性。所以，SATA 和 GAMC 一旦进入禁用模式，设备控制器不再保证驱动能正常工作，此时管控核心不能依靠计算平面的驱动提供信息来决定切换到其他工作模式，而是应该重启系统重置策略。目前 SATA 和 GMAC 控制器在禁用模式下的行为：

1) 寄存器通路：仅保证对硬件控制器的寄存器读写操作的完整性，不挂死内部 IO 总线，不保 证读数据的正确性，不保证写数据真正写到寄存器，不触发硬件行为（如发起 dma、置中 断、清中断等）。
2) DMA 通路：硬件不发起任何 dma 操作。
3) 中断通路：硬件不上报任何中断。
4) 硬件只保证内部通路不挂死，禁用模式下软件层次的握手完整性需要软件自己保证（比如：禁用模式下软件启动 DMA 传输，硬件会完成 IO 写操作，但不会发起 DMA，从软件的角 度看 DMA 传输肯定不会完成，此时软件不能死等 DMA 完成）。

1.6 初始流程说明

1.6.1 SATA 初始流程

1) 上电复位时处于明通模式。
2) 上电后由固件配置，SATA 控制器各端口的工作模式，之后 SATA 控制器各端口工作在对应 的模式下。

1.6.2 GMAC 初始流程

1) 上电复位时处于未配置模式（相当于禁用）。
2) 上电后由自引导固件配置 GMAC 控制器的工作模式，之后 GMAC 控制器工作在对应的模 式下。

1.6.3 USB 初始流程

1) 上电复位时处于未配置模式（相当于禁用）
2) 上电后由自引导固件配置 USB 各端口的工作模式，之后 USB 各端口工作在对应的模式下。

2 ICH2 芯片空间编址

2.1 拓扑结构

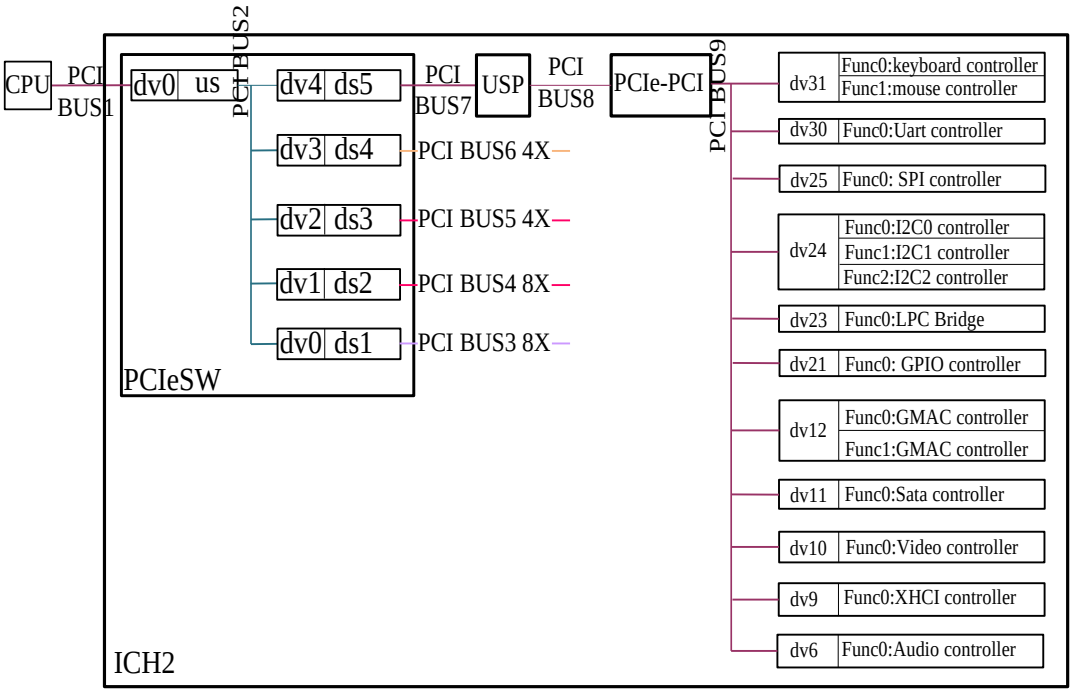


图 2-1　ICH2 PCI 拓扑

2.2 资源划分

表 2-1　PCI 资源划分

Bus:Device:Function	功能描述
Busx:Device0:Function0	PCIeSW 的上游端口
Busx+1:Device0:Function0	PCIeSW 的下游端口 0

Busx+1:Device1:Function0	PCIeSW 的下游端口 1
Busx+1:Device2:Function0	PCIeSW 的下游端口 2
Busx+1:Device3:Function0	PCIeSW 的下游端口 3
Busx+1:Device4:Function0	PCIeSW 的下游端口 4
Busx+6:Device0:Function0	内部 USP 端口
Busx+7:Device0:Function0	PCIe-PCI 桥
Busx+8:Device31:Function0	PS2 键盘控制器
Busx+8:Device31:Function1	PS2 鼠标控制器
Busx+8:Device30:Function0	Uart0 控制器
Busx+8:Device25:Function0	SPI 控制器，接显示屏或 FLASH
Busx+8:Device24:Function0	I2C 控制器 0
Busx+8:Device24:Function1	I2C 控制器 1
Busx+8:Device24:Function2	I2C 控制器 2
Busx+8:Device23:Function0	LPC Bridge
Busx+8:Device21:Function0	GPIO 控制器
Busx+8:Device12:Function0	千兆以太网控制器 0
Busx+8:Device12:Function1	千兆以太网控制器 1
Busx+8:Device11:Function0	SATA 硬盘控制器
Busx+8:Device10:Function0	显示控制器
Busx+8:Device9:Function0	USB XHCI 控制器
Busx+8:Device6:Function0	AC97 音频控制器

2.3 ICH2 芯片 PCI 空间定义

- 支持系统对设备的 12/16 位地址 Legacy IO 访问，粒度为 1B、2B、4B；
- 支持系统对设备的 64 位地址 MMIO 访问，读请求粒度为 1B、2B、4B、8B（只应用于显 存）；写请求粒度为 1B、2B、4B、8B~64B（只应用于显存）；
- 不支持带空洞的 IO/MMIO 访问，如4B 访问字节使能为 1001 或 1010 或 1011 或 0101 或 0110 或 1101 或 1110；
- 支持设备对主存的 32 位地址 DMA 访问；
- ICH2 芯片内部实现的 PCI 设备占用系统 2.5GB+16MB 可预取 MEM 空间；
- 套片 PS2、uart、VGA 兼容，以及 LPC 总线使用的 IO 空间属于 PCI Legacy IO 空间，不需 要实现 IO BAR，但需要在桥的 IO 基限范围内。

表 2-2 PCIe 64 位 BAR 地址空间定义

功能	空间大小及类型
PS2 键盘	8KB PRE-MEM
PS2 鼠标	8KB PRE-MEM
Uart0	8KB PRE-MEM

SPI	8KB PRE-MEM
I2C0 控制器	8KB PRE-MEM
I2C1 控制器	8KB PRE-MEM
I2C2 控制器	8KB PRE-MEM
LPC 控制器	BAR0：64KB PRE-MEM，LPC_HOST-IO BAR1：256MB PRE-MEM，LPC_memory-IO BAR2：256MB PRE-MEM，LPC_FW-IO
GPIO 控制器	8KB PRE-MEM
GMAC0 控制器	512KB PRE-MEM
GMAC1 控制器	512K B PRE-MEM
SATA 控制器	1MB PRE-MEM
显示控制器	BAR0：2GB PRE-MEM
	BAR2：4MB PRE-MEM，给软件约定： Address[21:19]=000 是 MC 的寄存器 Address[21:19]=001 是 VPU 的寄存器 Address[21:20]=01 是 GPU 的寄存器 Address[21:20]=10 是 DC 的寄存器 Address[21:20]=11 是 DMA 的寄存器
AC97 控制器	512KB PRE-MEM
USB XHCI 控制器	1MB PRE-MEM
PCIe-PCI 桥	512KB PRE-MEM

3 ICH2 芯片内设备

3.1 PCIe Switch

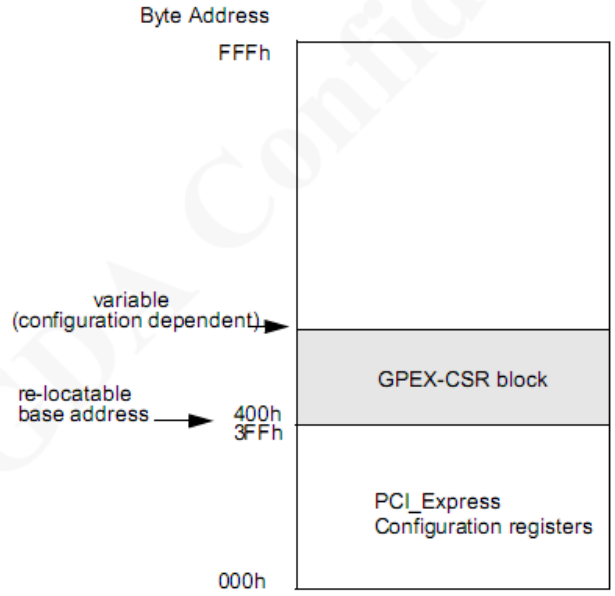
ICH2 中的 PCIE switch 转发 Host 给下游设备的 IO 和配置请求以及 DMA 响应，和下游设 备给上游设备的 DMA 请求和配置及 IO 响应。同时转发 PCIE 协议规定的各类中断包及电源管 理等消息包。

3.2 Switch 中的寄存器

Switch 中对软件可见的寄存器为 PCIE 配置寄存器。Switch IP 还提供 Device Specific Register-CSR，这些寄存器缺省只对硬件可见，但通过配置 PCIE 扩展能力寄存器 Vendor Specific Extended Capability(VSEC)，也可以使这些寄存器对软件可见。

寄存器详细说明请参考寄存器手册。

Figure 2 : PCI_Express Config Address Space



3.2.1 PCIE 配置寄存器

PCIE 配置寄存器是 PCIE 系统和软件的最主要接口，软件通过读写 PCIE 协议规定的配置 空间寄存器进行系统枚举及各种 PCIE 协议规定的操作。

3.2.2 CSR 寄存器

CSR 寄存器为 GPEX Switch 提供的一组特定寄存器，主要用于硬件对 Switch 的控制、观 测及调试，一般情况下不需要软件对其进行操作。

3.2.2.1 软件对 PCIE Switch 的相关操作

以下为软件对 PCIE 系统的主要操作。

3.2.2.2 系统枚举

系统枚举流程为标准的 PCIE 系统枚举流程，对 PCIE Switch 来说主要是配置上下游端口的 源级、次级总线号、基址、界限寄存器等内容。

3.2.2.3 电源管理

PCIE Switch 硬件支持 PCIE 规范规定的链路电源管理（提供电源管理能力、状态寄存器给 软件）及传输下游设备进入低功耗状态和唤醒所需要的 PME 消息（提供 PME 消息的路由）。

3.2.2.4 设备热拔插

对于外部下游端口，PCIE Switch 硬件支持 PCIE 规范规定的热拔插功能（提供热拔插能力 寄存器给软件）及根据热拔插事件产生热拔插中断消息的能力。

3.2.2.5 PCIE 错误信息处理

PCIE Switch 硬件支持 PCIE 规范规定的错误处理能力及状态寄存器，并实现错误消息的报 告，软件负责根据这些错误消息及错误状态寄存器的内容进行错误处理。

为了使 switch 的各个端口在发生“不可纠正非致命错”（ERR_NON_FATAL）或“不可纠 正致命错误”时能够产生错误消息发送给 CPU，必须使能的寄存器有：

1) 所有端口 Command Register 的 SERR 位（bit8）必须被置为 1；
2) 所有端口 Bridge Controll Register 的 SERR 位（bit1）必须被置为 1；
3) 所有端口 Device Controll Register 的 Non_Fatal_Error_Reporting_Enable （bit1）位和 Fatal_Error_Reporting_Enable（bit2） 位必须被置为 1。

PIU 会将这些错误消息转换成中断发送给 CPU。

3.3 PCIe-AMBA 桥

PCIe-AMBA 桥是实现 ICH2 芯片 PCI 拓扑，实现 ICH2 芯片所有设备 PCI 功能框架，实现系统对 ICH2 芯片所有设备的初始化、枚举和驱动，实现 ICH2 芯片所有设备中断，以及实现 PCIe 和 AMBA 两大部件间的协议转换的关键部件。PCIe-AMBA 桥实现以下功能：

1) 实现系统对 ICH2 芯片设备的 IO 访问 系统对 ICH2 芯片设备 IO 访问请求基于系统 MMIO 空间，请求被 PCIe-AMBA 桥认领，

PCIe-AMBA 桥将请求转换成 AHB 总线空间上的总线请求后发到 AHB 总线，由相关设备认领并返 回响应，PCIe-AMBA 桥接收响应并转换成 PCIe 完成事务返回给系统。

2) 实现设备对系统主存的 DMA 访问

ICH2 芯片各设备对系统主存的访问请求基于系统空间，这些请求在 AXI 总线上都被路由到 PCIe-AMBA 桥，桥接收请求并转换成基于系统主存空间的 PCIe MEM 事务发到系统主存。

3) 实现设备对系统的中断

PCIe-AMBA 桥中实现 ICH2 芯片设备中断控制器，用于各个设备对系统中断请求的集中提交 和处理。

语义上，所有挂接 ICH2 芯片 AMBA 总线上的需要由系统驱动和使用的 IO 设备都映射为 PCIe-PCI 桥次级总线上的 PCI 设备，最多 32 个虚 PCI 设备，这些虚设备对应了具体的 AMBA 总线上的 IO 设备。

3.3.1 桥的宏观特性

- 实现设备的 PCIe 拓扑结构
- 实现 PCIe2.0 8X 上游通道
- 实现高效低延时的 PCIe 总线和 AMBA 总线间的协议转换
- 实现系统对设备的配置访问
- 实现系统对设备的 64 位地址 MIO 访问，支持系统对显示控制器 2GB 显存的 MMIO 访问， 设备的 IO 空间定义在系统 PCIe 64 位 pre-MEM 空间
- 实现虚拟的 PCIe-PIC 桥
- 实现 PCIe 序

- 实现 AMBA AXI1.0 总线 Salve 接口
- 实现 AMBA AHB2.0 总线 Master 接口
- 实现设备多个相同 ID 的 AXI 总线读写事务的保序
- 实现设备对系统的 MSI 和 INTx 中断请求，实现可编程的中断优先级
- 实现设备对系统主存的 32 位地址 DMA 访问
- 实现 Legecy IO
- 支持系统 Max_Payload_Size: 128/256Bytes
- 支持系统 Max_Read_Request_Size: 128/256Bytes
- 支持系统的 PCIe 根联合体的两种 RCB：64B、128B，支持两种 RCB 下的 DMA 读请求的 多完成包
- 实现完备的错误处理
- 实现维护调试

3.3.2 桥的 IO 操作流程

IO 操作指由系统发起的对 ICH2 芯片设备的 IO 读写请求，对于设备而言，这些请求包括 对设备 PCI 配置空间的配置访问请求和对设备其它 IO 空间的 IO 访问请求。系统对设备的 IO 请求流程是：

1) CPU 核心流出 IO 访问指令至系统接口，系统接口根据请求的地址和读写类型将请求转换并生成 PCIe IO/MEM/CFG 请求 TLP 包的各个要素，发到 PCIe 总线并由 PCIe-AMBA 桥接收；
2) PCIe-AMBA 桥根据请求地址访问具体 IO 空间，配置请求转发给 DVn_CFG 模块，IO 请求 转发给 PCQ 模块，所有组建完成 TLP 需要的 TLP 信息都带在请求中；
3) DVn_CFG 模块收到配置请求，读写相应设备的配置寄存器，读写响应组成配置请求完成事务，通过四选一仲裁器向 PCIe-AMBA 桥发出给 CPU；
4) PCQ 模块收到 IO 请求，将 PCIe TLP 相关信息写入 PIO 请求悬挂缓冲，根据请求地址索引 AMBA 地址路由表，组建 AHB 请求包发向 AHB 总线接口；
5) AHB 总线接口将 AHB 请求包发向 AHB 总线，对于写请求，AHB 总线接口直接向 PAK 返 回写完成；
6) AHB 设备向桥 AHB 接口返回读响应，AHB 总线接口向 PAK 返回读完成；
7) PAK 生成 PIO 读写完成事务，释放 PIO 请求悬挂缓冲条目；通过四选一仲裁器向 PCIe-AMBA 桥发出给 CPU。

3.3.3 桥的 DMA 操作流程

DMA 访问指由 ICH2 芯片设备发起的对系统主存的 DMA 读写请求，设备 DMA 请求流程是：

1) 设备根据需求（CPU 对设备填写描述府，启动），组织对 PCIe-AMBA 桥的 AXI 读写请求，请求地址为系统空间；
2) PCIe-AMBA 桥的 AXI 接口接收请求，将请求发给 ACQ，带上组建 AXI 读响应的所有必须 信息，对于写请求直接向 AXI 总线返回写响应，AXI 设备接收写完成，后续操作；
3) ACQ 收到请求，将 AXI 包相关信息写入 DMA 请求悬挂缓冲，索引为 PCIe 总线 8 位事务 TAG，根据请求地址索引 AMBA 地址路由表，组建存储器读写请求 TLP，包括请求地址、 长度、请求 TAG、源方 PCIe ID 等各个要素，经过四选一仲裁器发向 PCIe-AMBA 桥接口；
4) PCIe-AMBA 桥接口接收 TLP 请求，经过事务类型和 BAR 规则过滤，判定为合法存储器读 写事务，则将请求 TLP 包通过 PCIe 总线发到 CPU 主存；
5) 主存返回读响应，由 PCIe RC 经过 PCIe MEM 读完成事务发给 PCIe-AMBA 桥；
6) PCIe-AMBA 桥接口将 MEM 读完成事务发给 AAK，AAK 根据 PCIe 事务 8 位 TAG 索引 DMA 请求悬挂缓冲读出对应请求的 AXI 相关要素，释放悬挂缓冲条目，形成多个 AXI 读响应发 向 AXI 接口；
7) AXI 接口将读响应发到 AXI 总线；设备接收 DMA 读响应，进行后续操作。

3.3.4 桥的配置寄存器定义

寄存器详细说明请参考寄存器手册。

3.3.5 桥的中断流程

1) 设备产生中断请求，以中断线的形式，向 PCIe-AMBA 桥发出；
2) PCIe-AMBA 桥中断处理模块接收各设备中断请求，查询各设备配置空间 Interrupt Pin 寄存 器和各 MSI 使能寄存器，确定 PCIe 提交方式: INTx 中断消息包或 MSI 中断消息包；

3) 中断处理模块对于以 MSI 发出的这类设备设置 MSI 中断开关寄存器，该寄存器硬件在发出MSI 时置 1，软件在清设备中断的同时对它清 0。为了避免多发中断，软件在清设备中断后， 再读一次设备中断，然后再清该设备的 MSI 开关；

4) PCIe-AMBA 桥接口收到中断处理模块来的 INTx 中断消息包或 MSI 中断消息包，通过 PCIe 总线发到 CPU；

5) CPU 的 PCIe 接口中断处理部件收到中断请求后，根据中断方式读取相应的中断核心使能寄 存器，获取该中断请求对应的服务器 CPU 核心，生成外部中断请求包，发向对应核心，进 入中断处理程序；

6) 中断处理程序最初通用程序通过 IO 读获得外部中断的具体类型；

7) 对于 MSI#中断，需要对核心核心中断向量 CSR 寄存器读获得设备中断的类型（能知道那个

设备的那类中断），如果需要（比如更详细的中断信息）再读各设备相关中断源寄存器的 IO 读，获得 MSI#中断的具体类型；

8) 对于 INTx 中断，需要基于 PCI INTx 中断提交策略，对各设备相关中断源寄存器做 IO 读， 获得 INTx 中断的具体类型；

9) 进入相应的设备中断处理程序，处理完毕后清除相关设备中断源寄存器，清除 MSI 中断提 交开关寄存器和 INTx 中断撤销寄存器；

10) 中断处理正式完成。

3.4 AC97 控制器

3.4.1 可编程寄存器

寄存器详细说明请参考寄存器手册。

3.4.2 AC97 配置

3.4.2.1 初始配置

上电后要求 wishbone 复位信号（wb_rst_i）和 AClink 复位（ac97_reset_pad_on）同时有效， 这样才能保证控制器的正常工作。

推荐每次使用前先软件检查，使用后软件关闭。检查 CORE_STATUS 寄存器，保证软件和 控制器的硬件间没有差别，否则音频或其他部分会出现不可预测的系统错误。

有三个寄存器需要进行根据系统时钟和 AClink 时钟比进行 timing 的计算。这些都是综合前 定义的，也可以软件重新配置。

软件通过使用 CORE_STATUS 寄存器释放 CODEC 复位（ac97_rst_pad_on）：

- 检测 AC97 控制器的活动状态（CORE_STATUS 31 位）
- 设置 timing 寄存器（COLD_CNT, WARM_CNT, SUSPEND_CNT）

- 释放 AC link 复位（CORE_STATUS 31 位）

AC97 控制器初始化

总线操作（R/W）	AC97 控制器寄存器名	地址(BMC)	读期望数据	写数据
R	AC97_CORE_STATUS	0x5070_0000	Data0[31:30] !=2‘b0	
W	AC97_CORE_STATUS	0x5070_0000		Data0 | 32'h08000000
R	AC97_INT_MASK	0x5070_0010	32'h00000000	
R	AC97_INT_DMA_MASK	0x5070_0018	32'h00000000	
R	AC97_INT_DMA_TRIGGER	0x5070_001c	32'h00000000	
R	AC97_COLD_CNT	0x5070_0024	32'h00000000	
R	AC97_WARM_CNT	0x5070_0028	32'h000000F5	
R	AC97_SUSPEND_CNT	0x5070_002c	32'h00000009	
R	AC97_REVISION_READ	0x5070_0030	32'h00004535	
	重复上述操作直到 AC97_CORE_STATUS		AC97_CORE_STATUS [31:30] ==2‘b00	
W	AC97_COLD_CNT	0x5070_0024		32'h00000064
W	AC97_WARM_CNT	0x5070_0028		32'h00000064
W	AC97_SUSPEND_CNT	0x5070_002c		32'h00000009
W	AC97_CORE_STATUS	0x5070_0000		32'h00000000

3.4.2.2 CODEC 初始化

上述复位完成后，将触发 AC97 CODEC 开始位时钟的生成，AC97 控制器生成 AClink 的同 步信号（ac97_sync_pad_o）。所有的传输都将停止，直到 CODEC_STATUS 的 CODEC_READY 位有效。

如果驱动或软件是依赖于 CODEC 的硬件，推荐软件检测主、第二 CODEC 的 vendor ID 寄 存器。否则软件将检测 CODEC 的功能，并决定如何使用 AC97 2.3 兼容的 CODEC。

随后，用户可设置 CODEC 寄存器的合适值（音量、音调、gain、通用寄存器、采样频率）。在使能 AClink 通道前，软件必须检测 CODEC subsection ready 位（CODEC 电源关闭/状态寄存器）。

- 检测 AC97 CODEC ready 信号（CORE_STATUS bit 28）

- 检测 CODEC vendor ID 或 CODEC 功能
- 设置 CODEC 寄存器
- 检测 sub-section ready 信号

AC97 Codec Read 操作

总线操作（R/W）	AC97 控制器寄存器名	地址(BMC)	读期望数据	写数据
R	AC97_CORE_STATUS	0x5070_0000	Data[31]= =1‘b0 即	
			ac97_rst_pad_on=b0 如上述条件不成立，反复读	
W	AC97_CODEC_READ	0x5070_0004		Data[6:0] Codec Read Addr
R	AC97_CODEC_STATUS	0x5070_000C	Data[30]= =1‘b1 即 CODEC register read data ready，如上述条件不成立，反复读 Data[22:16] Codec Read ehco Addr Data[15:0] Codec Read Data	

AC97 Codec Write 操作

总线操作（R/W）	AC97 控制器寄存器名	地址(BMC)	读期望数据	写数据
R	AC97_CORE_STATUS	0x5070_0000	Data[31]= =1‘b0 即 ac97_rst_pad_on== 1‘b0 如上述条件不成立，反复读	

W	AC97_CODEC_WRITE	0x5070_0008		Data[6:0] Codec Write Addr Data[31:16] Codec Write Data

AC97 Codec 初始化

总线操作（R/W）	Codec 寄存器名	Codec 地址	读期望数据	写数据
R	VENDOR_ID1	6'h7c	16‘h414c	
R	VENDOR_ID2	6'h7e	16‘h4760	
R	MASTER_VOL	6'h02	16'h8000	
R	MONO_VOL	6'h06	16'h8000	
R	EXT_AUD_ST_CTRL	6'h28	16'h09c4	
R	PWR_CTRL_STAT	6'h26	Data[3:0] ==4'hf，如上述条件不成立，反复读	

3.4.3 基本操作

AC97 控制器可在 WISHBONE 总线上作从模式、或主模式（DMA）。

由于 AC97 2.3 标准不支持单声道音频数据流，因此左右声道是同时使能的。其他的音频通 道是在综合前就定义好的，有软件进行后续配置。

下文介绍如何在两种模式下进行 AC97 控制器的配置。

3.4.3.1 从模式传输

如果使用从模式，用户（软件）必须初始为所已使能的音频通道填充音频数据至嵌入式存储器中。设置 INT_MASK 和 AUDIO_ENABLE 后，将启动音频数据传输。如有中断请求，用户需要读 INT_SOURCE 寄存器，确定是哪个通道引起的中断，在何地址边界内（offset/wrap），（从偏移量开始/地址回环），并填充相应音频通道嵌入式存储器的一半。

- 嵌入式存储器填充
- 更新 INT_MASK 寄存器
- 更新 AUDIO_ENABLE 寄存器
- 中断时，填嵌入式存储器的一半

3.4.3.2 从模式放音流程

- 驱动先通过 Memory IO 写的方式将放音数据发到 AC97 控制器，控制器将数据填入嵌 入式存储器，直到填满 64 个存储器条目。
- 驱动设置 INT_MASK 和 AUDIO_ENABLE 寄存器，打开使能信号，清除中断屏蔽。
- 控制器开始启动音频数据传输，从嵌入式存储器不断读取数据发到 AClink，直到放音 过程结束，即 AUDIO_ENABLE 寄存器的使能被驱动关闭。
- 在从嵌入式存储器读取数据放音的过程中，控制器判断每个声道的缓冲数据是否读到一半，如果读到一半对应通道则发出 INT_SOURCE 中断。
- 驱动根据 AC97 控制器发到系统的中断信号或中断包，通过 Memory IO 读的方式读取 INT_SOURCE 寄存器，确定是哪个放音声道的 offset 中断还是 wrap 中断。首先收到的 是 offset 中断，驱动通过 Memory IO 写的方式将对应通道的嵌入式存储器的前一半空 间进行装填数据，驱动发完一半的装填数据会将 INT_SOURCE 的对应中断清除。
- AClink 通路一直在读取存储器数据发送，读到末尾条目的数据，控制器再次发出 INT_SOURCE 中断，即 wrap 中断，通知驱动可以启动后一半数据的装填，过程同 offset 中断。
- 上述的放音过程从 AUDIO_ENABLE 启动后就一直不停，循环重复控制器发出 INT_SOURCE 中断，驱动装填数据并清除中断的过程，直到放音任务结束，驱动置 AUDIO_ENABLE 寄存器为关闭使能，并清除 INT_MASK 寄存器的中断。

3.4.3.3 从模式录音流程

- 驱动设置 INT_MASK 和 AUDIO_ENABLE 寄存器，打开使能信号，清除中断屏蔽。
- AClink 开始读入录音数据，并填入对应通道的嵌入式存储器缓冲，只要 AUDIO_ENABLE 没有关闭，就一直不间断的进行读取数据和填充操作。
- 录音通道的嵌入式存储器填到一半，AC97 控制器登记 INT_SOURCE 寄存器对应类型 中

断，即哪个通道的 offset 中断，并发出中断请求通知系统。

- 驱动根据中断类型读取 INT_SOURCE 寄存器，用 Memory IO 读的方式读取 AC97 嵌入式存储器中的前一半数据，读完后清对应中断。
- 嵌入式存储器的后一半缓冲填充完毕，同样控制器登记 INT_SOURCE 寄存器对应中断，即 wrap 中断，发出中断请求通知系统。
- 驱动的处理方式类似 offset 中断，不同的是读取的是存储器的后一半数据。
- 录音的任务完成，驱动置 AUDIO_ENABLE 寄存器对应通道使能关闭，清除 INT_MASK 寄存器的中断。

3.4.3.4 主（DMA）模式传输

对每一个激活的 AClink 音频通道需要进行多个域/寄存器的设置，由 Master 的 DMA 模块进行处理。

更新 STREAM_DESC 地址的 CHANNEL_DESC 为一个通道的 stream descriptor 数据。这些动作只针对配置了 DMA 触发器（INT_DMA_TRIGGER）的音频通道。

为每一个通道更新 STREAM_DESC 表的 buffer 基地址、长度、stream number 和 channel number、数据大小。对任何 stream，有多少个音频通道就有多少个 STREAM_DESC 条目。当收到启动 DMA 引擎的中断时，需要使用这个表来译码 stream 并正确的填写嵌入式存储器缓冲。设置 INT_DMA_MASK 来使能相应的音频通道的 DMA 引擎，该引擎将处理数据传输。这还将阻挡触发中断的信号 int_o（INT_SOURCE），并首次填充输出通道的嵌入式存储器的缓冲。设置 INT_DMA_TRIGGER 寄存器来将被中断请求使能 DMA 引擎，如果使用多通道音频数据流，只置最后一个通道的中断位。

举例来说：发送给前置左右通道的单声道的数据流将置 INT_DMA_MASK 和 INT_DMA_TRIGGER 的 [1:0] 位为 1，而立体声会设置 INT_DMA_MASK 的 [1:0] 位和 INT_DMA_TRIGGER[1]。

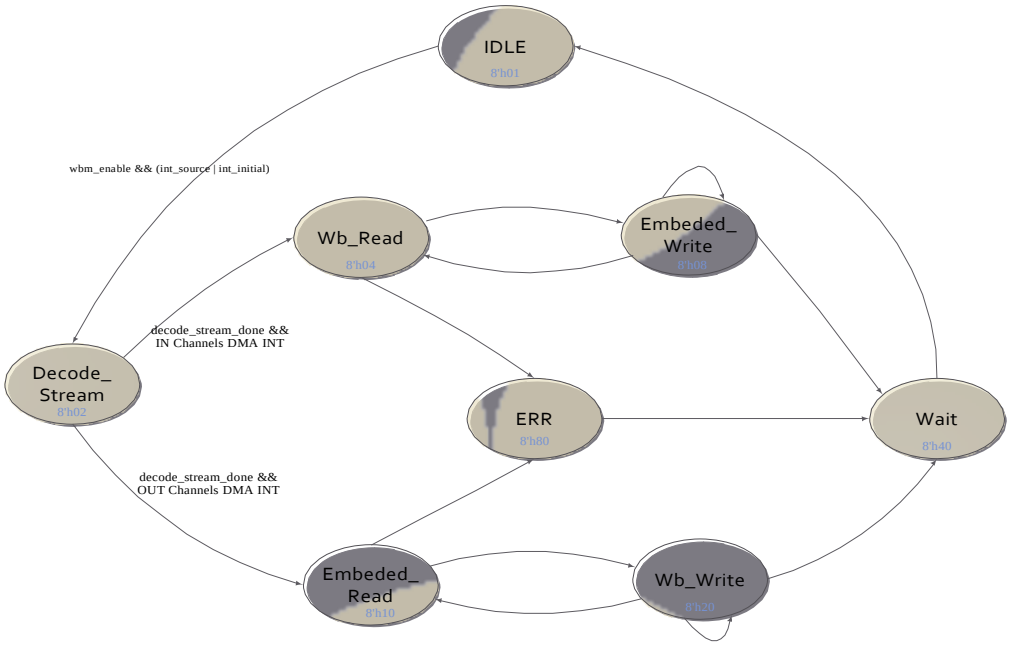
AClink 的最后一个通道触发的 DMA 传输，使能 AC97 控制器进行单或多声道交叉的音频数据流传输。

设置 AUDIO_ENABLE 启动音频的 playback/recording

- 更新 CHANNEL_DESC
- 更新 STREAM_DESC
- 更新 INT_DMA_MASK

- 更新 INT_DMA_TRIGGER
- 更新 AUDIO_ENABLE DMA

控制状态机的流程图：



3.4.3.5 主模式放音流程

- 驱动填好 CHANNEL_DESC、STREAM_DESC、INT_DMA_MASK 寄存器，准备好放音的 DMA 操作所需要的描述符信息，并清除 DMA 中断屏蔽。
- 驱动置 INT_DMA_TRIGGER 寄存器使能，以及 AUDIO_ENABLE 寄存器相应通道使 能，允许 AC97 控制器发起 DMA 请求和发送放音数据。
- AC97 控制器监测到 dma trigger 打开后，就根据通道描述符和流描述符寄存器的信息 发起 DMA 读请求，接收到的 DMA 读响应数据不断的填入嵌入式存储器缓冲。AClink 也不断的从嵌入式存储器缓冲读取数据发到 CODEC 芯片进行放音。
- DMA 读响应取得的数据先填充嵌入式存储器的一半，填满一半后会继续发起 DMA 读 请求，直到填满全部的存储器条目。
- 当前描述符对应长度的 DMA 读请求完成后，AC97 控制器会登记 INT_SOURCE 寄存 器对应通道的 offset 中断，并发出中断包通知系统。
- 驱动根据中断信息，读取 INT_SOURCE 寄存器，重新填写一轮流描述符，驱动填完描 述符后，清除 INT_SOURCE 寄存器对应的中断位。
- DMA 读继续进行，描述符对应长度的数据再一轮都被控制器收下，控制器登记

INT_SOURCE 寄存器对应通道的 wrap 中断，并发出中断包通知系统。

- 类似 offset 中断，驱动进行处理。
- 直到放音任务结束，驱动置寄存器 AUDIO_ENABLE 关闭放音使能，INT_DMA_TRIGGER 寄存器关闭 DMA 请求发起，清除描述符寄存器等。

3.4.3.6 主模式录音流程

- 驱动填好 CHANNEL_DESC、STREAM_DESC、INT_DMA_MASK 寄存器，准备好放音的 DMA 操作所需要的描述符信息，并清除 DMA 中断屏蔽。
- 驱动置 INT_DMA_TRIGGER 寄存器使能，以及 AUDIO_ENABLE 寄存器相应通道使 能，允许 AC97 控制器发起 DMA 请求和允许从 AClink 接收录音数据。
- AC97 控制器从 AClink 总线上收取录音数据，并填入嵌入式存储器缓冲，只要 AUDIO_ENABLE 寄存器没有关闭，就一直不断的收取录音数据。
- 当嵌入式存储器缓冲的条目填到一半时，控制器启动 DMA 传输，根据通道描述符和 流描述符的信息发起 DMA 写请求到主存。
- DMA 写请求将存储器缓冲中的一半条目的数据都发到总线上。
- 当嵌入式存储器缓冲的条目全部填完，控制器继续启动延续上一轮的 DMA 传输，将 剩下一半的数据通过 DMA 写发到主存。
- 重复上面的前一半和后一半数据分别通过 DMA 写发到总线的过程，直到流描述符要 求传输数据的 DMA 写都发到总线，控制器会置 INT_SOURCE 寄存器对应通道 offset 中断，并中断通知系统。
- 驱动收到中断包后查询 INT_SOURCE 寄存器，更新流描述符寄存器，准备好后续DMA 的描述符信息后清中断。
- 继续重复 DMA 写，当前描述符要求的数据再一轮发到总线，控制器会置 INT_SOURCE 寄存器对应通道 wrap 中断，并通知系统。
- 同样，驱动重复类似 offset 中断的处理流程。
- 当全部的录音任务结束，驱动置寄 存器 AUDIO_ENABLE 关闭放音使能，INT_DMA_TRIGGER 寄存器关闭 DMA 请求发起，清除描述符寄存器等。 说明：无论是主模式的录音还是放音过程，流描述符指定的 DMA 搬运数据长度和各通道嵌入式存储器条目数的关系都是倍数的关系，所以 DMA 发起的时机和中断发起没有必然联系。简单说，DMA 读/写发起的时机都是嵌入式存储器缓冲写入/读出一半或全部的时刻，而中断发起的时机都是 DMA 请求完成描述符规定长度数据搬运的时刻。

3.5 GMAC 控制器

3.5.1 可编程寄存器

寄存器详细说明请参考寄存器手册。

3.5.2 DMA 传输流程

GMAC-DMA 包含独立的传送和接收 DMA 引擎，CSR 空间。传送引擎从系统主存向设备 端口（MTL）传输数据，接收引擎从设备端口向主存传输数据。控制器实现描述符，高效地在 源和数据间移动数据，最小化 Host CPU 的参与。DMA 设计用于面向包的数据数据传输，例如 以太网中的包。控制器可以通过编程中断 Host CPU，例如帧传送和接收完成，或者其他正常或错误的条件。

3.5.2.1 DMA 控制器数据结构

DMA 控制器和 Host 驱动程序的通信主要通过下面两个数据结构：

- 控制状态寄存器，Control and Status registers (CSR)
- 描述符和数据缓冲，Descriptor lists and data buffers

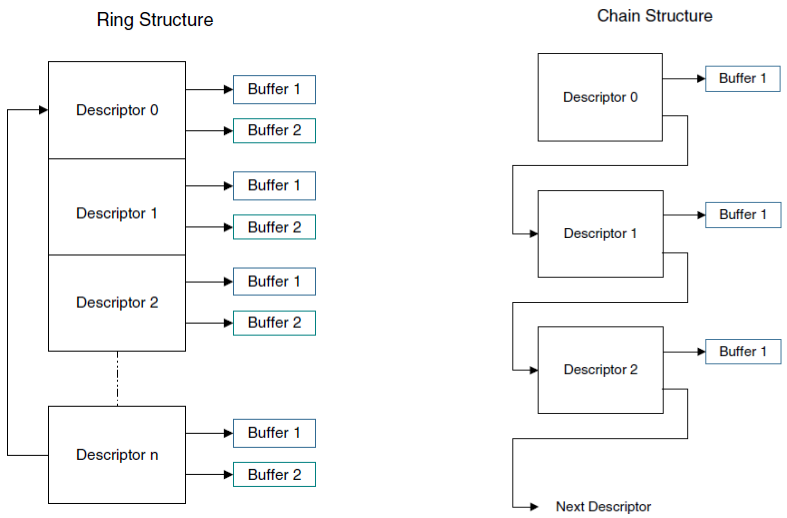
寄存器和的详细描述参考 DesignWare Cores Ethernet MAC Universal Databook, Chapter 5, Register. 描述符的详细描述参考 DesignWare Cores Ethernet MAC Universal Databook, Chapter 7, Descriptor.

DMA 发送的数据帧接收到主机主存的 Receive Buffer 中，而 DMA 接收的数据也来自主机 主存的 Transmit Buffer。驻留在主机主存中的描述符是指向这些缓冲的指针。

有 2 个描述符列表分别用于接收和传输。每个列表的基址寄存器分别写入 DMA Registers 3 和 4 中。描述符列表可以隐式的向前链接，最后的描述符可以往回指向第 1 个描述符从而实现 一个环结构。实现显示链描述符通过设置 RDES1[24]和 TDES1[24]（接收和发送描述符的第 2 个地址链接设置位）。描述符列表驻留在 Host 物理存储器地址空间。每个描述符最多可以指向 2 个缓冲，这

允许使用 2 个缓冲，物理地址寻址，而不只是主存中的连续缓冲。

下图示意了描述符的环和链结构：

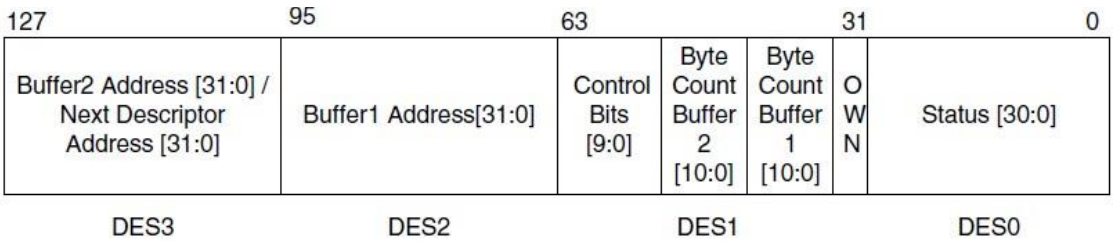


描述符列表驻留在 Host 物理存储地址空间中。每个描述符可以指向最大的 2 个缓冲。这可 以指向 2 个物理地址的 2 个缓冲，而不只是存储空间中的连续缓冲。一个主机物理地址空间中的数据缓冲有一个完整的帧或部分帧组成，但是不能超过一个帧。缓冲中只包含数据，缓冲的 状态保存在描述符中。指向帧的数据链跨越多个数据缓冲，但是一个描述符不能跨越多个缓冲。 DMA 检测到 EOF 时，将忽略下一个帧缓冲。可以使用也可以不使用数据链。

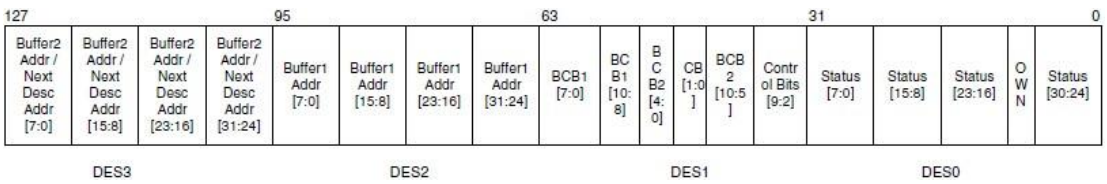
描述符根据模式和数据宽度的不同，存储的数据格式也有所不同，分为数据大小端模式 Little-Endian/Big-Endian ， 描 述 符 大 小 端 模 式 Reverse-Endian/Same-Endian ， 数 据 宽 度 有 32/64/128-bit。描述符的格式可以使用普通格式(16B)或可变/增强格式(32B)。

下面是 128 位数据总线的的描述符定义：

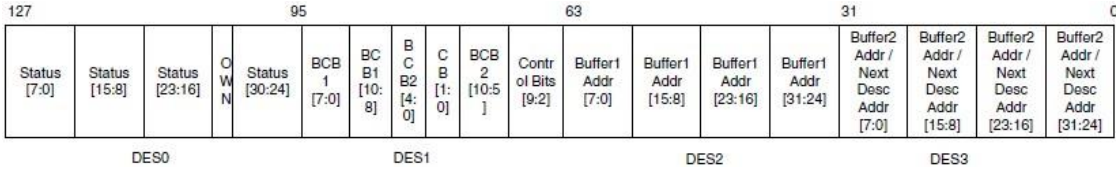
Little-Endian、Same-Endian 描述符：



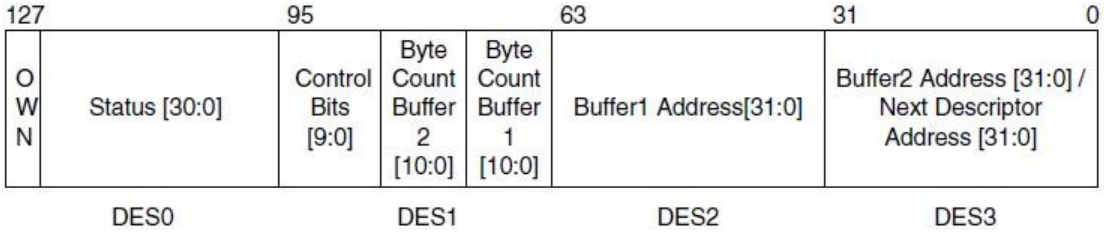
Little-Endian、Reverse-Endian 描述符：



Big-Endian、Same-Endian 描述符：



Big-Endian、Reverse-Endian 描述符：



3.5.2.2 DMA 初始化

1） 写 DMA Register 0（总线模式寄存器）设置主机总线访问参数。

2） 写 DMA Register 7（中断使能寄存器）屏蔽不必要的中断源。

3） 软件驱动创建发送和接收描述符列表。然后写 DMA Register 3（接收描述符列表地址寄存器）和 DMA Register 4（发送描述符列表地址寄存器），向 DMA 提供每个列表的起始地址。

4） 写 GMAC Registers 1, 2, and 3（MAC 帧过滤寄存器，HASH 表高位寄存器，HASH 表低位 寄存器）决定过滤的选项。

5） 写 GMAC Register 0（MAC 配置寄存器）配置和使能发送和接收操作模式。PS 和 DM 位需 要基于自动协商的结果（从 PHY 读到的结果）。

6） 写 DMA Register 6（操作模式寄存器）设置[13]位(ST，开始/停止传送)和[1]位(SR:，开始/停止接收)开始传送和接收。

7） 传送和接收引擎进入运行状态，并准备从接收描述符列表获得描述符。接收和传送引擎开始处理接收和传送操作。传送和接收处理是互相独立的，可以分别开始和停止。

3.5.2.3 DMA 普通模式传输

传输 DMA 引擎在缺省模式处理如下：

1） Host 建立传输描述符(TDES0-TDES3)并设置占有位(TDES0[31])在设置带以太网帧数据的对 应数据缓冲

2） 一旦设置了 ST 位(DMA Register 6[13])，DMA 进入运行状态

3） 当在运行状态，DMA 登记传输描述符列表用于帧请求传输。在登记开始后， 继续数据描述符环或者链顺序。如果 DMA 检测到描述符被 Host 标记了占有位，或者错误条件发生了， 传输被悬挂，所有的传输缓冲不可用(DMA Register 5[2])，设置正常中断汇总位(DMA Register 5[16])。传输引擎处理到第 9 步。

4） 如果请求的描述符属于 DMA (TDES0[31] = 1‘b1)，DMA 从请求描述符译码传输数据缓冲地 址。

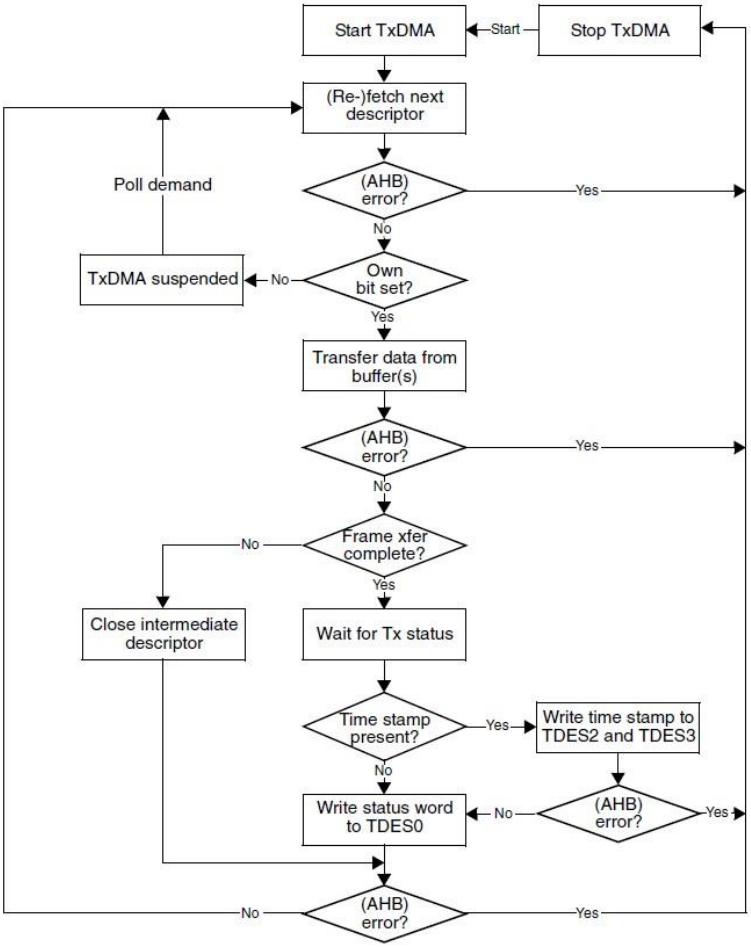
5） DMA 从 Host 主存取传输数据，并传输数据到 MTL 用于传输。

6） 如果以太网帧存储在数据缓冲在过个描述符中，DMA 关闭中间的描述符并取下一个描述符。重复第 3、4、5 步知道 EOF 数据传输给 MTL。

7） 当帧传输完成了，如果帧使能了 IEEE 1588 时间戳（标识在传输状态），从 MTL 得到的时 间戳值写入包含了 EOF 缓冲的传输描述符(TDES2 和 TDES3)。由于在这一步清除占有位， Host 占有这个描述符。如果这个帧没有使能时间戳，DMA 不改变 TDES2 和 TDES3 的内容。

8） 中断完成位(TDES1[31])设置在最后的描述符中，帧传输结束后设置传输中断位(DMA Register 5[0])，DMA 引擎然后返回第 3 步。

9） 在悬挂状态，当收到传输登记命令后，DMA 试着重新获得描述符（然后返回第 3 步），并 清除下溢中断状态位。



3.5.2.4 DMA OSF 模式传输

在 OSF(Operate on Second Frame, 第二个帧的操作)模式，传输 DMA 操作的运行状态如下 面的序列：

1） DMA 如果缺省模式 DxDMA 的第 1-6 步的操作

2） 不关闭前一个帧的最后描述符，DMA 去下一个描述符

3） 如果 DMA 占有请求的描述符，DMA 译码这个描述符中的传输缓冲地址。如果 DMA 不占 有这个描述符，DMA 进入悬挂模式并跳过第 7 步

4） DMA 从 Host 主存取传输帧，传输到 MTL 直到 EOF 被传输，如果这个帧分解在多个描述 符中关闭中间描述符。

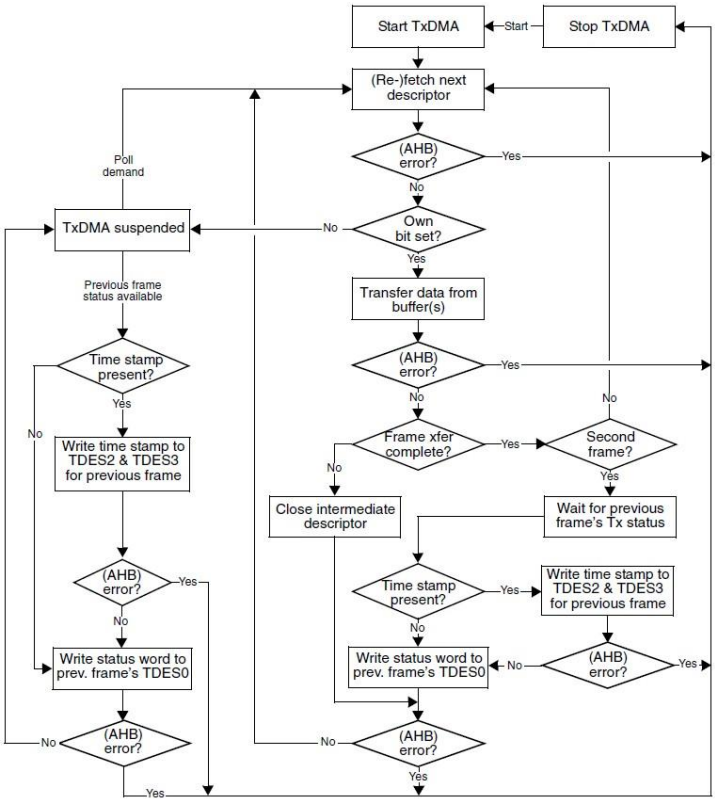
5） DMA 等待之前帧的帧传输状态和时间戳，一旦得到状态，DMA 写时间戳到 TDES2 和 TDES3，如果这样的时间戳可以得到（状态位描述的）。DMA 然后对对应的 TDES0 写状态， 清除占有位，

然后关闭描述符。如果前面的帧没有使能时间戳，DMA 不改变 TDES2 和 TDES3 的内容。

6） 如果使能，设置传输中断，DMA 取下一个描述符，然后处理到第 3步（当状态正常时）。 如果之前的传输状态显示下溢错误，DMA 进入悬挂模式（第 7 步）。

7） 在悬挂模式，如果从 MTL 收到悬挂的状态和时间戳，（如果使能了时间戳）DMA 写时间戳到 TDES2 和 TDES3，然后写状态到对应的 TDES0。然后设置相关的中断并返回悬挂模式。

8） 只有当收到传输登记命令(DMA Register 1)之后，DMA 才可以离开悬挂状态进入运行状态（根据悬挂状态进入第 1 步或第 2 步）。



3.5.2.5 DMA 接收

接收 DMA 引擎接收序列和处理过程如下：

1） Host 建立接收描述符(RDES0-RDES3)并设置占有位(RDES0[31]).

2） 一旦 SR 位(DMA Register 6[1])设置了，DMA 进入运行状态。在运行状态中，DMA 登记接 收描述符列表，以获得空闲描述符。如果取到的描述符不空闲（Host 占有），DMA 进入悬 挂状态并跳到第 9 步。

3） DMA 译码得到的描述符中的接收数据缓冲地址

4） 进入的帧处理和放置到得到的描述符数据缓冲。

5） 当缓冲满或帧传输完成时，接收引擎去下一个描述符

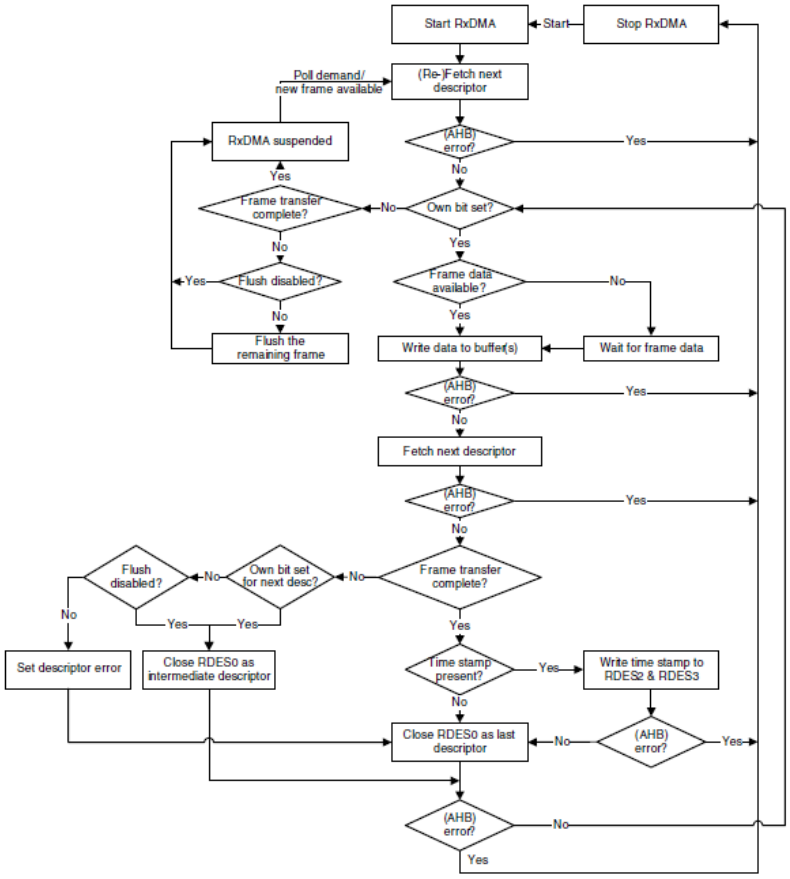
6） 如果当前帧传输完了，DMA 处理过程跳到第 7 步。如果 DMA 不占有下一个取到的描述符， 且帧传输没有完成（EOF 没有被传输），DMA 设置描述符错误位在 RDES0 中（除了刷新不 使能）。DMA 关闭当前描述符（清除占有位），通过清除最后段位（LS, Last Segment）在 RDES0 中的值标记它是中间的描述符，（如果刷新不使能，标志它是最后描述符，）然后跳 到第 8 步。如果 DMA 不占有下一个描述符但是当前帧传输没有完成，DMA 关闭当前描述 符作为中间的描述符，然后回到第 4 步。

7） 当 IEEE 1588 时间戳使能时，DMA 写时间戳（如果得到了）到当前描述符的 RDES2 和 RDES3。它然后从 MTL 得到接收帧状态并写状态字到当前描述符 RDES0，清除占有位， 设置最后段位。

8） 接收引擎检查最后描述符的占有位，如果 Host 占有描述符，接收缓冲不可用位（Register 5[7]）设置上了，DMA 引擎进入悬挂状态（第 9 步）。如果 DMA 占有描述符，引擎返回第 4 步等 待下一个帧。

9） 在接收引擎进入悬挂状态前，接收 FIFO 刷新部分帧（可以使用 DMA Register 6 第[24]位控 制刷新）。

10） 接收 DMA 退出悬挂状态当得到接收登记命令或者可以从 MTL 的接收 FIFO 中得到下 一帧的开始部分。引擎处理跳到第 2 步，并重取下一个描述符。



3.5.3 密通异常处理流程

在密通情况下，GMAC 会对不支持的发送帧类型和加解密数据错误两种情况报告异常中断。 当出现此类异常中断时，按照以下步骤进行处理：

1) 当 GMAC 控制器出现异常中断时，GMAC2PAB_cdint 信号保持为高；
2) 系统访问 GMAC 地址偏移为 0x304 的寄存器（异常中断寄存器），确定中断源；
3) 如果 0x304 寄存器的[0]或者[2]位为高，则直接对[0]或者[2]写 1 清除中断源，系统仍 然可以发起后续收发操作；如果 0x304 寄存器的[1]位为高，必须首先进行 GMAC 软 件复位（对地址偏移为 0x1000 寄存器的[0]位写 1）操作，并在重新 MAC 初始化和 DMA 初始化后恢复正常工作。

3.6 SATA 控制器

3.6.1 可编程寄存器

寄存器详细说明请参考寄存器手册。

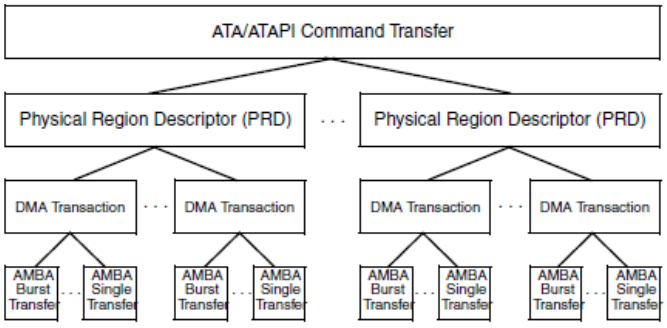
3.6.2 工作流程

基于以下规范:

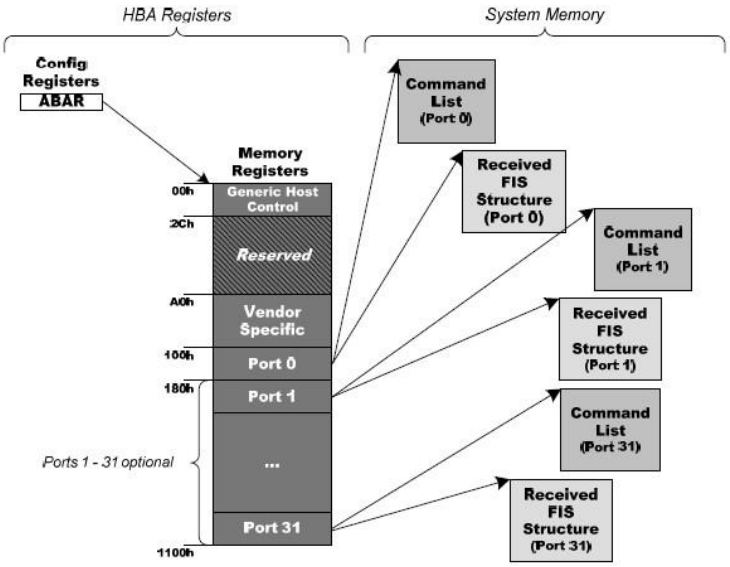
- ✧ DesignWare Cores SATA AHCI Databook, Chapter 2.5, Operation Details.
- ✧ Serial ATA AHCI 1.3 Specification.

3.6.2.1 数据结构

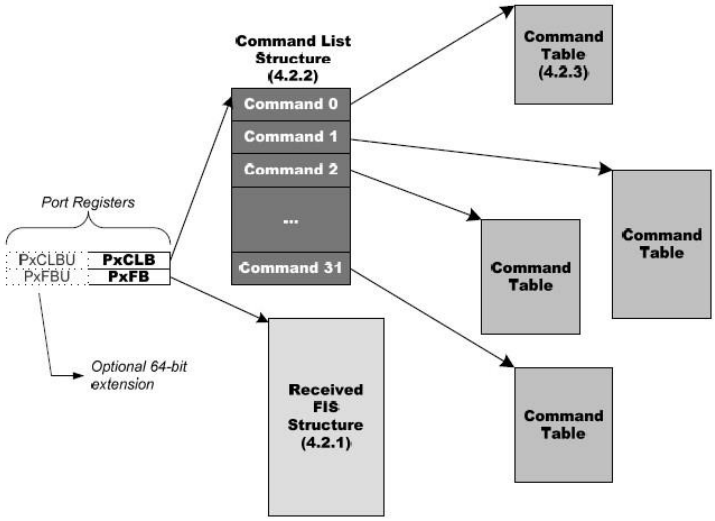
AHCI 数据传输层次如下图所示：



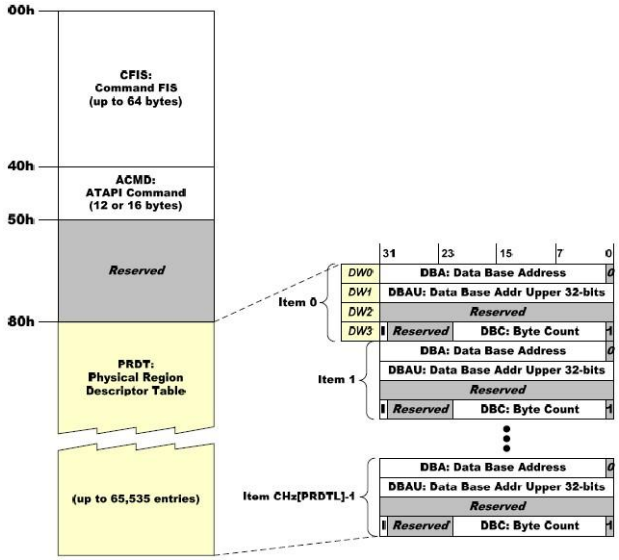
HBA 存储空间分配如下图所示：



端口存储空间分配如下图所示：



命令表结构如下图所示：



3.6.2.2 数据传输

3. 6. 2. 2. 1 ATA DMA 读

1） 软件通过读 P#CI 寄存器发现自由命令槽，然后在端口执行命令列表中建立 DMA 读命令， 并设置该命令槽的在 P#CI 寄存器中对应位。

2） PDMA 从系统主存中取命令头。

3） PDMA 从系统主存中取命令寄存器 FIS，并传送到设备。

4） 由于这是 DMA 读命令，设备对应一些数据 FIS。当 FIS 收到时，PDMA 执行下面的操作：

a. 从系统主存中取第一个 PRD。

b. 从 RX FIFO 传输数据到系统主存，直到传送完 PRD 中标记的字节数。

c. 继续取 PRD 并传输数据，直到传完命令需要的字节数。

5） 当 I-bit 设置上时，设备发送带命令结束状态的 D2H Register FIS 时产生中断。D2H Register FIS 被传到接受 FIS主存结构中。

6） 当这是最后一个命令，并且 DWC_ahsata 使能了主动电源管理，PDMA 请求链路层进入部 分睡眠或全部睡眠状态。

3. 6. 2. 2. 2 ATA DMA 写

1） 软件通过读 P#CI 寄存器发现自由命令槽，然后在端口执行命令列表中建立 DMA 写命令用 于端口执行，并设置该命令槽的在 P#CI 寄存器中对应位。

2） PDMA 从系统主存中取命令头。

3） PDMA 从系统主存中取命令寄存器 FIS，并传送到设备。

4） 设备响应 DMA 激活帧。当 FIS 收到时，PDMA 执行下面的操作：

a. 从系统主存中取第一个 PRD。

b. 传输数据从系统主存到 TX FIFO 直到满足 PRD 字节数或者到达 8-KB FIS 边界。链路层 发送数据 FIS 到设备。如果需要多个数据 FIS，设备对每个数据 FIS 发送 DMAActivate FIS

c. 继续取 PRD 并传输数据，直到传完命令需要的字节数。

5） 当 I-bit 设置上时，设备发送带命令结束状态的 D2H Register FIS 时产生中断。D2H Register FIS 被传到接受 FIS主存结构中。

6） 当这是最后一个命令，并且 DWC_ahsata 使能了主动电源管理，PDMA 请求链路层进入部 分睡眠或全部睡眠状态。

3.6.2.3 原始队列命令 (NCQ) 传输

DWC_ahsata 支持 NCQ 特征(读写 FPDMA 队列命令). 通过 DMA Setup FIS 激活数据传输， 通过 Set Device Bits FIS 执行命令完成操作。

3.6.2.4 PIO 传输

DWC_ahsata 支持多个 DRQ 块 PIO 操作(CAP.PMD=1)。从 DWC_ahsata 来看，PIO 操作看 起来像 DMA 传输：建立命令列表，数据从系统主存传输到 PDMA 模块。

3.6.2.5 传输大小

DWC_ahsata BIU 模块基于 PDMA 的传输请求产生对应的总线周期（突发/单个读或写）。 传输大小的设置通过 P#DMACR，使用 RXTS 和 TXTS 域（RX/TX 传输大小）对应 SATA 的接 受/写或发送/读。RXTS/TXTS 值的设置对应于 FIFO 的级别（in FIFO or bus-wide words，以 FIFO 或者总线宽度）：RX FIFO ae_level_d（almost empty_level destination，目标几乎空级别）和 TX FIFO af_level_s（almost full-level source，源几乎满）。

PDMA 使用对应的 FIFO 标记：RX FIFO almost_empty_d 和 TX FIFO almost_full_s 来向 BIU 产生 DMA 请求。在一些情况下，由于不能跨越不同的边界，传输大小可以小于 RXTS/TXTS 值。

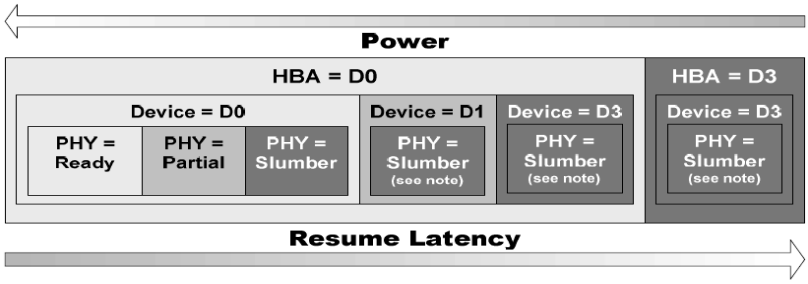
- AHB 总线的 1KB 地址边界
- AXI 总线的 4KB 地址或总线大小边界
- 数据帧信息结构 FIS 边界
- 物理区域描述符 PRD 边界
- 总线宽度边界（如果传输开始于非总线对界地址）

软件可以根据规范改变 RXTS/TXTS 值。通常最大传输大小是 FIFO 深度的一半，除了当 RX FIFO 的深度是 64 双字，这是受限于最小 af_level_s 值，这个值用于阻止 RX FIFO 由于 HOLD-HOLDA 延迟导致溢出。同样，在 AXI 总线环境中， RXTS/TXTS 值可以受限于 CC_MSTR_BURST_LEN 参数。在上电或者异步复位后，RXTS/TXTS 值设置成最大值。

DWC_ahsata 总线端的性能基本上由下面这些因素决定：

- 总线速度（hclk/aclk 频率）
- RX/TX FIFO 大小
- 传输/突发传输大小
- 端口数量
- 其他 master的数量（总线负载）

3.5.2.3 电源管理电源管理状态层次图：



DWC_ahsata 端口电源管理状态(PARTIAL/部分睡眠，或 SLUMBER/全部睡眠)可以通过软 件、端口自身，或设备进行初始化。电源状态机实现在链路层电源管理模块中。它驱动对应的 信号(p#_phy_partial or p#_phy_slumber)使 PHY 进入对应的电源管理状态。

软件通过使用 P#CMD.ICC 域请求进入部分睡眠或全部睡眠状态。然而，端口当链路层处 在 L_IDLE 状态才响应对应的操作，否则忽略这个请求。

设备通过传输 PMREQ_Pp 或 PMREQ_Sp 原语请求对应的端口进入电源管理状态。软件可 以通过设置 P#SCTL.IPM 域禁止端口进入电源管理状态。

DWC_ahsata 支持激进的电源管理状态，允许端口只要没有到设备的悬挂命令就初始化接口 电源管理状态。

可以通过 P#CMD.ASP 域控制端口初始化时进入部分睡眠还是全部睡眠状态， 通过 P#CMD.ALPE 位使能这个功能。当 P#CMD.ALPE=1，端口认为没有命令需要执行，端口根据 P#CMD.ASP 设置进入部分睡眠或全部睡眠状态。端口认为下面 2 种情况没有命令需要传输：

- P#SACT 被清为 0，P#CI 从非 0 值变为 0。
- P#CI 被清为 0，接受到 Set Device Bits FIS，P#SACT 从非 0 值变为 0。

DWC_ahsata 支持部分睡眠到全部睡眠状态的自动转换，这个特征通过软件设置 P#CMD.APSTE 位使能。当 P#CMD.APSTE=1，DWC_ahsata 链路不进入部分睡眠状态而协商部 分睡眠状态，然后核心自动转换进入全部睡眠状态，不考虑 Host 软件的、端口（激进）的、或 设备的初始化。

当下面任何一个条件成立时，电源管理状态结束：

- 软件写 P#CMD.ICC=1h 或 P#CI 写非 0 值，请求转换进入活动/Active 状态。
- 设备通过传输 COMWAKE OOB 请求接口唤醒。

接口状态（Active/活动, Partial/部分睡眠, Slumber/完全睡眠）反映在 P#SSTS.IPM 域。 电源管理事件：

当下面任何条件成立时，DWC_ahsata 输出 pme_req 有效：

- P#IS.PCS=1
- P#IS.DMPS=1（当包含 DEV_MP_SWITCH 时）
- P#IS.CPDS=1（当包含 DEV_CP_DET 时）
- P#IS.SDBS=1 并且设置了 P#SNTF.PMN 的任何位时（Set Device Bits FIS 接收到–Ⅱ或–NⅡ位 都设置为 1）。

pme_req 信号用于在基于 PCI 总线的系统上请求电源管理事件(PME#)。当软件清除对应的 P#IS 位时该信号置无效。例如，pme_req 信号可以通过某个电平（level）的同步逻辑连接到 DWC_pcie_ep 的核心输入 apps_pm_xmt_pme。

3.6.2.6 热插拔

热插拔包含了下面一些内容：

- 原始的热插拔
- 冷在位检测
- 机械在位切换**原始**

热插拔

DWC_ahsata 通过设置 P#SERR.DIAG_X 和 P#IS.PCS 位支持原始 SATA 热插拔。每次链路 检测到 COMINIT 序列时都设置这些位，表示热插插拔入或系统上电。热插拔移除通过检测端 口内部–PHY READY‖信号的改变，这事件反映在 P#SERR.DIAG_N 和 P#IS.PRCS 位。 **冷在位检测**

DWC_ahsata 可以通过设置参数 DEV_CP_DET 支持冷在位检测。设计上会增加 2 个信号接口：

- 输出信号 p#_cp_pod 使能设备的供电。
- 输入信号 p#_cp_det 检测增加或移除没有加电的设备。

P#CMD.POD 位控制 p#_cp_pod 信号的输出。P#CMD.CPS 位反映 p#_cp_det 的输入状态，P#IS.CPDS 位反映 p#_cp_det 状态的改变。当 P#IE.CPDE 和 GHC.IE 都设置时，产生中断信号 intrq。

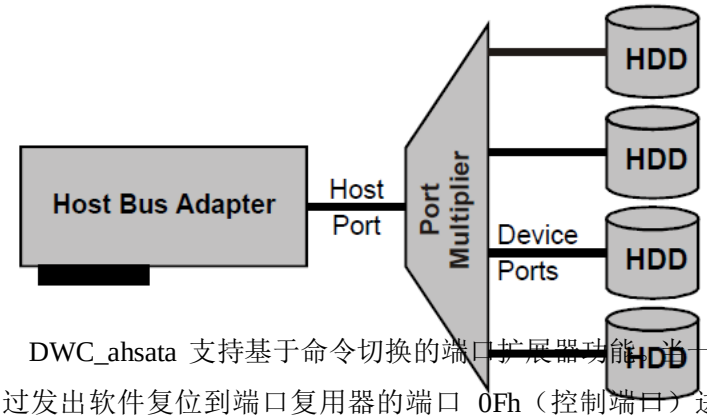
注意：平台必须有附加的硬件实现支持冷在位检测。 **机械在位切换**

DWC_ahsata 可以通过设置参数 DEV_MP_SWITCH 支持机械在位切换。设计上会增加 1 位 输入信号 p#_mp_switch 来表示机械在位切换的状态：

- 1：切换打开
- 0：切换关闭

p#_mp_switch 必须通过外部逻辑得到正确的电平(level)。引脚的状态反映在 P#CMD.MPSS 位，通过 P#IS.DMPS 位改变状态（假设 CAP.SMPS 和 P#CMD.MPSP 都被设置为 1）。注意：平台必须有附加的硬件实现支持机械在位切换。

3.5.2.5 端口扩展器支持端口扩展器示意图：



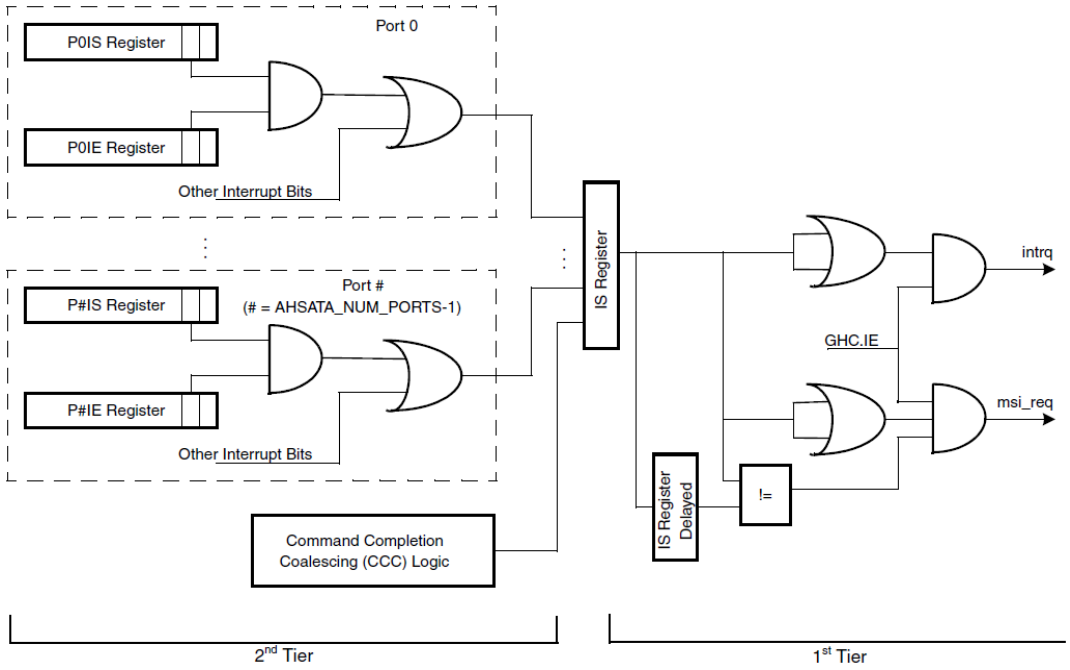
DWC_ahsata 支持基于命令切换的端口扩展器功能。当一个端口连接到端口扩展器上，软件 首先通过发出软件复位到端口复用器的端口 0Fh（控制端口）进行枚举。当对应端口扩展器返 回签名是–端口扩展器‖时，端口扩展器就被连上了对应的端口。如果返回的签名是其他的设备 类型，那么端口扩展器就没有被连接上。DWC_ahsata 支持命令列表越权控制的特征（当 CAP.SCLO=1），

通过 P#CMD.CLO 帮助软件可靠地枚举端口扩展器。

1) 软件确保 P#CMD.ST 位为1；
2) 软件在命令列表中为软件中断产生 2 个寄存器 FIS 请求，PM port 域的值清为 0Fh；
3) 软件设置 P#CMD.CLO 位 1，强制清除 P#TFD 寄存器的 BSY 和 DRQ 位。
4) 软件设置 P#CMD.ST 位为 1，设置适当的 P#CI 位开始执行软件复位。

3.6.2.7 中断

DWC_ahsata 使用了 AHCI 规范中定义的两层组合产生中断信号的方式：



第一层中断（IS 寄存器）

第一层中断通过 IS 和 GHC 寄存器定义。GHC.IE 位使能整个 DWC_ahsata：当这一位清除 时，中断输出无效，不考虑 IS 寄存器中是否有中断位。GHC.IE 位充当了屏蔽位而不影响中断 状态位的设置。32 位的 IS 寄存器报告了端口是否有中断悬挂。这是按位映射的寄存器，对应每 个 DWC_ahsata 实现的端口（实现了几个端口，就有几位可以被使用）。命令完成合并（CCC） 通过设置 IS.IPS[INT]位产生中断（如果软件使能了中断），其中 INT=AHSATA_NUM_PORTS （DWC_ahsata 配置实现的端口数量）。例如，如果 DWC_ahsata 配置成 8端口，那么端口使用 IS[7:0]，CCC 中断使用 IS[8]。

第二层中断（P#IS 寄存器）

第二层中断通过 P#IS (status)和 P#IE (interrupt enable)寄存器定义。P#IS 有不同的中断位， 通过设置 P#IE 中对应的位可以独立地使能或不使能。P#IS 中的中断状态位不需要考虑对应的 P#IE 位是否设置。

3.6.2.8 物理层和链路控制

DWC_ahsata 提供 2 个 32 位的寄存器 GPCR 和 GPSR 用做通用目的的控制和状态，它们都 实现在 GCSR 模块中。GPCR 寄存器映射到 gp_ctrl 输出总线的对应位，GPSR 寄存器映射到 gp_status 输入总线对应的位。GPCR 寄存器和 gp_ctrl 总线需要配置 DWC_ahsata 的参数 GP_CTRL 实现。在这种情况下，GPCR 异步复位的初始值设置为 GP_CTRL_DEF。GPSR 寄 存器和 gp_status 总线通过配置 DWC_ahsata 的参数 GP_STAT 实现。

DWC_ahsata 提供 2 个可配置的寄存器 P#PHYCR 和 P#PHYSR 用做物理层的控制和状态， 它们都实现在端口 PCSR 模块中。P#PHYCR 寄存器映射到 p#_phy_ctrl 输出总线的对应位， P#PHYSR 寄存器映射到 phy_status 输入总线对应的位。宽度的配置参数 PHY_CTRL 和PHY_STATUS 被所有端口共用。端口链路层特征（scrambler, de-scrambler, repeat drop，扰频器， 去扰频器，重复丢弃）通过 BISTCR.LLC 域控制（BISTCR 寄存器在 GCSR 模块中），通过清除对应的位， 在某些正常操作时可以不使能， 例如在测试时。 实现这种操作的端口通过 TESTR.PSEL 来选择。
BISTCR.LLC 位在上电时设置，默认使能扰频器，去扰频器，RPD 功能。 如果不使能这些功能，软件必须执行下面的步骤：

1) 设置 P#SCTL.DET 为 1h
2) 清除需要的 BISTCR.LLC 位
3) 清除 P#SCTL.DET 为 0

3.6.2.9 复位条件 系统复位

系统总线通过驱动 hresetn=0（异步总线复位）复位 DWC_ahsata。这通常发生在上电或系 统总线失败时。所有的 DWC_ahsata 组件都被初始化，包括端口，属性寄存器，接口逻辑等。 **全局复位**

软件可以通过设置 GHC.HR 为 1 进行 DWC_ahsata 全局复位。当软件设置 GHC.HR 位为 1 时，DWC_ahsata 执行内部复位动作，复位结束时清除 GHC.HR 位为 0。软件写 GHC.HR 为 0 没有作用（硬件动作）。

注意：这个复位清除 P#SCTL.SPD 域，所有端口通信以 p#_phy_spdmode 设置的最大许可 速

率重新开始。

为了执行全局复位，软件设置 GHC.HR 为 1，可能查询直到这一位读出为 0，表明复位结 束。DWC_ahsata 的初始化如过程如下：

GHC.AE, GHC.IE, IS 寄存器，所有的端口寄存器域（除了 P#FB/P#FBU/P#CLB/P#CLBU），在寄存器存储器空间中所有没有 HwInit 的位都进行复位。

设置 GHC.HR 为 1 不影响所有其他全局寄存器/位和任何端口寄存器中的 HwInit 位。 设置 GHC.HR 为 1 不影响端口寄存器中的 P#FB, P#FBU, P#CLB, P#CLBU 域。

P#CMD.SUD 位复位为 0；软件对设置 P#CMD.SUD 和 P#SCTL.DET 域做出适当的反应， 如可以在 SATA 链路上建立通信。

端口复位（COMRESET）

软件写 P#SCTL.DET 域为 1 发起端口复位，在接口上发起 COMRESET 并开始重新建立物 理层通信。软件在清除 P#SCTL.DET 位 0 前至少需要等待 1ms。在清除 P#SCTL.DET 为 0 后， 当 P#SCTL.DET[0]设置为 1 表示软件需要等待重新建立通信。软件需要向 P#SERR 寄存器中写 全-1ll 来清除寄存器中可能有的错误位，这也是端口复位的一部分。

注 意 ： DWC_ahsata 当 设 置 P#SCTL.DET=0x1 时 ， 驱 动 p#_phy_reset(_n) ， 当 P#SCTL.DET=0x0 时取消 p#_phy_reset(_n)并发送 COMRESET OOB 序列。

软件复位

软件在命令列表中建立 2 个 H2D 寄存器 FIS。第 1 个寄存器 FIS 的 SRST 位设置为 1，C 位设置为 0，命令表中 CH[R] (reset)和 CH[C] (clear BSY on R_OK)设置为 1。CH[R] (reset)位触 发端口执行 SYNC 脱离，在执行软件复位前必须使设备进入空闲条件。由于设备在软件复位序 列中不能发出对第 1 个 FIS 的响应，需要在第 1 个寄存器 FIS 中设置 CH[C] (clear BSY on R_OK)位以清除 BSY 位，并发出下一个寄存器 FIS。第 2 个寄存器 FIS 的 SRST=0，命令表中 CH[R] (reset)和 CH[C] (clear BSY on R_OK)清除为 0。当发出软件复位序列时，不能有其他命令在命令表中。在发出软件复位前，软件必须清除 P#CMD.ST，等待端口进入空闲(P#CMD.CR='0')，然后重新设置。 P#CMD.ST。发出复位前， P#TFD.STS.BSY 和 P#TFD.STS.DRQ 必须清除。当 P#TFD.STS.BSY 和 P#TFD.STS.DRQ 仍然根据失败命令设置时，需要尝试使用端口复位或使用忽略命令列表(P#CMD.CLO)。 注意：1）当进行错误恢复时应该使用端口复位(COMRESET)而不是软件复位。

2）P#CI 必须在 P#CMD.ST 之前设置，如果当前 DMA 读传输被中断时，否则，PDMA 不 能 SYNC 脱离 DATA FIS。

3.6.2.10 接口速率支持

CAP.ISS=2h 表示 DWC_ahsata 支持 1.5 Gb/s, 3 Gb/s, 6.0 Gb/s 接口速率。软件可以通过设置 P#SCTL.SPD 为 1h 限制端口工作在 1.5GB/s 的速率。

3.6.2.11 交错旋转

当 CAP.SSS=‘1‘时，DWC_ahsata 支持交错旋转(Staggered Spin-up, SSS)操作。这个特征用于 分开旋转连接在一起的设备。这样减少多个设备上电时的电源需求。

注意：如果需要系统支持交错旋转，那么设备和 BIOS/驱动软件也必须支持交错旋转。

如果需要旋转连接设备到 DWC_ahsata 端口，软件需要执行 DesignWare Cores SATA AHCI Databook 第 3.1.1 节 Firmware Specific Initialization 中定义的过程。

3.6.2.12 活动 LED

CAP.SAL=1 表明软件使能了活动 LED。根据端口活动状态 P#_act_led 输出驱动外部 LED。

- _1‘ – LED On (端口活动)
- _0‘ - LED Off (端口不活动)

当下面这些情况时，端口驱动 LED On(p#_act_led=‘1‘)：

- (P#CI != 0h or P#SACT != 0h) and P#CMD.ATAPI = ‘0‘;
- (P#CI != 0h or P#SACT != 0h) and P#CMD.ATAPI = _1‘ and P#CMD.DLAE = _1‘.

当 P#CI 和 P#SACT 都为 0h 时，端口驱动 LED off (p#_act_led=‘0‘)

3.6.2.13 异步通知

当 CAP.SSNT=1 时 DWC_ahsata 支持异步通知。它允许 ATAPI 当传输介质插入或移除时向 Host 发送信号，避免由于更换传输介质而查询设备。发送到 Host 的信号是一个带_I‘ (interrupt) and _N‘ (notification)的 Set Device Bits FIS 。

为了使用异步通知，软件需要设置 P#IS.SDBS 位使能 Set Device Bits FIS 的中断通知。当访 问

ATAPI 设备是空闲状态时，软件需要使设备进入低功耗状态。当设备改变传输介质时，通过 Set Device Bits FIS 通知 DWC_ahsata 端口。

空闲端口发出 P#IS.SDBS 中断后，软件应该轮询设备来确定是谁发出的中断。Host 收到的 任何 FIS 的第一个双字包含了 4 位端口扩展器端口（PM Port）域。端口扩展域表明哪个端口扩 展器上的端口/目标向 DWC_ahsata 端口发出了 FIS。当 DWC_ahsata 端口收到_N‘ (notification) 位设置上了的 Set Device Bits FIS 时，P#SNTF 寄存器中 PM Port 域对应的位也被设上了。当 Set Device Bits FIS 的_I‘ (interrupt)位也设置上时， DWC_ahsata 端口设置 P#IS.SDBS 为 1，当中断 使能时会发出中断。

注意：当端口扩展器不在位时，PM Port 域在 Set Device Bits FIS 中是 0h，会引起 P#SNTF 寄存器第 0 位（端口扩展器端口 0）被设置。

3.6.2.14 BIST 操作

每个 DWC_ahsata 端口可以进入下面描述的 BIST 回环模式。注意：端口链路层的扰频器和去扰频器在所有 BIST 模式下默认是旁路（不使能）的。如果需要使用，可以在进入 BIST 模式前通过软件清除 BISTCR.LLC 中的 SCRAM 和 DESCRAM 位来使能。

- 回环响应（Loopback Responder）
  - Far-end retimed
  - Far-end analog (Port PHY must support this mode)
  - Far-end transmit only
- 回环初始化（Loopback Initiator）
  - Far-end retimed
  - Far-end analog
  - Near-end analog (Port PHY must support this mode)
  - Far-end transmit only

3.6.2.15 命令完成合并

命令完成合并(Command Completion Coalescing, CCC)用于减少中断和命令完成对重负载系 统的动作开销。但下面任何条件成立时，DWC_ahsata 核心产生一个中断允许软件处理完成的命令：

- 完成了软件定义数量的命令
- 到达了软件定义的超时时间

软件通过 CCC_PORTS 寄存器设置需要使用 CCC 特征的端口。基于应用时钟频率，CCC 逻辑使用 DWC_ahsata 定义的寄存器 TIMER1MS 来产生 1ms 的间隔。软件必须在使能 CCC 特征前向寄存器中装入需要的值：Fappclk * 1000，Fappclk 为以 MHz 为单位的 AMBA 总线时钟 (hclk/aclk)频率。CCC 的详细介绍和例子可以参考 Serial ATA AHCI 1.3 Specification，Chapter 11，Command Completion Coalescing。

注意：可以通过配置 DWC_ahsata 参数 CCC_SUPPORT 实现或去除 CCC 逻辑，CCC 相关的寄 存器都成为–reserved‖（读返回 0）

3.7 显示与多媒体子系统

3.7.1 DMAC

3.7.1.1 基本功能

DMAC 用于代理显存与系统主存之间，以及显存内部的 DMA 传输，包括：

- DMA 读主存写显存（DMAR）：主存 > 显存
- DMA 读显存写主存（DMAW）：显存 > 主存
- DMA 读显存写显存（DMAV）：显存 > 显存

DMAC 的基本特性如下：

- 支持源、目的数据的聚合/分散传输；
- 支持不同中断源的独立使能；
- 支持 AXI 总线读请求数量（0~16 个）动态可配置；
- 支持各通道 DMA 事务传输性能实时统计；
- 要求描述符地址 64B 对界。

3.7.1.2 基本操作流程

- **基本传输方式**

软件启动 DMAC 的基本流程如下：

(1). 根据全局状态寄存器选择 DMA 事务号；

(2). 初始化 DMA 描述符列表（每个描述符须明确指出传输方向）；

(3). 将链表起始地址配置到 IDAR（初始描述符地址寄存器）启动 DMA 引擎，开始传输；

(4). 传输完毕置中断信号。

基本传输方式的 DMAC 启动配置

配置顺序	寄存器名称	配置信息
1	IDAR	初始描述符地址（64B 对界）

描述符在主存中的数据格式要求如下表，包括源地址、目的地址、传输数据长度、下一个 描述符地址以及控制寄存器。当初始化描述符列表时，软件根据需求将 CTLR[9:8]配置为不同 的 DMA 传输类型，该配置信息在后续描述符列表项的 CTLR 中保持不变。若描述符为链表的 最后一个，则将 CTLR[0]配置为 1，否则为 0。

描述符数据格式

地址偏移	寄存器名称	寄存器说明
0x00	SAR	源地址寄存器
0x04	SNR	源数据块数量寄存器
0x08	SDLR	源数据块长度寄存器
0x0C	SINCR	源数据间距寄存器
0x10	DAR	目的地址寄存器
0x14	DNR	目的数据块数量寄存器
0x18	DDLR	目的数据块长度寄存器
0x1C	DINCR	目的数据间距寄存器
0x20	NDAR	下一个描述符地址寄存器
0x24	CTLR	DMA 事务控制寄存器，控制传输类型和描述符属性

0x28~0x3F	保留	-

➢ **单块传输方式**

软件可以通过配置 SAR、SNR、SDLR、SINCR、DAR、DNR、DDLR、DINCR 和 CTLR 寄存器启动单块方式的 DMA 传输，用于性能评估和故障处理，基本流程如下：

(1). 根据全局状态寄存器选择 DMA 事务号；

(2). 依次配置 DMA 事务的 SAR、DAR、DLR 寄存器；(3). 配置 DMA 事务的 CTLR 启动 DMA 引擎，开始传输； (4). 传输完毕置中断信号。

单块传输方式的 DMAC 启动配置

配置顺序	寄存器名称	配置信息
1	SAR	源地址寄存器
2	SNR	源数据块数量寄存器
3	SDLR	源数据块长度寄存器
4	SINCR	源数据间距寄存器
5	DAR	目的地址寄存器
6	DNR	目的数据块数量寄存器
7	DDLR	目的数据块长度寄存器
8	DINCR	目的数据间距寄存器
9	CTLR	CTLR[0]=1‘b1

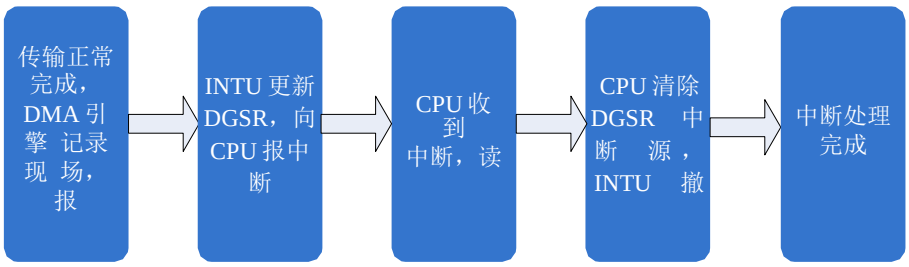
3.7.1.3 中断流程

DMAC 通过中断方式与 CPU 交互，下表对不同中断进行了比较。需要注意：对于正常完 成中断，CPU 对 DMAC 进行一次 IO 读，对于其他类型中断，则进行两次 IO 读。

DMAC 中断类型

中断类型	中断描述	中断产生部件	处理方式
正常完成中断	DMA 传输完成，且传输过程中无错误发生	DMA 引擎（DMARU/DMAWU/DMAVU）	清除中断源
软件异常中断	IO 地址越界	AHB slave 接口/DMA 通道/GDU/AXI Master 接口	
	只读寄存器错误		
	描述符地址非 64B 对界	DMA 通道	
异常完成中断	源端传输错误	DMA 引擎（DMARU/DMAWU/DMAVU）	
	目的端传输错误		
	数据缓冲奇偶校验错误		
硬件故障中断	取描述符传输错误	DMA 通道	
	状态机非法状态错误	AHB slave 接口/DMA 通道/DMA 引擎/GDU/AXI Master 接口	软件复位

➢ **正常完成中断处理流程**



正常完成中断处理流程 基本流程如上图，各步骤说明如下：

(1). 当 DMA 传输正常完成时，DMA 引擎记录中断现场，并向 DMAC 中断部件（INTU） 发

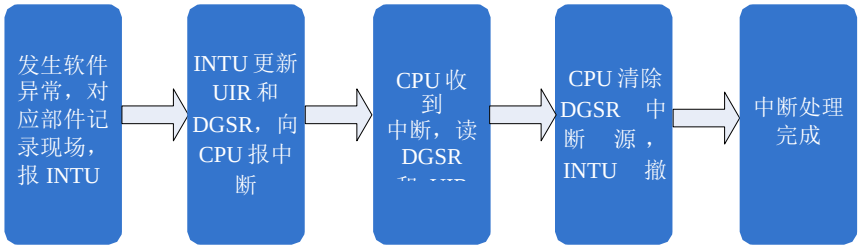
出传输完成中断信号；

(2). INTU 更新对应 DMA 事务的 DGSR，通过中断线向外发出中断。

(3). CPU 收到中断后，读 DGSR 获得产生中断的 DMA 事务号。

(4). CPU 清除 DGSR 的对应中断源，INTU 撤销中断并清除 DMA 引擎中断寄存器（DRIR/DWIR/DVIR）。

(5). 中断处理完成。



软件异常中断处理流程

➢ 软件异常中断处理流程

基本流程如上图，各步骤说明如下：

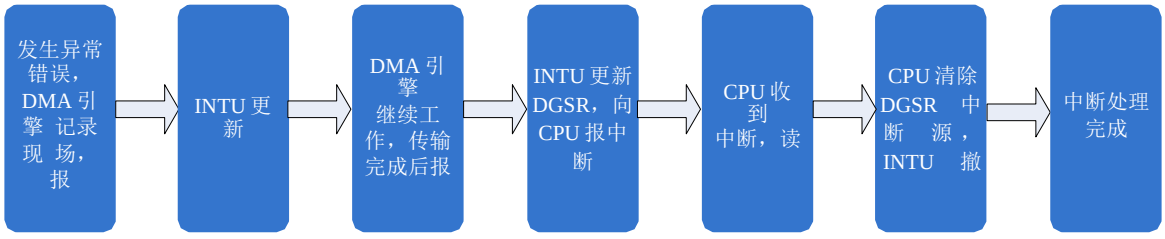
(1). 当发生软件异常（IO 地址越界、只读寄存器错误、以及描述符地址非 32B 对界）时，对应部件（AHB slave 接口或通道）记录中断现场，并向 INTU 发出相应的软件异常信号；

(2). INTU 更新对应 DMA 事务的 UIR 和 DGSR 存器，通过中断线向外发出中断；

(3). CPU 收到中断后，若是描述符地址非 32B 对界错误，直接读 DGSR 获得产生中断的 DMA 事务信息。若是 IO 地址错误或只读寄存器错误，则读 DGSR 和 UIR 获得具体的中断部件信息；

(4). CPU 清除 DGSR 的对应中断源，INTU 撤销中断并清除相应中断寄存器；

(5). 中断处理完成。



异常完成中断处理流程

➢ **异常完成中断处理流程**

基本流程如上图，各步骤说明如下：

(1). 当发生异常错误（源端传输错误、目的端传输错误、及数据缓冲奇偶校验错误）时，DMA 引擎记录中断现场，并向 INTU 发出相应的异常错误信号；

(2). INTU 更新对应 DGSR 存器；

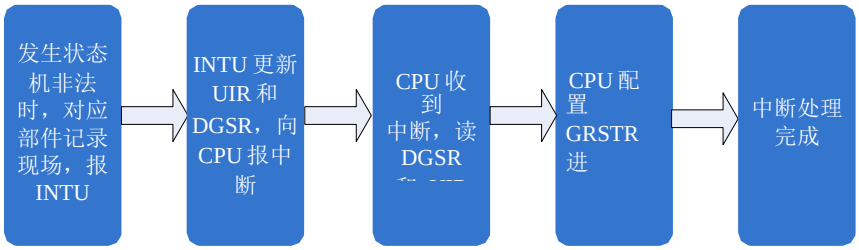
(3). DMA 引擎继续后续数据传输，传输完成后向 INTU 发出传输完成中断信号；

(4). INTU 更新对应 DGSR 存器，通过中断线向外发出中断；

(5). CPU 收到中断后，读 DGSR 获得产生中断的 DMA 事务号和中断类型；

(6). CPU 清除 DGSR 的对应中断源，INTU 撤销中断并清除 DMA 引擎中断寄存器。

(7). 中断处理完成。



状态机非法错误处理流程

➢ **硬件故障中断处理流程**

软件处理不同硬件故障类型的处理方式不同：当发生取描述符传输错误时，错误现场保存 在 DMA 通道中，中断流程同软件异常中断处理方式；当发生状态机非法状态错误时，基本流 程如上图，各步骤说明如下：

(1). 当发生状态机非法状态错误时，对应部件（AHB slave 接口、DMA 通道、DMA 引擎、GDU、以及 AXI Master 接口）记录中断现场，并向 INTU 发出错误中断信号； (2). INTU 更新对应 DMA 事务的 UIR 和 DGSR 存器，通过中断线向外发出中断。(3). CPU 收到中断后，读 DGSR 和 UIR 获得产生中断的具体部件。

(4). CPU 配置 GRSTR 进行软件复位。

(5). 中断处理完成。

3.7.1.4 IO 寄存器

寄存器详细说明请参考寄存器手册。

3.7.2 DC

3.7.2.1 基本功能

显示控制器 DC 主要功能是从显存中获取帧缓冲数据，将其输出至片外显示器。ICH2 中 DC 支持双 DVO 输出接口。DC 支持–交换（Switch）Ⅱ特性，即每个 DVO 接口可选择本显示通道内容输出或借用对方显示通道内容输出。

ICH2 中 DC 两显示通道均可最高支持 HD1080P（即 1920x1080）规格的分辨率。

3.7.2.2 基本操作流程

DC 按照用户设定的显示分辨率输出显示帧缓冲数据，其基本操作流程如下：

1) 用户确定待显示分辨率，准备待输出帧缓冲数据显示分

辨率规格参考显示和视频标准 VESA 规范。

2) 软复位显示通道流水线；

对 IO 寄存器 Frame Buffer Configuration 进行操作，将其[20]置为 0 即可实现对显示通道流 水线的软复位。例如，若待设置的分辨率像素格式为 RGB888，则可将其设置为 32‘h00000104。

软复位后，该 DVO 接口显示输出无效，对应的显示器显示会进入–黑屏‖状态。

3) 设定显示分辨率对应像素时钟；

根据显示 VESA 标准设定分辨率对应的像素时钟频率。目前 ICH2 像素时钟主要由片外时 钟发生器 CDCE925 芯片提供。用户可通过 I2C 接口配置该器件内 IO 寄存器，设定对应显示通 道的像素时钟。具体参考软件接口手册《I2C 接口控制器》章节说明。

4) 设置显示分辨率对应 IO 寄存器；

根据显示 VEIS 标准设定分辨率对应相关 IO 寄存器。主要涉及 5 个 IO 寄存器：Frame Buffer Stride、HDisplay、HSync、VDisplay 和 VSync。其中 Frame Buffer Stride 含义为显示屏一行的字 节数，其设置值要求为 128 倍数，不足时向上靠。

5) 设置帧缓冲基地址；

设置待显示帧缓冲数据的基地址至 IO 寄存器 Frame Buffer address，要求 128B 对界。

6) 撤销显示通道流水线软复位；

再次配置 Frame Buffer Configuration 中[20]位，将其设置为 1 即可撤销对显示通道流水线的 软复位。例如，将其设置为 32‘h00100104。

当软复位被撤销后，显示通道流水线即开始正常工作，对外输出显示像素数据。DC 部件完 成一帧图像显示后，会通过中断通知外部其已完成一帧的显示。并按照分辨率规格等待若干固 定的间隔后开始下一帧的显示，无论用户响应中断与否。

当用户收到该显示通道对应的中断请求后，即可修改寄存器 Frame Buffer address 值，将其 配置为下一帧待显示帧缓冲数据基地址。从而完成显示数据刷新，后续可依次连续显示。

用户可通过配置寄存器 Frame Buffer Configuration 中 Switch Panel 位（即第[9]位）实现显示接口借用对方显示通道内容输出。其流程如下：

1) 设置被借用显示通道； 具体参考前述设置
分辨率基本流程。

2) 设置本显示通道借用位并软复位；

设置 IO 寄存器 Frame Buffer Configuration 的 Reset 位（即第[20]位）为 0，Switch Panel 位（即第[9]位）为 1。例如，将寄存器 Frame Buffer Configuration 值设置为 32‘h00000200。

3) 撤销本显示通道软复位。

设置 IO 寄存器 Frame Buffer Configuration 的 Reset 位（即第[20]位）为 1，Switch Panel 位（即第[9]位）为 1。例如，将寄存器 Frame Buffer Configuration 值设置为 32‘h00100200。

3.7.2.3 参考文档

DC 其它功能描述请参考文档《DispalyController_IntegrationManual.pdf》。

3.7.3 GPU

3.7.3.1 基本功能

ICH2 中 GPU 和 GAR 部件主要实现对图形处理的 2D 渲染和 3D 渲染。其中 2D 渲染还包 括 YUV 转 RGB 视频后处理功能。

2D 渲染包括矩形填充与清除、拉伸和缩减，Alpha 混淆，直线描绘、光栅操作、90/180/270 度旋转等功能；3D 渲染兼容 OpenGL ES2.0 及其扩展规范，包括顶点处理、像素处理、纹理处 理等。

3.7.3.2 基本操作流程

ICH2 中 GPU 主要以加速模型方式执行，无论是 2D 渲染还是 3D 渲染，其基本执行流程均 如下：

1) 设置访主存地址偏斜值；

GAR 部件根据 GPU 访问地址确定其路由方向，低 2GB 地址空间位于显存中；高 2GB 地址 空间位于主存中。为了统一地址视角，GAR 支持对路由后的访主存地址进行代换，其代换方法 为原地址减去用户设定的偏移值。

访主存地址偏斜寄存器 GPUR_MMBIAS 物理上位于显示存储控制器 MC 部件中（部件内 地址偏移值为 0x0848），其复位后默认值为 32‘h4，对应 2GB 的地址偏移（即以 512MB 为偏移 单位，512MB*4=2GB）。例如以默认偏移计算：GPU 发出 2GB 访存地址对应主存中 0 地址单

元。

2) 构造虚实代换表；

GPU 访存支持虚实代换，其原始地址位宽为 32 位（即[31:0]）。其中[31]位为虚拟地址标记 位，标识该地址为物理地址访存还是虚拟地址访存。因此，其物理地址访存地址范围为 2GB 大 小（从 0 地址开始）。由于 Command Buffer 数据主要由 CPU 构造而 GPU 消费，其需要置于主 存中。而根据 GAR 路由特点，访主存地址范围为 2GB~4GB。因此对其访问采用虚拟地址进行。

GPU 部件内部访存请求来源包括多个，其中支持虚拟地址访存包括：FE、TX、PE、PEZ 和 RA。每个对应一个页表基址寄存器。访问 Command Buffer 对应 FE 部件页表基址寄存器。

GPU 访存中，每个页大小对应 4KB 空间。其虚实代换过程如下：

1) 对虚拟地址进行拆分

VirtualAddr[31:0] = {1‘b1, VPA[30:12], offset[11:0]}

其中 VPA[30:12]代表页表号。

2) 根据页表基址和页表号计算页表项地址

PageTableAddr[31:0] = {11‘b0, VPA[30:12], 2‘b0} + PAGE TABLE BASE

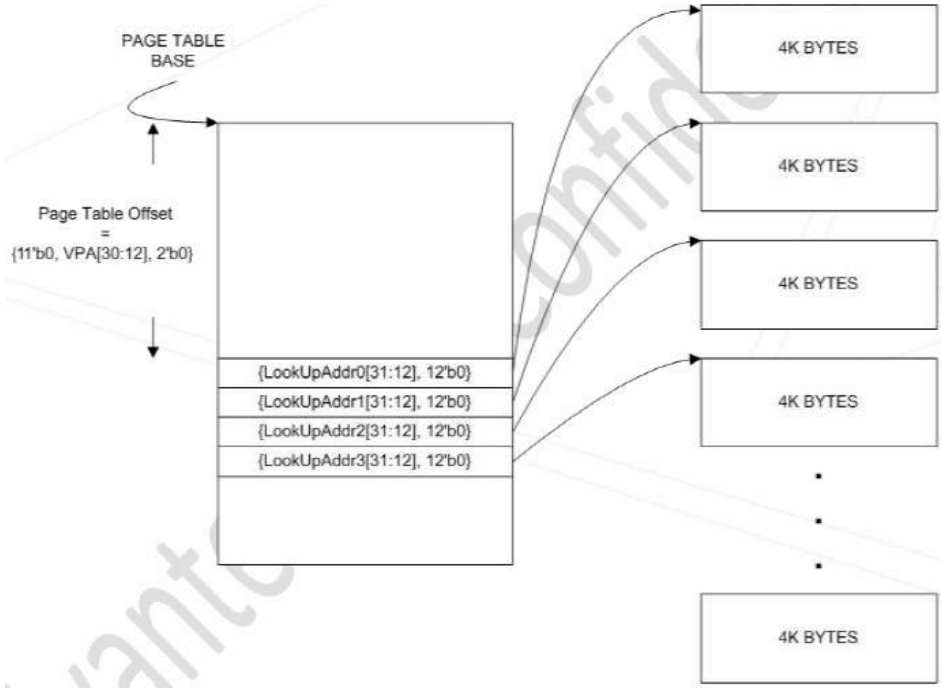
每个页表项占用 4B 存储空间。

3) 根据页表项地址获得物理页表基地址，计算物理地址

PhysicalAddr[31:0] = {LookUpAddr[31:12],offset[11:0]} 其中

LookUpAddr[31:12]为页表项 4B 数据的高 20bit 位。

对应的示意图如下所示：



4) 加载 Command Buffer 数据至主存中；

5) 加载其它 Bufffer类型数据至显存中；

根据课题类型，例如 2D 渲染和 3D 渲染等，加载不同类型的 Buffer 数据至指定位置（具体 地址由 Command Buffer 内容决定）。

6) 配置课题相关的 IO 寄存器；

例如配置中断使能寄存器 AQIntrEnbl，时钟控制寄存器 AQHiClockControl 等。

7) 配置前端启动 PC 值；

配置寄存器 AQCmdBufferAddr 指示 Command Buffer 基地址。由于 Command Buffer 位于主 存中，采用虚地址才能访问到。因此，该寄存器第[31]位必须设置为 1。

8) 配置寄存器启动 GPU 渲染；

配置寄存器 AQCmdBufferCtrl 指示 Command Buffer 程序段长度（[15:0]位标示，以 8B 为 单位）。同时置第[16]位为 1 启动 GPU 渲染。

9) 等待 GPU 渲染完成。

通过等待 GPU 完成中断（读访问寄存器 AQIntrAcknowledge）或查询空闲状态（读访问寄 存器 AQHiIdle）确定 GPU 渲染完成情况。其结果保存至 Command Buffer 程序中所指定的 FrameBuffer 位置。

3.7.3.3 参考文档

驱动开发与移植参考文档《Driver_Development.GCCORE.pdf》；2D 驱动 API 参考文档《Driver_2D_API.pdf》；SDK 用户手册参考文档《Vivante.SDK.User_Guide.pdf》；IO 寄存器列表 和虚实地址代换详细过程参考文档《Hardware_Integration.GC860T.pdf》。

3.7.4 VPU

3.7.4.1 基本功能

ICH2 中 VPU 部件主要实现对视频编解码的加速。其中支持视频解码格式包括：H.264、 H.263、

MPEG-4、MPEG-2、AVS、RVX、DIVX、VC1、JPEG 和 MPEG 等；支持视频编码格 式包括：H.264、H.263、MPEG-4、JPEG 和 MJPEG 等。

VPU 仅支持对显存的访问，最高可支持对 1080P 规格视频进行编解码。需要说明的是：对 于视频解码功能，VPU 不支持解码结果 YUV 转 RGB 视频后处理过程，此功能在 ICH2 中需通 过 GPU 完成。

3.7.4.2 基本操作流程

VPU 部件内包含 16bit 位嵌入式处理器核心 BPU。其基本操作流程包括：VPU 初始化、视 频解码和视频编码等。

（一）VPU 初始化流程

VPU 初始化（亦即 BPU 初始化）流程如下：

1) 通过设置寄存器 CodeRun 停止 BPU 运行；
2) 加载完整 BPU 固件至显存中；
3) 直接加载固件核心至 BPU PRAM 中；

固件核心（固件前 1KB 数据）为视频编解码公共部分，与具体视频格式无关，主要用于 BPU 的自举。用户可通过 IO 操作（写 IO 寄存器 BIT_CODE_DOWN）将其加载至 BPU 片上 PRAM 存储器中。

4) 设置 BPU 运行相关指针寄存器；

包括 Working Buffer、Parameter Buffer 和 Code Buffer（固件）基址寄存器。 5)

设置流缓冲和帧缓冲控制选项，包括大小端设置等；

6) 设置中断使能和复位寄存器；
7) 通过设置寄存器 CodeRun 使能 BPU 运行；
8) 等待直至 BPU 空闲。

查询 BPU 忙寄存器 BIT_CODEC_BUSY。若其返回结果为 0，则表明 BPU 此时已空闲。

(二) 视频解码流程

视频解码流程包括：视频头解析和视频解码。视频头解析流程如下：

1) 加载视频流至流缓冲中；
2) 解码序列初始化 SeqInit；

用户通过发送 DEC_SEQ_INIT 命令至 VPU 可以获取视频流解码配置信息。通过对视频头 解析，用户可以获取视频图形规格、帧率、所需最小帧缓冲数等信息。

3) 登记帧缓冲信息；

根据 SeqInit 结果显示所需最小帧缓冲数分配帧缓冲地址，并在 VPU 中登记。

4) 配置 Second AXI 使用情形；

在 ICH2 中 VPU 不支持 Second AXI 接口。 因此，用户需通过配置寄存器关闭掉 Second AXI 接口使用。

在完成对视频头解析并获取解码相关配置信息后，可以进行具体视频解码。其流程如下：

1) 初始化视频解码；

用户根据需要配置相关解码选项，包括：提前扫描使能及其模式（Pre-scan Enable、Pre-scan Mode）、I 帧搜寻使能（I-Frame Search Enable）、跳帧模式（Frame Skip Mode）等。

2) 开启视频解析

用户通过发送 DEC_PIC_RUN 命令至 VPU，开启视频解析。 3)

解码器视频处理

在开始 Pre-scan 模式时，前面所填充的视频流信息可能不够。因此要求用户根据 VPU 解码 状态提前补充视频流数据。

4) 等待视频解码完成

用户可通过查询相关状态寄存器或等待中断确认 VPU 解码完成。视频解码结果保存至帧缓 冲中。

当用户从帧缓冲中取出解码结果后，即可重新启动新视频解码，通过重复上述视频解码流 程即可实现对完整视频文件的解码。

(三) 视频编码流程视频编码流程包括：视频头构造和视频编码。视频头构造流程如下：

1) 确定编码格式及相关信息；包括视频流地址及其规模，编码格式、视频图像规格、目标帧率等信息。

2) 编码序列初始化 SeqInit；

根据目标视频相关信息设置 VPU 编码格式相关的寄存器，并发送 ENC_SEQ_INIT 命令至 VPU。

3) 登记帧缓冲；

4) 构造高层视频头信息。

用户可采取两种路径构造视频头信息： PARA_BUF 或视频流缓冲。推荐用户采用 ENC_PUT_AVC/MP4 _HEADER 等命令通过视频流缓冲构造，视频头信息根据大小端设置直接 保存至视频流缓冲中。

视频编码流程如下：

1) 加载 YUV 图像数据；

加载待编码 YUV 图像数据至显存。

2) 初始化视频编码；

用户配置源缓冲地址及其格式、量化步骤、强制跳帧和 I 帧选项等至 VPU。 3)

开启视频编码；

用户发送 ENC_PIC_RUN 命令至 VPU 开启视频编码

4) 等待视频编码完成。

用户可通过查询相关状态寄存器或等待中断确认 VPU 编码完成。视频编码结果保存至视频 流缓冲中。

当用户从视频流缓冲中取出编码结果后，即可重新启动新视频编码，通过重复上述视编解 码流程即可实现对完整视频文件的编码。

3.7.4.3 参考文档

VPU 用户编程手册参考《cnm-coda851-programmers_guide.pdf》；VPU 工作详细流程参考《cnm-coda851-datasheet.pdf》。

3.7.5 MC

3.7.5.1 基本功能

显存控制器负责存取显示相关的数据，供给 GPU、VPU 以及 DC 使用，并可以通过 DMA 与主存交换数据。

显存有 4 个 AXI 接口，分别用于 GPU、VPU、DMA、DC；一个 AHB 接口，用于 WBSYS 或者 CPU 读取显存数据；一个 APB 接口，连接到套片的 AHB 总线上，用于 WBSYS 或者 CPU 读取显存的 IO。

3.7.5.2 基本操作流程

在访存前，需要对存控进行 PHY 初始化，SDRAM 初始化以及数据训练。

（一）PHY 初始化

在 APB 复位后，需要对 PHY 进行初始化。在 PLL 初始化时，如果按照时序要求撤销 PLL 复位以及 power down，经常会导致 PLL 初始化不成功，所以 PLL 初始化需要做两遍，第一遍 直接写 PRCR[9]为 1，完毕后，按照写 PRCR 为 d6->d0->2d0->d1 的顺序，重新进行一遍 PLL 初始化。在上述步骤中，每次写 IO，均需要等待 5us，再写下一个 IO。

完成上述步骤后，可以读 ACGSR 寄存器，地址 0x1018，若为 9，则表示 PHY 初始化成功， 否则为失败。

（二） 配置参数

完成 PHY 初始化后，需要配置存控各个 IO 寄存器，具体参数需要根据所使用的 memory， 工作频率等进行调整，详细配置方法可参考显存存控设计方案的 IO 寄存器部分。

（三）SDRAM 初始化

首先对 SDRAM 进行上电操作，方法是，写 POWCTL（地址 0x44）寄存器为 0x1，并等待 POWSTAT（地址 0x48）寄存器为 0x1。

然后，根据规范的要求，发送 MRS 命令，配置 SDRAM 颗粒，发送的 MRS 命令根据所使 用的颗粒而不同，具体参考 DDR3 规范，以及颗粒手册，也可以根据 SPD 寄存器内容进行配置。

(四) 数据训练数据训练有以下几个步骤：

1) Loopback 测试：简单的回路测试，测试 PHY 的内部读写数据通路，该步骤为可选；
2) 写平衡调整：将 DQS 和 CK 对齐，该功能集成在 PHY 中，通过 IO 写置开始标志，然后通 过 IO 读判断结束标志；
3) 读平衡训练：通过 MPR 寄存器读来使读 DQS 对齐到读数据眼图的中间；
4) 差拍写平衡调整：调整各 DATX8 之间的差拍；
5) 写数据以及掩码的位偏斜调整：调整一个字节通道内 8 位读数据之间的偏斜，并将写 DQS 对齐到写数据眼图的中间；
6) 读数据位偏斜调整：调整一个字节通道内 8 位读数据之间的偏斜，并将读 DQS 对齐到读数 据眼图的中间；
7) DQS gating 训练：调整 DQS 门控的延迟以及宽度。

3.8 USB3.0 控制器

主要特征如下：

- 支持 USB3.0 的所有特征；

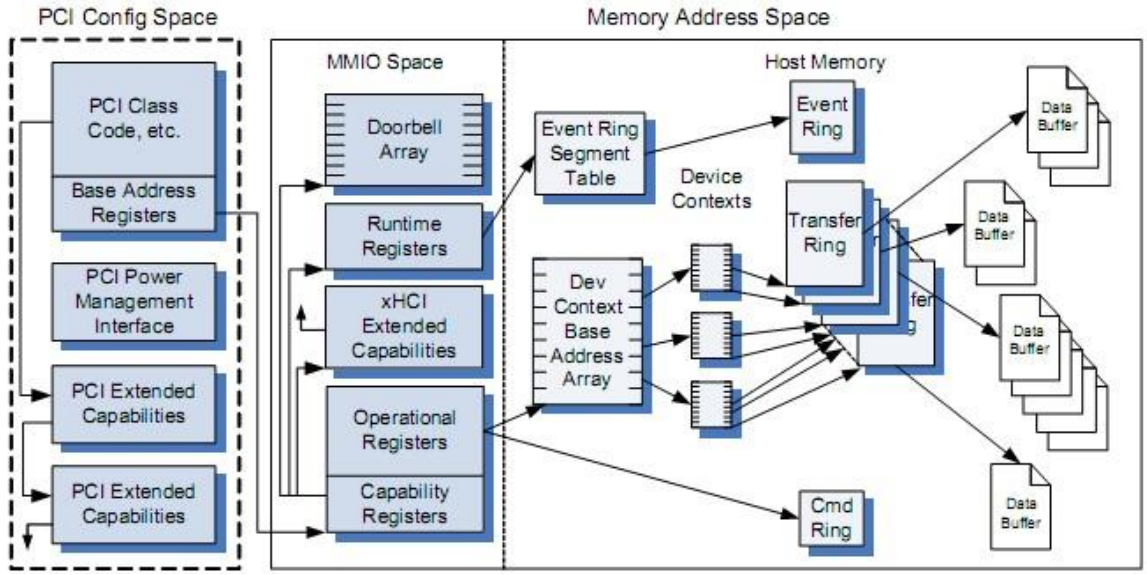
- 支持所有的 USB 设备速率，包括 2.0 的 LS/FS/HS，1.1 的 LS/FS
- 128bit 数据宽度、32bit 地址宽度的 AXI 接口作为 Master 接口，用于 DMA 访问；
- 32bit 数据宽度、32bit 地址宽度的 AHB 接口作为 Slave 接口，用于 CSR 访问和 RAM Debug；
- RAM 接口，采用 3 个两端口 RAM，实现描述符 cache、TxFIFOs、RxFIFOs；
- 实现 6 个 3.0 端口；

3.8.1 可编程寄存器

寄存器详细说明请参考寄存器手册。

3.8.2 接口空间

Figure 3: General Architecture of the eXtensible Host Controller Interface



XHCI 有三类接口空间：

PCI 配置空间：包括基址、功耗管理、能力、扩展能力寄存器等；

MMIO 空间：包括操作寄存器，能力、扩展能力寄存器，runtime 寄存器、门铃阵列等；

Host Memory 空间。

能力寄存器：

只读，控制器的各种限制、约束、能力等，作为给控制器驱动使用的参数；

Runtime、操作寄存器

控制器的配置，以及在运行中可更改的状态等，给软件用来控制和监测控制器的操作 状态；这些寄存器按照，在运行过程中会频繁访问，还是只在初始化阶段访问或在运 行中很少访问，来分开成了两块，便于实现控制器的虚拟化；

扩展能力寄存器：

实现可选的功能，以及为以后新增的能力留出接口；

门铃阵列：

最多 256 个门铃寄存器，即最多支持 255 个设备或 hub（0 号门铃寄存器分配给控制器 用于 command ring 的管理）。门铃寄存器用于软件向控制器告知，对应的 device slot 或 Endpoint 是否有事情要做。DB Target 域用来指明 ringing 的原因。 与一个设备相关的一系列数据结构，统称为-Device Slot‖，-Slot ID‖用来索引具体的某 个 Device Slot。

Device Context 基址阵列：

最多支持 255 个设备或 hub，每个条目指向一个 Device Context 数据结构；

Command Ring：

用于软件向控制器传送命令，对于控制器而言，应该是只读的；

Event Ring：

用于控制器向软件传送命令完成情况和异步事件，对于软件应该是只读的；

Transfer Ring：

用于软件调度针对某一个 Endpoint 的 work items，对于控制器而言，应该是只读的。 是一个 TD（Transfer Descriptor）的循环队列，每个 TD 定义了一个或多个 Data Buffer，Data Buffer 里是要传送的数据。

3.8.3 数据结构

Device Context Base Address Array

一个用 slot ID 来索引的查询表，找到每个 slot 的 Device Context 的基址。当新连 上一个设备时，软件初始化一个 Device Context，向控制器申请一个 slot ID，然后向 DCBAA 中加入一项

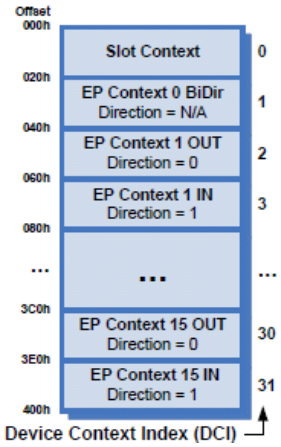
Device Context

由 xHC 管理，向软件报告设备的配置、状态等。一个 Device Context 又包含 32

个数据结构，其中 0 号是 Slot Context，其余的是 Endpoint Context。

软件在 host memory 中为设备新开一个 Device Context，初始化为全 0，这个属于 设备枚举过程的一部分。然后通过一个 Address Device 命令，将管理权让渡给控制器，直到 Disable Slot 将该 slot 禁止掉，在控制器管理期间，Device Context 对软件而言是 只读的。

Figure 71: Device Context Data Structure

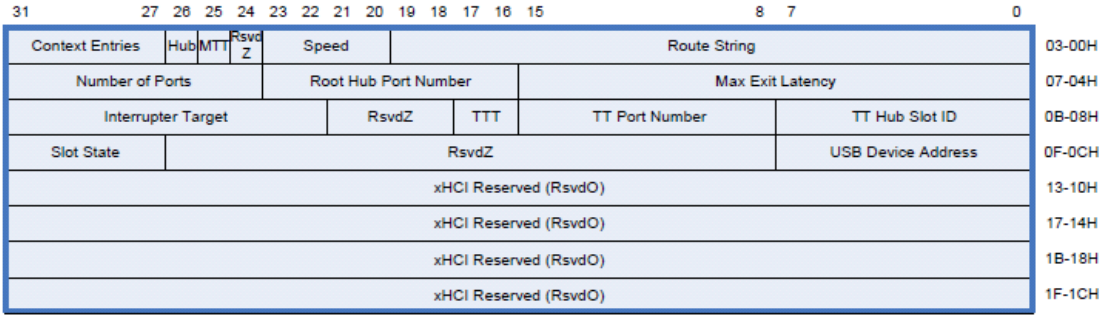


Slot Context

存放和设备有关的、影响到设备全部 Endpoint 的信息。包括控制、状态、寻址、 功耗管理。

它是 Device Context 的成员（向软件传设备参数等），也是 Input Context 的成员（向 控制器传命令等）。

Figure 72: Slot Context Data Structure



Endpoint Context

针对具体的 Endpoint 的配置、状态等信息，包括类型、控制、状态、带宽信息等， 类似对应的端点描述符中的信息。

Endpoint Context 中定义了一个 TR Dequeue Pointer 域，一般是指向和 pipe 关联的 Transfer Ring。USB3 Bulk 端点的 Streams 有一种特殊情况，一个 Endpoint 的 data stream 可以在不同的 Transfer Ring 中复用，所以引入了间接访问 Transfer Ring，即，Endpoint Context 中的 TR Dequeue Pointer 域含一个指针，指向 Streams Context Array，而 Streams

Context Array 中的 Streams Context 可能有一个空指针、或指向 Transfer Ring 的指针、 或指向另一个相关的 Streams Context 的指针。

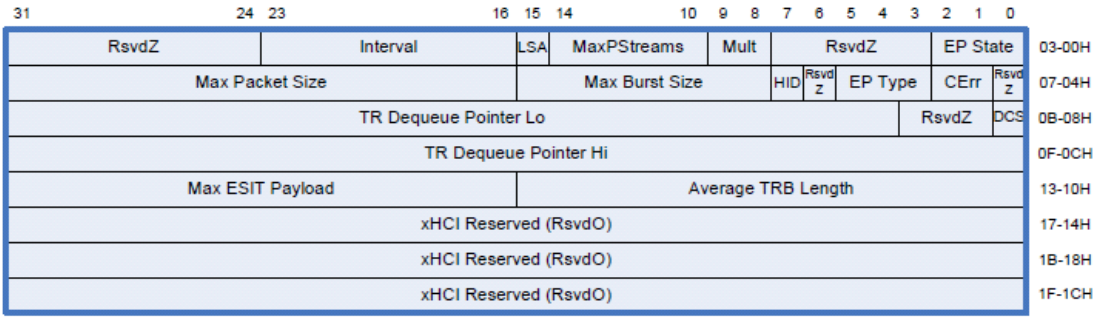
要注意，Device Context 和 Input Context 都为一个设备所有可能的 31 个 Endpoint 提供了对应的 Endpoint Context，但是大多数设备都只会声明少数几个 Endpoint，所以有大量的 Endpoint Context 是 Unused。

Endpoint Context 还有一些用于 debug 的域，比如一个 CErr（Error Count）可以用 于 USB 传输强制重试。

作为 Device Context 的成员，Endpoint Context 用于控制器向软件报告端点相关的 命令参数，也被成为–Output Endpoint Context‖；

作为 Input Context 的成员，Endpoint Context 用于软件向控制器传递相关的命令参 数，也被成为–Input Endpoint Context‖；

Figure 73: Endpoint Context Data Structure



Stream Context Array

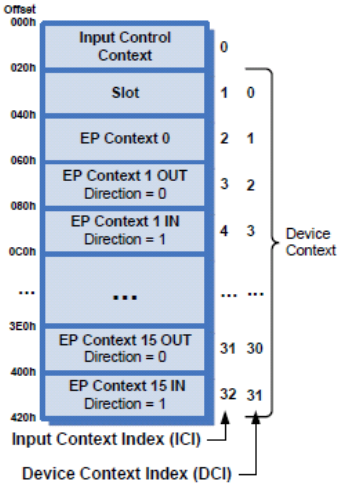
对与支持 Streams 的 USB3 Bulk 端点，引入了这个数据结构，其中包含一些 Stream Context。Stream Context 包含指针指向该 Stream 对应的 Transfer Ring。

Input Context

该数据结构用于软件（通过 Address Device、Configure Endpoint、Evaluate Context 命令）向控制器传递设备的配置、状态信息。

它包含一个 Slot Context、一个 Input Control Context、1-31 个 Endpoint Context。 其中 Input Control Context 来决定其余的哪个 Context 受命令的影响，命令完成后，软 件可以重用或释放 Input Control Context。

Figure 75: Input Context



每个条目 32B。

Input Control Context

含两组按 bit 位组织的 flag：Drop 和 Add。主要用于指明那个 Endpoint 要对软件 命令做出反应，以及如何反应。

Rings

Ring 是一个循环队列，其上的每一个元素都是一个数据结构。xHC 中有三种 Ring：

Command Ring：每个控制器一个；

Event Ring：每个中断源一个；

Transfer Ring：每个 Endpoint 或 Stream 一个；

Transfer Request Block

软件通过 TRB，在 host memory 和控制器之间，搬运物理上连续的一个数据块。TRB 中有一个 Data Buffer 指针、buffer的大小、其它控制信息。

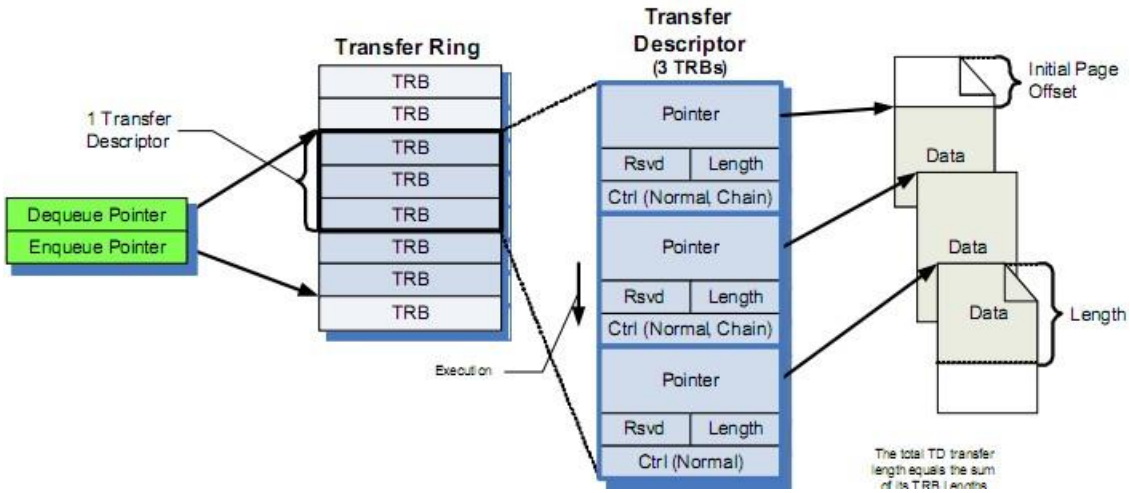
对于小的、单 buffer 操作的 TD，只有一个 TRB；对于多 buffer 的操作（例如 Scatter/Gather），可以有多个 TRB 链接起来构成一个复杂的 TD。Data

Buffer 指针域：提供字节粒度的数据寻址；

Length 域：寄居在 Status 双字中，指明了 Data Buffer 指针所指的 Data Buffer 的大 小，最大为 64K。当 Length 这么多数据传完之后，控制器自动开始访问链上的下一个 TRB。

Scatter/Gather Transfers

Figure 6: Scatter/Gather Transfer Example

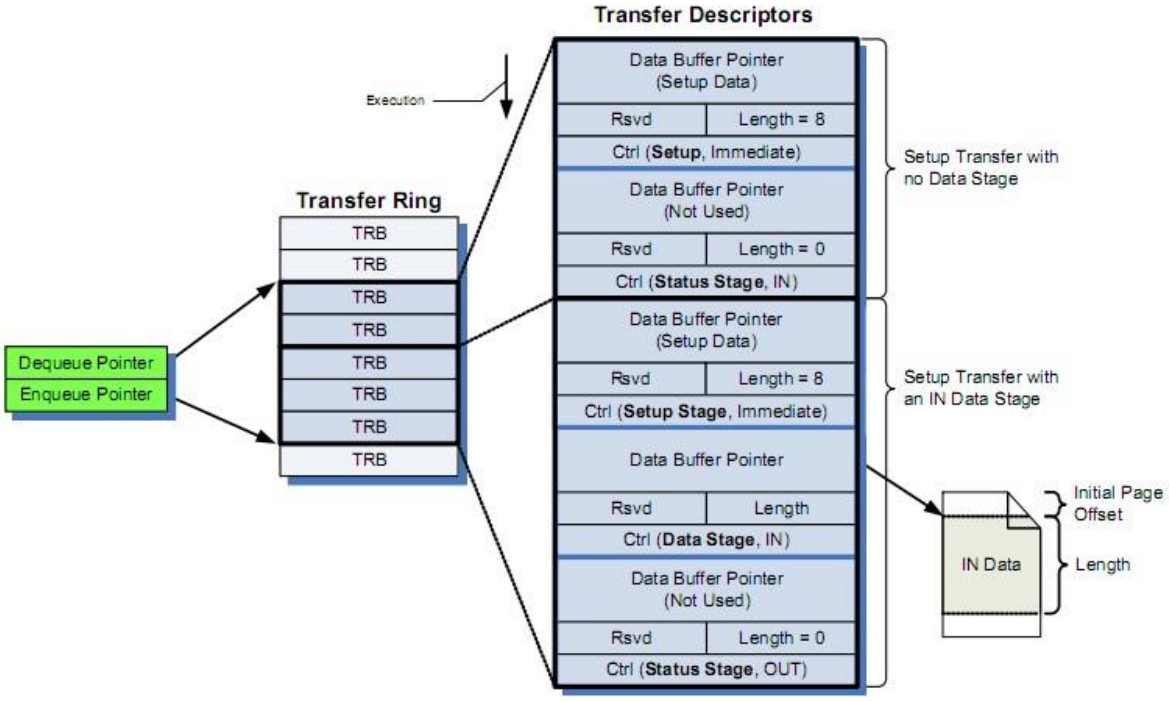


- **Control Transfers**

控制 Endpoint 定义了一个 Message Pipe，而其它的 Endpoint 定义的是 Stream Pipe。一个控制传输可能有两步：Setup 和 Status；也可能是三步：Setup、Data、Status； 针对每一步，有对应的 TD，软件将两个或三个 TD 放到 Transfer Ring 上，然后按门铃。在 Setup 这一步，总是一个 Setup 的 TRB（带 8B 的 Setup 数据），在 Data 这一步，一个 Data 的 TRB 后面可以再接 n 个普通的 TRB，如果是物理上不连续的数据块，就会是多个 TRB 链接在 Data TRB 上。Data 这一步 TD 所含的所有的 TRB 都是往一个方向传数据的。Status 这一步的 TD 通过设备向控制器发完成来结束一个控制传输，这一步的 TD 总是一个 Status 的 TRB，可能有一个 Event Data TRB。

下图是两个控制传输的例子。

Figure 7: Control Transfer Descriptor Example

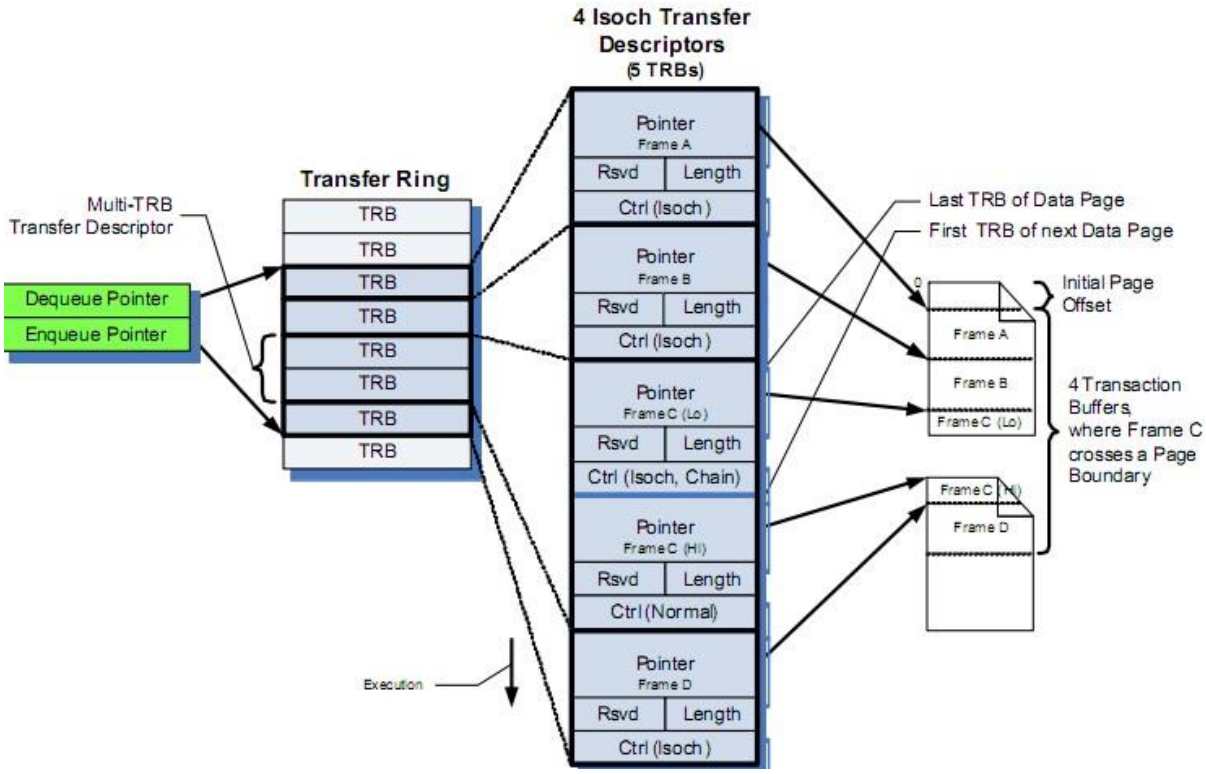


Bulk and Interrupt Transfers

用 Nomal TRB 来构成，根据 data buffering 的需求来确定要一个还是多个 TRB 来 构成一个 TD。多 TRB 的 Bulk 和中断 TD 可以实现 Scatter/Gather 操作。

Isoch Transfers

Figure 8: Isochronous Transfer Example



3.8.4 命令接口

软件通过 Command Ring 来管理控制器和连接其上的设备，Command Ring 上的每个元素称 为 CD（Command Descriptor）。基本工作方式：软件通过在 Command Ring 上放一个 CD 的方式发送命令，然后按 Host Controller 门铃。所有的命令都会在 Event Ring 上产生一个 Command Completion Event，用来报告命令的完成情况。控制器按照 Command Ring 上的顺序来执行命令。软件可以在 Command Ring 运行的过程中往其上增加 CD，如果软件要删除或对 CD 进行重排序， 正在执行的 CD 必须停下来。

	名称	描述

1	**No Op**	Exercise the TRB Ring mechanism，对控制器和设备的状态不产生影响。也可以用来报告当前 Command Ring 的 Dequeque 指针。
2	**Enable Slot**	软件通过这个命令来为一个有效的 Device Slot 获得一个
		ID 值。软件根据得到的 ID，加到 Device Context Base Address Array，来索引为这个设备新增的 Device Context。
3	**Disable Slot**	软件通知某个 Device Slot 已经不需要了，所有和这个设备有关的资源都可以释放。当一个设备从系统移出时，会使用这个命令。
4	**Address Device**	控制器将软件枚举 USB 设备时产生的 SET_ADDRESS 命令转换成这个命令。设备复位后都是 default 地址为 0，执行这个命令，控制器会向设备发 SET_ADDRESS 请求，为其分配一个唯一的地址，使设备从 Defualt 状态进入 Address 状态。 这个命令紧跟在 Enable Slot 命令之后。
5	**Configure Endpoint**	使能或是禁止设备的某个 Endpoint
6	**Evaluate Context**	告知控制器，Device Context 中的某个域需要修改。
7	**Reset Endpoint**	让 Endpoint 从 halted 条件下恢复过来
8	**Stop Endpoint**	用于管理 Transfer Ring，可以放弃、调整优先级、暂时停止其上 TD 的执行。
9	**Set TR Dequeue Pointer**	这个命令是对 Stop Endpoint 命令的补充，允许软件修改 pipei 的 Dequeue 指针，重新定向到 Transfer Ring 上的某个 TD 去执行。
10	**Reset Device**	通知控制器，某个 Device Slot 被复位。对应的 Slot state 进入 Defualt 状态，Device Address 变成 0。除了默认的控制 Endpoint，该 Device 的其它 Endpoint 都被 Disable 掉了。
11	**Force Event**	这个命令是控制器可选的，仅在控制器的虚拟化特征使能时。允许 VMM 将 USB 设备枚举到虚拟机。
12	**Negotiate Bandwidth**	这个命令是控制器可选的，用于在一个运行中的系统上恢复 USB 带宽。

13	**Set Latency Tolerance Value**	通过这个命令在设置软件定义的 Best Effort Latency Tolerance 值
14	**Get Port Bandwidth**	软件通过这个命令得到 Root Hub 上每个端口的有效周期带宽。如果设备因为带宽得不到满足而被枚举不上时，软件可以用这个命令来 recommend topology changes to the user。
15	**Force Header**	用于向某个选中的端口发 LMP 或 TP。

3.8.5 工作流程

软件发命令，实际上是将 Command TRBs 放到 Command Ring 上，然后再按 Command 门铃寄存器。例如将控制器命令代码，写到 0 号门铃寄存器的 DB Target 域。

所有的命令执行完，都会在 Event Ring 上产生一个命令完成事件。

3.8.5.1 xHC 初始化

系统 boot 起来后，软件会先枚举到控制器，为 xHC 的寄存器分配一个基址，为 FLADJ 寄存器设置一个系统规定的值。

然后进行以下一系列操作，实现控制器的初始化：

1、 初始化系统 IO 映射；

2、 芯片硬复位之后，等待 USBSTS 寄存器中 CNR（Controller Not Ready）置 1； 3、 配 CONFIG 寄存器中 MaxSlotsEn 域，指明软件将可使用的最大 Slot 号；

4、 为 DCBAAP 寄存器配一个 64 位的地址，指向 Device Context 基址阵列的所在；

5、 为 Command Ring 控制寄存器配一个 64 位的地址，这是 Command Ring 中第一个 TRB 的起始地址；

6、 初始化中断：

a) 分配、初始化 MSI-X 消息表，设置消息地址和消息数据，使能向量。至少，表向量条 目 0 要被初始化、使能；

b) 分配、初始化 MSI-X Pending Bit Array；

c) 在 MSI-X 能力结构体中分别为消息控制表和 Pending Bit Array 指定 Offset； d) 初始化 MSI-X 能力结构体中的消息控制寄存器；

e) 初始化每一个活跃的中断源：

i. 定义 Event Ring：

1. 分配并初始化 Event Ring Segment(s)；
2. 分配 ERST（Event Ring Segment Table），初始化 ERST 表的各个条目，指向 并定义各个 Event Ring Segment 的大小；
3. 根据 ERST 中定义的大小，配置中断源 ERSTSZ（Event Ring Segment Table Size）寄存器；
4. 根据 ERST 中定义的起始地址，配置中断源 ERDP（Event Ring Dequeque Pointer）寄存器；
5. 用一个 64 位的地址指针指向 ERST，即配置中断源 ERSTBA（Event Ring Segment Table Base Address）寄存器；写 ERSTBA 寄存器，则使能了 Event Ring；

ii. 定义中断：

1. MSI-X 能力结构体消息控制寄存器中的 MSI-X 使能标志设置为 1，使能MSI-X 中断机制；
2. 根据目标中断调节率，初始化中断调节寄存器的 Interval 域；
3. 置 USBCMD 寄存器的 Interrupt 使能位 INTE 为 1，使能系统总线中断使能；
4. 置中断管理寄存器的 Interrupt 使能位 IE 为 1，使能中断源；

7、 置 USBCMD 寄存器的 Run/Stop 位为 1，相当于打开控制器，表明控制器可以开始接收按门 铃了。

至此，控制器已经开始工作，Root Hub 端口开始报告设备的连接情况了，系统软件可以开 始枚举设备。

USB2.0 设备需要端口复位过程，使端口进入 Enabled 状态，一旦端口进入 Enabled 状态， 端口上就会有有效的 SOF 了，但是 pipe 调度还没有使能；SS 端口一旦检测到设备连接，可以 直接进入 Enabled 状态。

3.8.5.2 设备初始化

这里讨论的是 Root Hub port 上的设备初始化。

芯片硬件复位（HCRST, or commanded to the PLS = RxDetect state）之后，所有的 Root Hub 端口将处于 Disconnected 状态，端口处于上电状态（PP=1），等待设备连接。
当一个设备连接到处于 Disconnected 状态时：

3.0 端口将：

进入 Polling 状态

如果 Polling 成功，端口进入 Enable 状态，CCS 和 CSC 被置 1； 如果 Polling 不成功，端口进入 Disconnected 状态；

2.0 端口将：

进入 Disabled 状态，CCS 和 CSC 被置 1；

下述为一个典型的设备初始化过程：

1、 当控制器监测到一个设备连接时，它将置 CCS 和 CSC 为 1。如果 CSC 的置 1，导致 PSCEG(Port Status Change Event Generation)有一个从 0 到 1 的变化，则，控制器会生成一个 Port Status Change Event；

2、 一旦收到 Port Status Change Event，系统软件分析 PORT ID，确定是哪个端口给出的这个事 件；

3、 然后软件来读这个端口的 PORTSC 寄存器，如果 CSC=1 且 CCS=1，表明这个 Event 的产生 是由于设备连接；如果 CSC=1 且 CCS=0，表明这个 Event 的产生是由于设备移除；现在讨 论设备连接：

a) 3.0 端口将自动向 Enabled 状态转；
如果成功，端口进入 Enabled 状态，Port Enabled/Disabled（PED）位将置 1，Port Rest(PR) 和 Port Link State(PLS)域将被清 0。设备将处于 Default 状态；
如果不成功，端口进入 Disconnected 状态，PED 和 PR 位将置 0，PLS 域将被置成 RxDetect (_5‘)。设备将仍处于 Powered 状态；

b) 2.0 端口需要软件来启动端口复位，促使端口向 Enabled 状态跳转、设备向 Default 状 态跳转。
设备连接后，PED 和 PR 都是 0，PLS 域为 7(polling)。软件置 PR 为 1，启动端口复位，等待 Port Status Change Event。
复位完成后，PRC 和 PED 将被置 1，PR 将被清 0，PLS 为 0(U0)。如果 PRC 的置 1， 导致

PSCEG 有一个从 0 到 1 的变化，则，控制器会生成一个 Port Status Change Event。 复位操作使 2.0 设备进入 Default 状态，准备接收 SET_ADDRESS 请求。

4、 端口成功进入 Enabled 状态后，软件通过 Enable Slot 命令，为新连接上的设备得到一个 Device Slot；

5、 成功获得 Device Slot 之后，软件为其初始化各种数据结构；

6、 Solt 对应的各种数据结构都初始化好了之后，软件用一个 Address Device 命令为 Device 分 配一个地址，使能设备的默认的控制端点；

7、 对于 LS、HS、SS 设备，对默认的控制端点的访问包有唯一允许的大小，分别为 8B、64B、512B，下面的步骤 a 可能跳过；

对于 FS 设备，软件通过发 GET_DESCRIPTOR 请求，先读设备描述符的前 8 个字节，获 得 bMaxPacketSize0 域，来确定对默认的控制端点的访问包的实际 Max Packet Size，更新 到 Default Control Endpoint Context

a) GET_DESCRIPTOR 请求需要一个 Setup Stage TD、一个 Data Stage TD、一个 Status Stage TD。软件将：

i. 分配一个 8 个字节的 buffer 准备接收 Device Descriptor；

ii. 在 Endpoint 0 的 Transfer Ring 上初始化 Setup Stage TD（单个的 Setup TRB）

1. TRB Type = Setup Stage TRB；
2. TRB Transfer Length = 8；8B 的 Setup 数据
3. IOC =0；
4. IDT =1；
5. bmRequestType = 0x80(Dir = Device-to-Host, Type = Standard, Recipient = Device)；
6. bRequest = 6 (GET_DESCRIPTOR)；
7. wValue = 0100h. Low byte = 0 (Descriptor Index), High Byte = 1 (Descriptor type)
8. wIndex = 0
9. wLength = 8.
10. Cycle bit = Current Producer Cycle State

iii. 调整（Advance）Endpoint 0 的 Transfer Ring 的 Enqueue Pointer；

iv. 在 Endpoint 0 的 Transfer Ring 初始化 Data Stage TD（单个的 Data Stage TRB）

1. TRB Type = Data Stage TRB
2. Direction (DIR) = ‗1‘.
3. TRB Transfer Length = 8 8B 的 Descriptor 数据
4. Chain bit (CH) = 0
5. Interrupt On Completion (IOC) = 0
6. Immediate Data (IDT) = 0
7. Data Buffer Pointer = The address of the Device Descriptor receive buffer.

8. Cycle bit = Current Producer Cycle State.

v. 调整（Advance）Endpoint 0 的 Transfer Ring 的 Enqueue Pointer；

vi. 在 Endpoint 0 的 Transfer Ring 初始化 Status Stage TD（单个的 Status Stage TRB）

1. TRB Type = Status Stage TRB
2. Direction (DIR) = ‗0‘.
3. TRB Transfer Length = 0.
4. Chain bit (CH) = 0.
5. Interrupt On Completion (IOC) = 1.
6. Immediate Data (IDT) = 0.
7. Data Buffer Pointer = 0.
8. Cycle bit = Current Producer Cycle State.

vii. 调整（Advance）Endpoint 0 的 Transfer Ring 的 Enqueue Pointer；

viii. 摁该 Device Slot 对应的门铃，DB Target = Control EP 0 Enqueue Pointer Update.

ix. 当 GET_DESCRIPTOR Status Stage TRB 对应的传输完成 Event 返回时，软件将用 Device Descriptor buffer 中的 wMaxPacketSize 值来更新 Endpoint 0 Context 里的 Max Packet Size；

x. 软件接着会发出一个 Evaluate Context Command，V1 置 1，用来通知 xHC，Default Control endpoint 的 Max Packet Size 这个参数发生了改变。xHC 执行完这个命令后， 后续的经过 Default Control endpoint 的传输，将都使用这个新的 Max Packet Size。

8、 现在默认的控制端点（Default Control endpoint）已经完全可以进行所有的操作了。软件可 以读完整的设备描述符、配置描述符等，将设备交给对应的类驱动接管。通过 Default Control endpoint 发送 GET_DESCRIPTOR 请求即可。

9、 读完配置描述符，软件会发一个 Evaluate Context Command，V0 置 1，用来通知 xHC，Hub and Max Exit Latency 这些参数。Output Slot Context 的 Interrupter Target 域，也可以通过这 个命令来修改。

10、 现在，类驱动可以用 Configure Endpoint Command 来配置 Device Slot，以及通过 SET_CONFIGURATION 来配置设备本身（经 Default Control Endpoint）。这两个动作都成功 完成了，则设备状态将从 Address 跳到 Configured，而 xHC Device Slot 的状态将从 Addressed 跳到 Configured。

11、 如果需要的话，软件还要配置 Alternate Interface；

12、 现在个设备的 pipe interface 已经可以完成正常使用了。

注意，软件为了保证行为的正确性，从 Root hub 开始，会对 USB 拓扑中的每一层 hub 和 TTs（Transaction Translator）都做初始化之后，才开始做再下一层的设备初始化。

3.8.5.3 设备移除

设备移除时，PORTSC 寄存器中 CCS 清零，CSC 置 1。控制器报告一个 Port Status Change Event，软件发一个 Disable Slot 命令，将相关信息 disable 掉。

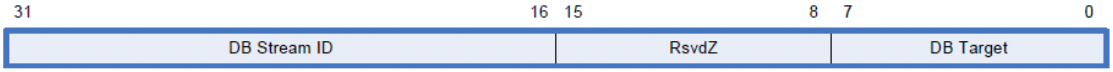
3.8.5.4 Device Slot 管理

最多 255 个 slot。每个 slot 对应一个设备，主要有三部分组成：

DCBAA 中一个条目，一个 Device Context 数据结构，一个 doorbell 寄存器。 软件用 slot ID 来索引 DCBAA，得到 Device Context 地址和 doorbell 寄存器地址。

3.8.5.5 门铃

Figure 70: Doorbell Register



一共 256 个 32 位的门铃寄存器，0 号是控制器的，其余 255 个由 slot ID 索引。

PCI Config space 中的 Base Address Register 中，定义了 XHCI 的 MMIO 基址：**Base** 读 addr[Base + 14h]，即 DBOFF 寄存器，得到 Doorbell Array 基址：

Doorbell Array Base = Base+DBOFF[31:0]；

DB Target 表明按门铃的原因：

Device Context Doorbells (1-255)

Value	Definition
0	Reserved
1	Control EP 0 Enqueue Pointer Update
2	EP 1 OUT Enqueue Pointer Update
3	EP 1 IN Enqueue Pointer Update
4	EP 2 OUT Enqueue Pointer Update
5	EP 2 IN Enqueue Pointer Update
…	…
30	EP 15 OUT Enqueue Pointer Update
31	EP 15 IN Enqueue Pointer Update
32:247	Reserved
248:255	Vendor Defined

Host Controller Doorbell (0)

Value	Definition
0	Command Doorbell
1:247	Reserved
248:255	Vendor Defined

DB Stream ID 仅对 MaxPStreams > 0 的 SS Bulk 端点有意义，指明是哪个 stream。

3.9 低速设备

3.9.1 Super IO

3.9.1.1 PS/2 接口

PS/2 设备接口用于一些鼠标和键盘，是由 IBM 开发。PS/2 鼠标和键盘履行一种双向同步 串行协议。每次数据线上发送一位数据并且每在时钟线上发一个脉冲就被读入。键盘/鼠标可以 发送数据到主机，而主机也可以发送数据到设备，但主机总是在总线上有优先权，主机可以在 任何时候抑制来自于键盘/鼠标的通讯，只要把时钟拉低即可。

从键盘/鼠标发送到主机的数据在时钟信号的下降沿被读取；从主机发送到键盘/鼠标的数据 在上升沿被读取。不管通讯的方向怎样，键盘/鼠标总是产生时钟信号。如果主机要发送数据， 它必须首先告诉设备开始产生时钟信号。

键盘和鼠标在通过 MSI 中断方式发给 CPU 时，MSI 中断信息中包含了区分键盘还是鼠标 的信息，软件只需直接读取 60h 寄存器中的数据即可，完成 60h 寄存器读后，PS2 接口自动清 除对应的

中断。

而当采用 INTx（共享）中断时，软件需要首先读 64h 寄存器中的值来区分是键盘还是鼠标 发出了中断，然后再读取 60h 寄存器中的值，之后 PS2 接口自动清除对应的中断。

PS2 接口设备的时钟最高频率是 33K，一般工作在 10KHz~20KHz。ICH2 芯片低速 IO 子系统核心工作频率 100M，分频系数为 500（devide_reg），能够得到 5 微秒的计时值（200KHz）。寄存器说明：

LegcyIO 地址	Wishbone 子系统内部地址	位宽	属性	功能说明
60H	0x0000_0060	8	R	读输入缓冲区
60H	0x0000_0060	8	W	写输出缓冲区
64H	0x0000_0064	8	R	读状态寄存器
64H	0x0000_0064	8	W	发送命令
68H	0x0000_0068	12	RW,0x1f4	配置 5 微秒的分频系数

发送命令给键盘控制器是写 64h 寄存器，在写命令之前要测试状态寄存器的 OBF(输出缓冲 区满)是否置起。如果要往 PS2 接口键盘鼠标设备发送数据，则写 60h 输出缓冲寄存器，读 60h 输入缓冲寄存器则可以获得外部输入的数据。

3.9.1.2 UART 接口

UART（Universal Asynchronous Receiver/Transmitter）接口提供串行通信能力，可以和 modem 或其它外部设备，例如使用串口线实现 RS232 协议的另一台计算机通信。ICH2 芯片的 uart 接 口实现的是全功能的串口，也可以通过该 uart 接口对 SPI 接口 Flash 进行编程。uart 接口只实现 FIFO mode。

寄存器说明：

Mem IO 地址	wishbone 子系统内部地址	位宽	属性	功能说明
0x3F8	0x1000-0000	8	R	Receiver Buffer
0x3F8	0x1000-0000	8	W	Transmitter Holding Register (THR)
0x3F8	0x1000-0000	8	RW	Divisor Latch Byte1
0x3F9	0x1000-0001	8	RW	Interrupt Enable
0x3F9	0x1000-0001	8	RW	Divisor Latch Byte2
0x3FA	0x1000-0002	8	R	Interrupt Identification

0x3FA	0x1000-0002	8	W	FIFO Control
0x3FB	0x1000-0003	8	RW	Line Control Register
0x3FC	0x1000-0004	8	W	Modem Control
0x3FD	0x1000-0005	8	R	Line Status
0x3FE	0x1000-0006	8	R	Modem Status

如果 Line Controll Register 的 bit7 设置成“1”，那么 0x3F8 和 0x3F9 的 divisor latch 寄存器 就可以被访问，否则这两个寄存器不可见。UART 接口正常工作操作流程如下：

- 设置 Line Control Register 为一个合适的 line control 参数值，设置 bit7 为“1”允许访 问 divisor latch 寄存器；
- 设置 divisor latch 寄存器，先写 byte2，再写 byte1；
- 设置 LCR 的 bit7 为“0”禁止访问 divisor latch 寄存器，这时传输引擎开始工作数据可 以正常发送和接收；
- 设置 FIFO Control 寄存器的 FIFO trigger level，如果这个值设的比较大，发到系统的中 断会比较少一些，推荐该值设为 14 bytes；
- 设置 Interrupt Enable 寄存器的相应位，使能对应位的中断；
- 判断 LSR[5]位的 Transmit Holding Register Empty 是否为“1”，“1”代表 THR 寄存器是空的，可以写数据到该寄存器，写 THR 寄存器的数据会被控制器输出到 uart 接口；
- 读取 LCR 寄存器的 Data Ready 位的值，如果为“1”，则可以读 Receive Buffer 寄存器 获得输入数据，只要 Data Ready 位为“1”就可以一直读取 Receive Buffer 寄存器获得 数据。

3.9.2 LPC

3.9.2.1 基本功能

ICH2 芯片将实现 LPC-HOST 功能，该功能下包含 LPC 控制器和 LPC-DMA 控制器，实现符合 LPC 总线标准规范的 LPC 总线接口。 **特点：**

- 基于 Intel LPC 总线规范（Intel Low Pin Count Interface Specification,Revision1.1）。
- 支持 I/O、Memory、DMA、FHW 四中周期类型，不支持 Bus Master 周期类型。
- 内部实现 4 个通道的 LegacyDMA 控制器。
- 支持 Serial IRQ。
- 支持在 target（I/O、Memory 和 DMA）方式下设备超时后由 Host 主动终止总线周期，

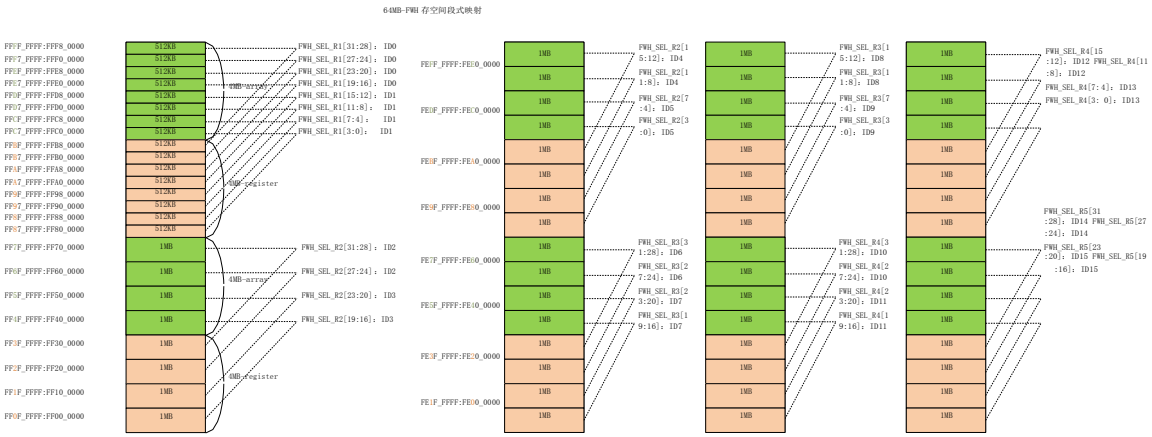
并返回全“1”数据（兼容 ISA 标准）的控制。

- LegacyIO 访问粒度大于 8bit 时（2B、4B），自动拆分成多个 IO 周期，并按访问粒度返回数据。
- 支持 256MB 窗口的 Memory 直接寻址（ICH10:64KB），64MB 窗口的 FWH 分段段空间寻址（ICH10:16MB）。

LPC-HOST 占用一个 PCI 功能，实现 64KB 的 LegacyIO。占用三个 BAR：LPC_HOST_BAR 大小 64KB，LPC_MEM_BAR 大小 256MB，LPC_FHW_BAR 大小 64MB。其中 64KB 的 LPC_HOST_BAR 用于 LPC-HOST 内部寄存器访问，256MB 的 LPC_MEM_BAR 用于 LPC 总线的 MemoryIO 周期寻址， 另外 64MB 的 LPC_FHW_BAR 用于 LPC 总线的 FHW 周期寻址。

LPC-HOST 支持 1B、2B、4B 粒度的 LegacyIO 访问，控制器收到 2B 和 4B 的 LegacyIO 访问时， 自动拆分成 2 个和 4 个 IO 周期发送给 LPC 设备，所有 IO 周期完成后给系统返回对应粒度的响应。

LPC-HOST 仅支持 256MB 窗口的直接 Memory 周期寻址，高 4 位地址必须为全 0 或全 1。 LPC-HOST 支持 64MB 窗口的 FWH 周期寻址，为了支持多设备（16 片 16Mbit 的 flash）访问，控制器内部对 64MB 空间划分为不同粒度的段式地址，通过对每段地址的 ID 配置，实现 64MB 空间到到不同设备的地址映射，如下图所示：



3.9.2.2 基本操作流程

一、LPC-HOST 内部寄存器访问流程

系统用 64 位 EP 存储空间地址 LPC_HOST_BAR+xxxx 访问 LPC-HOST 内部寄存器，PAB 将其转 换成 AMBA 地址 0x4000,xxxx，AHB 总线将其路由到 S8 端口，LPC-HOST 收到请求后根据地址[31:28] 译码，为 0x4 则周期认为是访问 LPC-HOST 内部寄存器，不生成 LPC 总线。

二、LegacyIO 访问流程（对应 LPC 总线的 IO 周期）

LPC-HOST 控制器实现了 64KB 的 LegacyIO 空间。由于套片自带了 PS2、UART 和 VGA 兼容相 关 IO，占用了 LegacyIO 中的 KBC（60H 和 64H）、 SerailPort（3F8h-3FFh）， VGA（3B0h-3BBh 和 3C0h-3DFh）， 默认情况下这些 LegacyIO 往低速 IO 的 PS2、UART 以及片内的显示设备 路由，当 PAB 中 LegacyIO 路由选择寄存器的低 4 位对应位置为“1”时则往 LPC-HOST 控制器路由。

系统用 EP 的 IO 地址空间进行 LegacyIO 访问，PAB 负责将 LegacyIO 的地址加上 0x1000,0000 的 AMBA 地址头后上 AHB 总线，AHB 总线将其路由到 S8 端口，LPC-HOST 收到请求后译码[31:28]， 为 0x1 则向 LPC 总线发起 IO 周期。

三、MemoryIO 访问流程（对应 LPC 总线的 Memory 周期）

系统用 64 位 EP 存储空间地址 LPC_MEM_BAR+xxx_xxxx 访问 LPC 设备的存储映射空间，PAB 截取地址[27:0]拼上 0x2000,0000 的 AMBA 地址头发送给 AHB 总线，AHB 总线将其路由到 S8 端 口，LPC-HOST 收到请求后译码[31:28]=0x2，则发起地址为{Haddr[3:0],addr[27:0]}的 LPC-Memory 周期。

四、FWH 访问流程（对应 LPC 总线的 FWH 周期）

驱动根据 LPC 总线上的 flash 设备的数量和单片的容量，配置 LPC-HOST 中段地址与设备 ID 的映射表。系统用64 位 EP 存储空间地址 LPC_FHW_BAR+xxx_xxxx 访问 LPC 设备的存储映射空间， PAB 截取地址[27:0]拼上 0x3000,0000 的 AMBA 地址头发送给 AHB 总线，AHB 总线将其路由到 S8 端口，LPC-HOST 收到请求后译码[31:28]=0x3，查询段地址映射表，生成 IDSEL， 发起地址为[27:0]的 LPC-FHW 周期。

五、DMA 访问流程（对应 LPC 总线的 DMA 周期）

LPC-HOST 最多支持两个有 DMA 功能的 LPC 设备，这两个设备的 DMA 请求信号分别为 LDRQ0 和 LDRQ1。在允许 LPC 设备发起 DMA 申请前，驱动需要申请一个空闲的通道号，初始化该通道 的相关寄存器（如传输类型，主存基址，传输长度等），并告诉 LPC 设备可以使用的通道号。LPC 设备根据约定的通道号，通过 LDRQ#发起 DMA 传输申请，LPC-HOST 中的 DMA 控制器仲裁上该 通道号后发起 DMA 传输，传输完成后 LPC-HOST 仅置完成状态供中断处理程序查询，由 LPC 设 备主动发起 DMA

完成中断。

一个完整的 DMA 读和写请求工作流程如下：

(1) 系统复位结束后，所有通道处于 free 状态（通道状态寄存器所有位为“0”-free），所 有通道的屏蔽位为 1（通道屏蔽寄存器的所有位为“1”-屏蔽：不允许申请）。LPC 设备对处于屏蔽状态的通道号发起的 DMA 申请，LPC-HOST 将登记该通道的 DMA 申请错，并报告给系统。

(2) 驱动配置 DMA 控制寄存器（DMA_CTRL：包括通道间的仲裁原则等信息），读取通道状 态寄存器（Ch_Status），分配处于空闲的通道号。开始初始化 LPC 控制器，首先置该通 道的屏蔽寄存器，不允许对该通道申请 DMA 请求。填写该通道号对应的传输模式寄存 器（DMA_Mode：包括读写类型、DMA 周期长度等）、主存基址寄存器（MEM_Base） 和传输字节长度寄存器（Byte_Cnt），最后清除对应通道的屏蔽位，允许申请该通道进 行 DMA 传输。

(3) 驱动通过 Memory 或 IO 方式，配置要发起 DMA 请求的 LPC 设备的相关寄存器（如通 道号，一次 DMA 周期的传输长度等信息），启动设备发起 DMA 请求，设备置通过专用 的 LDRQ# 申请某通道的 DMA 传输；

(4) LPC Host 控制器检测到 LDRQ0 和 LDRQ1 信号其中一个有效或两个同时有效，分别解析 出 DMA 请求的通道号；

(5) 控制器的仲裁逻辑根据 DMA 控制寄存器（DMA_CTRL）中确定的仲裁原则，以及通道 屏蔽状态对多个通道的 DMA 请求进行仲裁，仲裁成功以硬件自动修改通道状态寄存器 中相应通道的忙闲位（置为“1”-busy），LPC-HOST 根据该通道的传输模式寄存器、主 存基址寄存器和传输长度寄存器发起 DMA 请求；

(6) 如果传输模式寄存器的读写类型是 DMA 读请求：

a) 根据通道描述符信息（周期长度，主存基址）发起 AHB 读请求，等待 AHB 总线返 回读响应；

b) 收到 AHB 读响应后，向 LPC 总线发起相应长度的 DMA 读周期的，由发出该通道 请求的设备自行认领，控制器开始向 LPC 总线发送最后一个传输时，置 LPC 总线 DMA 周期的 TC 标志（CHANNEL 周期的 LDA[3]=1），同时，置状态寄存器对应的完 成位为“1”，置对应的忙闲位为“0”-free；

(7) 如果传输模式寄存器的读写类型是 DMA 写请求：

a) 根据通道描述符信息，向 LPC 总线发起该通道相应长度的 DMA 写周期，由申请该 通道的 LPC 设备认领并给出 DMA 写数据，LPC Host 控制器收下写数据；

b) 根据通道描述符信息，向 AHB 总线发起对应长度的写请求；

c) 根据传输计数，控制器开始向 LPC 总线发送最后一个传输时，置 LPC 总线 DMA 周 期的 TC 标志，收到来自 LPC 设备的最后一个 DMA 写数据和 AHB 写响应后，置状 态寄存器

对应的完成位置为“1”，置对应的忙闲位为“0”-free。

(8) LPC 的 DMA 周期传输完成后（LPC 设备收到带 TC 标记的 DMA 周期，并完成数据传输），由 LPC 设备通过 SIRQ 主动发起 DMA 完成中断，LPC-HOST 收到 SIRQ 后在中断源寄存器 中登记对应设备的中断请求有效位，若中断没有屏蔽，则向系统报告 LPC 中断。驱动根据收到 INTx 中断线或 MSI 中断请求包，进入中断处理程序。中断处理程序查询 PAB 中的中断源寄存器，确定中断来自 LPC-HOST，进一步查询 LPC-HOST 控制器的中断源寄存器，确定中断来自某 LPC 设备，并查询该 LPC 设备的中断源寄存器，如果是 DMA 写中断，且该中断对应的通道传输完成状态为“1”（如何将设备与通道号挂钩？上一个 DMA 中断没有处理完，不允许配置下一个 DMA 请求），才能消费此次 DMA 写数据。

五、SIRQ 工作流程

LPC 支持 Serial IRQ，通过一根双向信号 SERIRQ 来实现中断请求。Serial Interrupt 信息通过 3 个帧类型（Frame）来传输：一个 start frame，多个 IRQ Data frames，和一个 stop frame。包含 两种模式：Quiet mode（由 LPC 设备启动），Continuous mode（由 LPC HOST 启动）。系统复位后 处于 Continuous mode 模式，但不工作，当 LPC 控制寄存器中的 SIRQ_EN 位被置 1 后，SIRQ 将 根据 SIRQ_MODE 位的置来区分 Quiet mode 与 Continusou mode。若是 Continuous mode，则 LPC Host 不断地启动 SIRQ 检测状态机，向 LPC 外设索取中断信息；若是 Quiet mode，则由 LPC 外设 启动 SIRQ 检测状态机，主动向 LPC Host 申请中断。

LPC 设备若需要进行数据传输时，先通过 SERIRQ 信号线按照 Serial IRQ 协议向 LPC HOST 发 出 LPC 外设中断请求，驱动根据收到的 INTx 中断线，查询 LPC-HOST 控制器的 Interrupt Source 寄存器，确定 LPC 设备中断源，然后对 LPC 设备发起 Memory 或 IO 读写周期。驱动对中断源进 行写 1 清 0。

表 LPC 设备中断源映射

LPC INT Sources	Legacy IRQ
LPC IRQ0	8254 Timer
LPC IRQ1	Keyboard
LPC IRQ2	-
LPC IRQ3	UART
LPC IRQ4	UART
LPC IRQ5	Parallel Port2
LPC IRQ6	Floppy
LPC IRQ7	Parallel Port1
LPC IRQ8	RTC
LPC IRQ9~11	-
LPC IRQ12	Mouse

LPC IRQ13	FPU
LPC IRQ14	Primary IDE
LPC IRQ15	Secondary IDE

六、周期异常终止及主动退出流程（Abort、stop、DMA_inactive、FHM_Abort）

1、 周期异常终止（Abort）

在周期出现异常时，LPC Host 能够主动终止并退出周期。以下情况出现时，LPC Host 将终止 异常周期：

（1） LPC 设备返回的 SYNC 信号超时（SYNC Time-out）

- LPC HOST 发起任意一个周期后，连续 3 拍内没有收到有效的 SYNC，则主动终止当前 周期；
- LPC HOST 发起任意一个周期后，3 拍内收到有效的 SYNC（短等待模式，SYNC 值为 4’b0101），但短等待时间超过 8 拍，则主动终止当前周期；
- LPC HOST 发起任意一个周期后，3 拍内收到有效的 SYNC（长等待模式，SYNC 值为 4’b 0110），则 LPC HOST 不会主动终止周期，因此 LPC 外设必须保证周期的完成。

（2） LPC 设备返回保留的 SYNC

当 LPC 设备返回的 SYNC 值为保留值（4’b0001~4’b0100 或 4’b1011~4’b1111）时，LPCHOST 主动终止当前周期。

（3） LPC 设备返回 Error 的 SYNC

当 LPC 设备返回的 SYNC 值为“4’b1010”（error）时，LPC HOST 主动终止当前周期。

2、 DMA 传输取消（DMA Deassertion）

当软件配置 LPC 设备启动 DMA 传输后，LPC 设备通过 LDRQ 信号请求 DMA 传输，LPC HOST 收到 DMA 请求后立刻进行仲裁；若此时软件取消 DMA 传输，LPC 设备通过 LDRQ 信号请求取消DMA 传输，但之前的 DMA 请求很可能已经仲裁成功并开始启动传输，此时 LPC 设备无需向 LPC HOST 反馈响应，LPC HOST 便会自动取消 DMA传输。

3、 Firmware Memory Write Abort

对于 Firmware Memory 写周期，只允许 memory 设备返回 1 个 clock 周期的“SYNC”，且其 值必须为“4’b0000”（ready）或“4’b1010”（error），否则异常终止并退出当前周期。

3.9.2.3 LPC-HOST 内部寄存器

表# LPC HOST 寄存器定义

Offset	Type	Name	Description	Default value
00h	RW	LPC Ctrl	[31:4]：Reserved [3]：Serial IRQ Enable. 0 – Disable; 1 – Enable. [2]：Serial IRQ Mode. 0 – Continuous (Idle); 1 – Quiet (Active). [1:0]: Start Frame Pulse Width.设置 start frame 宽度，指定其占用的时钟个数。 00: 4 clocks 10: 8 clocks 10: 6 clocks 11: Reserved	0x0
04h	RO	LPC _Status	[31:16]：保留 [11:8]：周期 size [7:4]：周期类型 [3:0]：错误指示, 00–正常, 01 -超时, 10-错误完成, 11-请求/地址异常（对于 IO 请求来说是地址异常；对于 DMA 请求来说是请求异常，如如申请了屏蔽位为 1 的通道号；对于 Memmory 和 FWH 周期来说无意义）。其他编码保留	0000000h
08h	RO	LPC_Einf	如果是 IO 周期： [31:16]：保留. [15:0]：IO 周期出错时的地址如果是 MEM 周期：[31:0]：MEM 周期出错时的地址 如果是 FWH 周期：[31:28]：IDSEL. [27:0]：FWH 周期出错时的地址 如果是 DMA 周期：[31:4]：保留 [3:0]：DMA 周期出错时的通道号	0000000h
0ch	RW	MMHaddr	[31:28]:LPC-Memory 地址高 4 位其他 reserved[0]：高位地址替换使能位，为 1 表示高 4 位地址采用配置值，为 0 表示高 4 位地址自动从[27]位扩展。	0xF0000000

10h	RW	FWH_SEL_R1	FWH_SEL_R1[31:28]：FWH_F8_IDSEL 0xFFF8_0000 – 0xFFFF_FFFF：512KB 0xFFB8_0000 – 0xFFB8_FFFF：512KB 0x000E_0000 – 0x000F_FFFF：128KB FWH_SEL_R1[27:24]：FWH_F0_IDSEL 0XFFF0_0000 – 0xFFF7_FFFF：512KB 0xFFB0_0000 – 0xFFB7_FFFF：512KB FWH_SEL_R1[23:20]：FWH_E8_IDSEL 0XFFE8_0000 – 0xFFEF_FFFF：512KB 0xFFA8_0000 – 0xFFAF_FFFF：512KB FWH_SEL_R1[19:16]：FWH_E0_IDSEL 0xFFE0_0000 – 0xFFE7_FFFF：512KB 0xFFA0_0000 – 0xFFA7_FFFF：512KB FWH_SEL_R1[15:12]：FWH_D8_IDSEL 0XFFD8_0000 – 0xFFDF_FFFF：512KB 0xFF98_0000 – 0xFF9F_FFFF：512KB FWH_SEL_R1[11:8]：FWH_D0_IDSEL 0xFFD0_0000 – 0xFFD7_FFFF：512KB 0xFF90_0000 – 0xFF97_FFFF：512KB FWH_SEL_R1[7:4]：FWH_C8_IDSEL 0XFFC8_0000 – 0xFFCF_FFFF：512KB 0xFF88_0000 – 0xFF8F_FFFF：512KB FWH_SEL_R1[3:0]：FWH_C0_IDSEL 0xFFC0_0000 – 0xFFC7_FFFF：512KB 0xFF80_0000 – 0xFF87_FFFF：512KB	

14h	RW	FWH_SEL_R2	FWH_SEL_R2[15:12]：FWH_70_IDSEL 0xFF70_0000 – 0xFF7F_FFFF：1MB 0xFF30_0000 – 0xFF3F_FFFF：1MB FWH_SEL_R2[11:8]：FWH_60_IDSEL 0XFF60_0000 – 0xFF6F_FFFF：1MB 0xFF20_0000 – 0xFF2F_FFFF：1MB FWH_SEL_R2[7:4]：FWH_50_IDSEL 0XFF50_0000 – 0xFF5F_FFFF：1MB 0xFF10_0000 – 0xFF1F_FFFF：1MB FWH_SEL_R2[3:0]：FWH_40_IDSEL 0xFF40_0000 – 0xFF4F_FFFF：1MB 0xFF00_0000 – 0xFF0F_FFFF：1MB	
18h	RW	FWH_SEL_R3	按以上规律待细化	
20h	RW	FWH_SEL_R4	按以上规律待细化	
24h	RW	FWH_SEL_R5	按以上规律待细化	

18h	RW	FWH_DEC_EN1	WH_DEC_EN [15]：FWH_F8_EN 0 = 关闭；1 = 使能 FWH 周期的以下 512KB 地址段： FFF80000h – FFFFFFFFh FFB80000h – FFBFFFFFh WH_DEC_EN [14]：FWH_F0_EN 0 = 关闭；1 = 使能 FWH 周期的以下 512KB 地址段： FFF00000h – FFF7FFFFh FFB00000h – FFB7FFFFh WH_DEC_EN [13]：FWH_E8_EN 0 = 关闭；1 = 使能 FWH 周期的以下 512KB 地址段： FFE80000h – FFEFFFFh FFA80000h – FFAFFFFFh WH_DEC_EN [12]：FWH_E0_EN 0 = 关闭；1 = 使能 FWH 周期的以下 512KB 地址段： FFE00000h – FFE7FFFFh FFA00000h – FFA7FFFFh WH_DEC_EN [11]：FWH_D8_EN 0 = 关闭；1 = 使能 FWH 周期的以下 512KB 地址段： FFD80000h – FFDFFFFFh FF980000h – FF9FFFFFh WH_DEC_EN [10]：FWH_D0_EN 0 = 关闭；1 = 使能 FWH 周期的以下 512KB 地址段： FFD00000h – FFD7FFFFh FF900000h – FF97FFFFh WH_DEC_EN [9]：FWH_C8_EN 0 = 关闭；1 = 使能 FWH 周期的以下 512KB 地址段： FFC80000h – FFCFFFFFh FF880000h – FF8FFFFFh WH_DEC_EN [8]：FWH_C0_EN 0 = 关闭；1 = 使能 FWH 周期的以下 512KB 地址段： FFC0 0000h – FFC7 FFFFh FF80 0000h – FF87 FFFFh WH_DEC_EN [7:4]：保留 WH_DEC_EN [3]：FWH_70_EN 0 = 关闭；1 = 使能 FWH 周期的以下 1M 地址段： FF70 0000h – FF7F FFFFh FF30 0000h – FF3F FFFFh FWH_DEC_EN [2]：FWH_60_EN 0 =关闭；1 =使能 FWH 周期的以下 1M 地址段： FF60 0000h – FF6F FFFFh FF20 0000h – FF2F FFFFh FWH_DEC_EN [1]：FWH_50_EN 0 =关闭；1 =使能 FWH 周期的以下 1M 地址段： FF50 0000h – FF5F FFFFh FF10 0000h – FF1F FFFFh FWH_DEC_EN [0]：FWH_40_EN 0 =关闭；1 =使能 FWH 周期的以下 1M 地址段： FF40 0000h – FF4F FFFFh FF00 0000h – FF0F FFFFh	

18h	RW	FWH_DEC_EN2	按以上规律待细化	
1ch	RW	LPC IRQ (LPC_IRQ)	**When read:** 1 – INT; 0 – No INT. **When write:** Write 1 to clear corresponding IRQ bit; Write 0 has no effect. [31:20]：Reserved [17]：LPC HOST Error [15]：LPC RQ15 (Secondary IDE) [14]：LPC RQ14 (Primary IDE) [13]：LPC RQ13 (FPU) [12]： LPC IRQ12 (Mouse) [11:9]：LPC IRQ# (Undefined) [8]：LPC IRQ 11 ~ 8 (RTC) [7]：LPC IRQ7 (Parallel Port1) [6]：LPC IRQ6 (Floppy) [5]： LPC IRQ5 (Parallel Port2) [4]：LPC IRQ4 (UART)[3]： LPC IRQ3 (UART) [2]：Reserved [1]：LPC IRQ1 (Keyboard) [0]：LPC IRQ0 (8254 Timer)	0000000h
20h	RW	LPC IRQ Mask (LPC_IRQMSK)	Corresponding to “LPC_IRQ” 1 – Masked; 0 – Unmasked.	0000000h
LPC DMAC Related Registers				
24h	RW	DMA_CTRL	[31:2]：Reserved. [1]：Priority mode 1 – Rotating priority; 0 – Fixed priority (ch0>ch1>ch2>ch3) [0]：DMA Controller Enable 1 – Enable; 0 – Disable	0000000h
28h	RO	Channel Status	[7]：1 – Channel 3 done [6]：1 – Channel 2 done [5]：1 – Channel 1 done [4]：1 – Channel 0 done [3]：1 – Channel 3 busy/free [2]：1 – Channel 2 busy/free [1]：1 – Channel 1 busy/free [0]：1 – Channel 0 busy/free	0000000h
2ch	RW	ch0 Base Address	Channel 0 Base Address (32 bits)	0000000h
30h	RW	Ch1 Base Address	Channel 1 Base Address (32 bits)	0000000h
34h	RW	Ch2 Base Address	Channel 2 Base Address (32 bits)	0000000h
38h	RW	Ch3 Base Address	Channel 3 Base Address (32 bits)	0000000h
28h	RW	Ch0 Transfer Size	Channel 0 Transfer Size（16 bits, Count by BYTE）	0000000h

2Ch	RW	Ch1 Transfer Size	Channel 1 Transfer Size（16 bits, Count by BYTE）	0000000h
30h	RW	Ch2 Transfer Size	Channel 2 Transfer Size（16 bits, Count by BYTE）	0000000h
34h	RW	Ch3 Transfer Size	Channel 3 Transfer Size（16 bits, Count by BYTE）	0000000h
38h		Ch0_Transfer_Mode	[31:5]：Reserved [4:3]：Transfer mode 00 – Signal transfer mode; 01 – Block transfer mode 10 – demand transfer mode; 11 -- Reserved [2]：Transfer Direction 1 – DMA Write； 2 – DMA Read	
3ch		Ch1_Transfer_Mode		
40h		Ch2_Transfer_Mode		
44h		Ch3_Transfer_Mode		
48h	RW	All channel mask	[31:4]: Reserved [3]: 1 – Channel 3 masked; 0 – Channel 3 unmasked [2]: 1 – Channel 2 masked; 0 – Channel 2 unmasked [1]: 1 – Channel 1 masked; 0 – Channel 1 unmasked [0]: 1 – Channel 0 masked; 0 – Channel 0 unmasked	0000000h
4ch	WO	DMAC SW Reset	[1]：DMAC software reset 1 – Reset the DMA engine; 0 – No effect.	0000000h
50h	RO	LPC Current Status (LPC_CSTAT) ***Global Register**	[31:3]：Reserved [2]：LPC HOST Current status. 1 – Busy; 0 – Idle. [1:0]：Current Cycle Type 00: IO Cycle 01: Memory cycle 10: FW Cycle 11: DMA cycle	0000000h

3.9.3 GPIO

寄存器详细说明请参考寄存器手册。

GPIO 接口系统可见，实现 2 组，每组各 8 根输入/输出线。采用 DW_apb_gpio IP 实现 GPIO

接口功能，可以控制输出数据和 I/O pad 的方向，也可以使用寄存器读回 I/O pad 的数据。

配置 portA 和 portB，每个 port 配置 8 根线，利用软件控制实现输入输出数据和方向的控制。 为了使用方便，GPIO 要实现按位操作，而不是按字节操作，由于 AHB/APB 总线的特性，

最小粒度也只能做到按字节控制写操作。所以，增加对应 8 位数据的屏蔽码，和数据同拍写入

数据寄存器的高位，即[7:0]位为写入的 8 位数据，[15:8]位为 8 位数据对应的屏蔽码，GPIO 内

部根据 8 位数据和 8 位屏蔽码进行对应位的修改。

GPIO 控制器还可以作为中断控制器使用，允许外部输入中断信号，并且可以将外部的多根 中断信号或操作出一根中断线输出。

3.9.3.1 正常工作流程

采用软件控制方式，软件通过读写寄存器 gpio_swportx_dr 和 gpio_swportx_ddr 实现数据和 方向的控制。

- 通过写寄存器 gpio_swportx_ddr 控制外部 I/O pad 的方向，写“1”到 gpio_swportx_ddr 对应位代表选择输出方向；
- 写数据到 gpio_swportx_dr 寄存器对应位代表输出数据，并且数据寄存器的低 8 位是输 出数据，第[15:8]位是对应低 8 位的写使能，写使能位写“1”代表允许对应位写入， 写“0”代表不进行对应位的写；
- 写 “0” 到 gpio_swportx_ddr 对应位代表选择输入方向，软件通过读寄存器 gpio_ext_portx 获取输入数据。从输入信号 gpio_ext_portx 可以获得外部的输入数据，但是前提是取 决于 gpio_ext_portx 是被配置成输入还是输出：
  - 配置成输入——读取输入信号 gpio_ext_portx 的数据；
  - 配置成输出——读取 Portx 的数据寄存器。

  gpio_ext_portx 寄存器是只读的，驱动不能对该寄存器进行写操作。

3.9.3.2 中断流程

GPIO Port A 的 8 根线可以选择部分也可以选择全部配成中断信号，即驱动可以根据板级实 现情况，按位配置 Port A 的信号为 GPIO 信号或中断信号。GPIO 的中断功能实现了中断屏蔽寄 存器，中断状态寄存器，中断清除寄存器。

- 驱动配置寄存器 gpio_inttype_level，指定中断输入信号触发属性，是电平还是脉冲触 发，应该配置成电平触发；
- 驱动可以配置中断屏蔽寄存器，选择中断源的屏蔽和不屏蔽；

- 驱动配置寄存器 gpio_int_polarity，配置输入的每根中断信号是高电平触发还是低电平 触发；
- 驱动配置方向寄存器，指定中断线对应位的方向为输入；
- 驱动配置寄存器 gpio_inten，按位指定每一位是 GPIO 信号还是中断信号； 组合生成的一根中断线通过 PAB 桥以 MSI 包方式或 INTX 中断线方式提交到系统后，软件

查询 GPIO 的中断源寄存器并进行中断处理后，需要写 GPIO 的中断清除寄存器，清除中断。 虽然 GPIO 控制器提交了一个中断信号，但是可能是多个设备的中断源组合而成。

3.9.4 SPI

寄存器详细说明请参考寄存器手册。

ICH2 芯片所使用的 DW_apb_ssi IP 是一个可配置，可综合，可编程的全双工 master 或 slave 串行接口，配置成 master 模式，和外围设备之间的通信协议允许配置成一下三种形式：

- Motorola Serial Peripheral Interface（SPI）
- Texas Instruments Serial Protocol（SSP）
- National Semiconductor Microwire

如果实现成支持 SPI 协议，在寄存器 CTRLR0 的 FRF（frame format）域可以对选用支持的 串行协议进行编程配置。

3.9.4.1 时钟

当 SSI 控制器配置成 master 模式，位速率时钟 sclk_out 的最大频率是 ssi_clk 频率的一半， 即 ssi 控制器的核心工作时钟频率至少是 SPI 接口位数据传输时钟频率的 2 倍以上：

$F_{sclk\_out} = F_{ssi\_clk} / SCKDV$

SCKDV 的值可以通过可编程寄存器 BAUDR 进行修改，一般该值配成 2，如果配成 0，那 么就不产生 sclk_out。

如果波特率配置成 2，就可以实现控制器在时钟上沿发送命令、地址和数据，在时钟下沿 接收响应数据，对应的 slave 芯片在时钟上沿采样命令和数据，在时钟下沿发送响应数据。这样 双方的数据发送和接收都有半拍的延迟可用。

3.9.4.2 发送和接收 FIFO

SSI 控制器的发送和接收数据都要进入对应的 FIFO，发送和接收 FIFO 的队列深度可分别 配置，大小是 2 到 256 之间，宽度都固定成 16bits，因为串行协议的一次传输的数据帧（data frame）是 4 到 16bits。一个 FIFO 条目只能存放一个数据帧，对于一个 16bits 宽的 FIFO 条目，无法存放 2 个 8bits 的数据帧。如果在一个 FIFO 条目中写入了 8bits 的一个数据帧，那高 8 位也不会再使用。

当 SSI 控制器复位时，即 wb_rst_n 信号有效，或控制器关闭使能（SSI_EN=0）时，发送和 接收 FIFO 会被清除。

发送 FIFO 的写入条件是，wishbone 接口对数据寄存器（DR）的写操作。然后移位逻辑把 发送 FIFO 中的数据读到移位寄存器中。当发送 FIFO 中的有效条目数小于或等于阈值时，会产生一个 FIFO 空的中断请求（ssi_txe_intr）。阈值的设定是通过对寄存器 TXFTLR 的写入，决定 FIFO 条目数为多少时可以产生中断，达到提早通知系统发送 FIFO 即将读空的情况。如果对一 个已满的发送 FIFO 尝试写入数据，会产生一个上溢中断（ssi_txo_intr）。

接收 FIFO 的读出条件是，wishbone 接口对数据寄存器的读操作。移位控制逻辑把移位寄 存器中的数据写入接收 FIFO。当接收 FIFO 中的有效条目数大于或等于阈值加 1 的时候，会产 生一个 FIFO 满的中断（ssi_rxf_intr），阈值的设定是通过寄存器 RXFTLR 的写入。

3.9.4.3 中断

SSI 控制器的中断可以分别进行 mask，在 IP 生成的时候也可以选择中断有效的高低电平， 有如下几种中断信号：

1） 发送 FIFO 空中断（ssi_txe_intr）2） 发送 FIFO 上溢中断（ssi_txo_intr） 3） 接收 FIFO 满中断（ssi_rxf_intr）4） 接收 FIFO 上溢中断（ssi_rxo_intr） 5） 接收 FIFO 下溢中断（ssi_rxu_intr）

6） 多 master 冲突中断（ssi_mst_intr）：由于只有一个 master，不会使用该中断。7） 合成中断请求（ssi_intr）：上述几种中断请求的或。

3.9.4.4 传输模式

通过寄存器 CTRLR0 的 TMOD 位的配置，可以选择 SSI 控制器进行如下四种模式的传输：

1） 发送和接收

TMOD=2‘b0，发送和接收逻辑都有效，发送数据从发送 FIFO 中取出送到 txd 输出接口，并从 rxd 接口接收输入数据写入接收移位寄存器，然后写入接收 FIFO。数据的接收需要发送 FIFO 不断的有数据输出到串行总线，以保证持续的串行时钟的产生。

2） 只发送

TMOD=2‘b01，只有发送逻辑有效，只往串行总线上上发送数据，不会从总线上接收数 据。

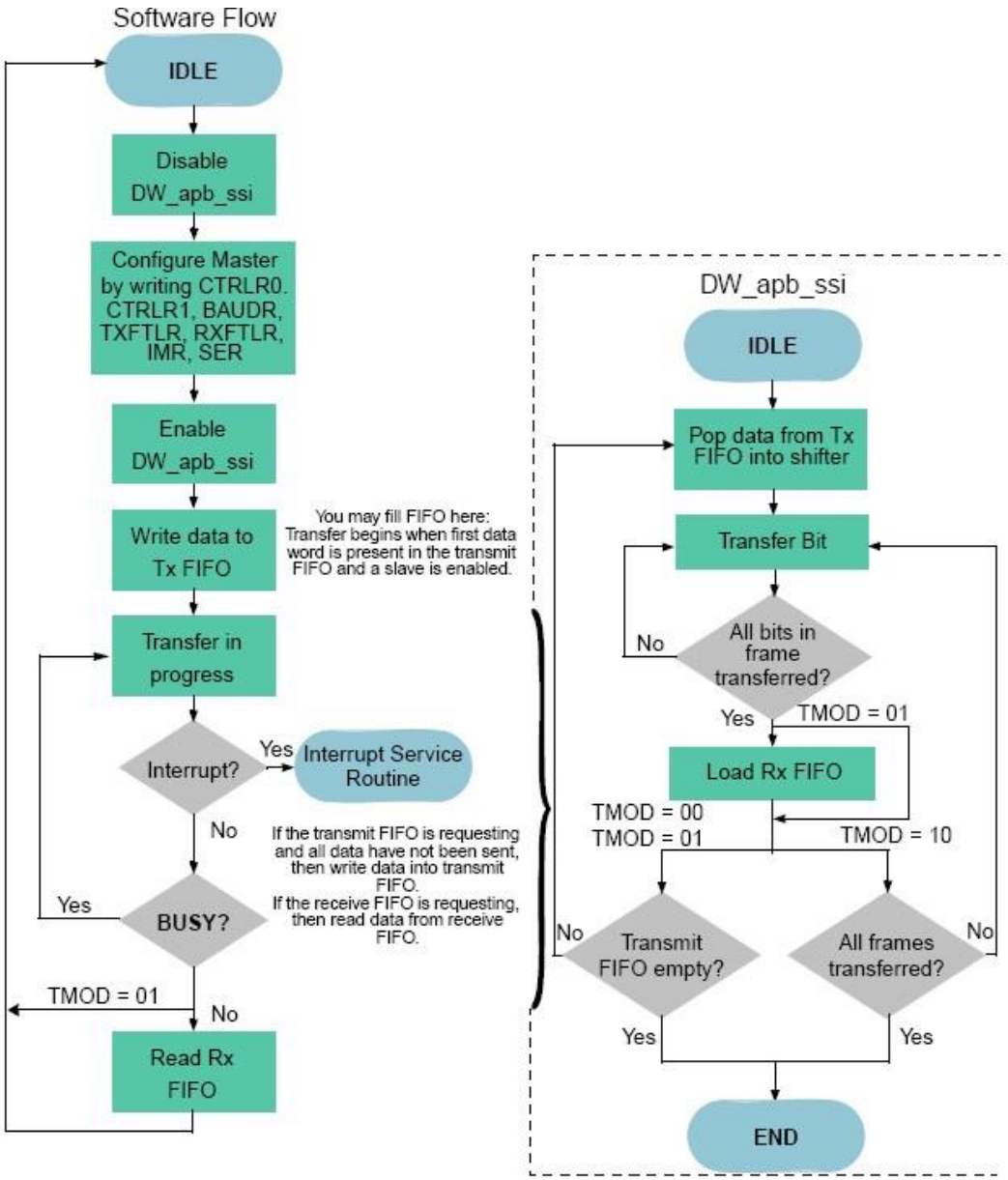
3） 只接收

TMOD=2‘b10，只有接收逻辑有效。 4）

EEPROM 读

TMOD=2‘b11，发送数据用于传输命令码和地址到 EEPROM 设备，一般需要 3 个数据帧（8-bit 操作码，8-bit 高位地址，8-bit 低位地址）。在操作码和地址传输的过程中， 控制器不会从串行总线上接收数据。当发送 FIFO 中的条目发送完毕，接收数据才开始 被采样，采样的数据帧数是 NDF+1，见寄存器 CTRLR1。

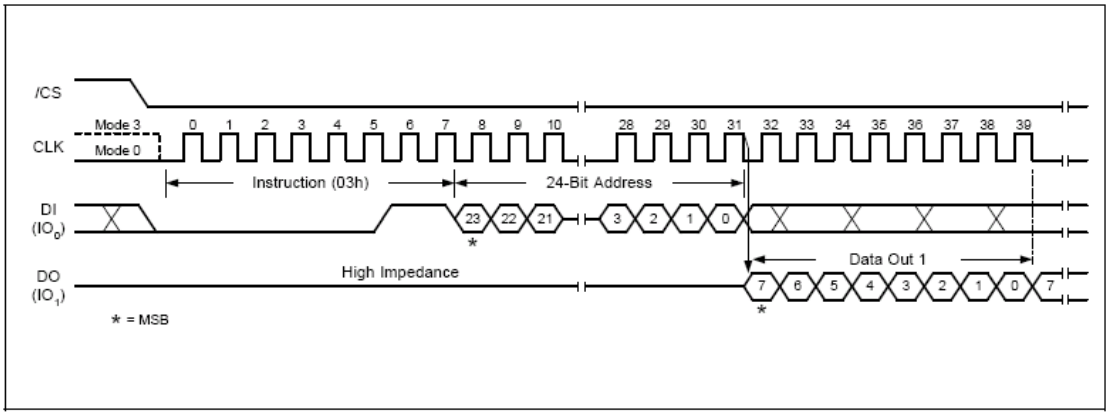
3.9.4.5 操作模式


Software Flow
IDLE
Disable DW_apb_ssi
Configure Master by writing CTRLR0, CTRLR1, BAUDR, TXFTLR, RXFTLR, IMR, SER
Enable DW_apb_ssi
Write data to Tx FIFO
You may fill FIFO here: Transfer begins when first data word is present in the transmit FIFO and a slave is enabled.
Transfer in progress
Interrupt?
Yes
Interrupt Service Routine
No
If the transmit FIFO is requesting and all data have not been sent, then write data into transmit FIFO.
If the receive FIFO is requesting, then read data from receive FIFO.
BUSY?
Yes
TMOD = 01
No
Read Rx FIFO
DW_apb_ssi
IDLE
Pop data from Tx FIFO into shifter
Transfer Bit
No
All bits in frame transferred?
Yes
TMOD = 01
Load Rx FIFO
TMOD = 00
TMOD = 01
TMOD = 10
No
Transmit FIFO empty?
All frames transferred?
No
Yes
Yes
END

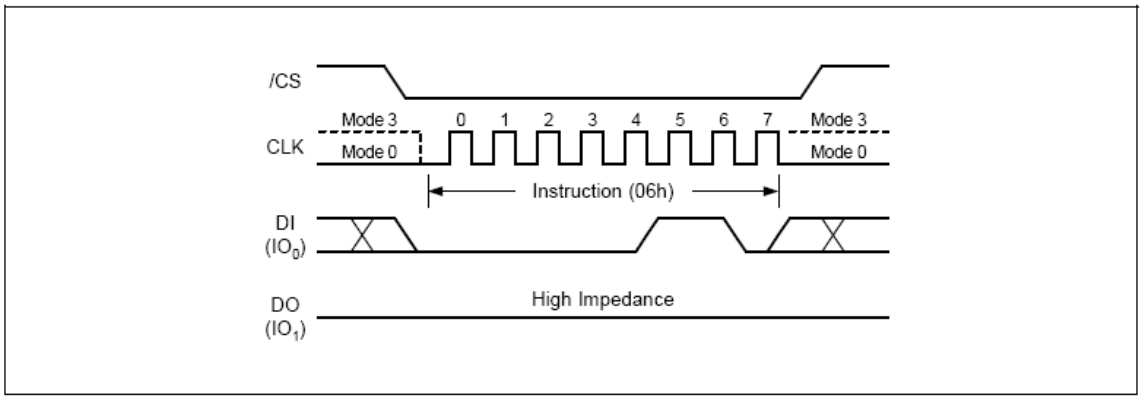
3.9.4.6 工作流程

Flash 访问命令

不同的 flash 芯片接受的访问命令和数据传输方式可能有所不同，但是基本的命令 Read，Write Enable，Page Program，Erase 都会支持，只是不同的芯片操作码定义和实现方式略有不同。以 winbond 的一款 flash 芯片为例，几种命令的操作码和传输格式如下：

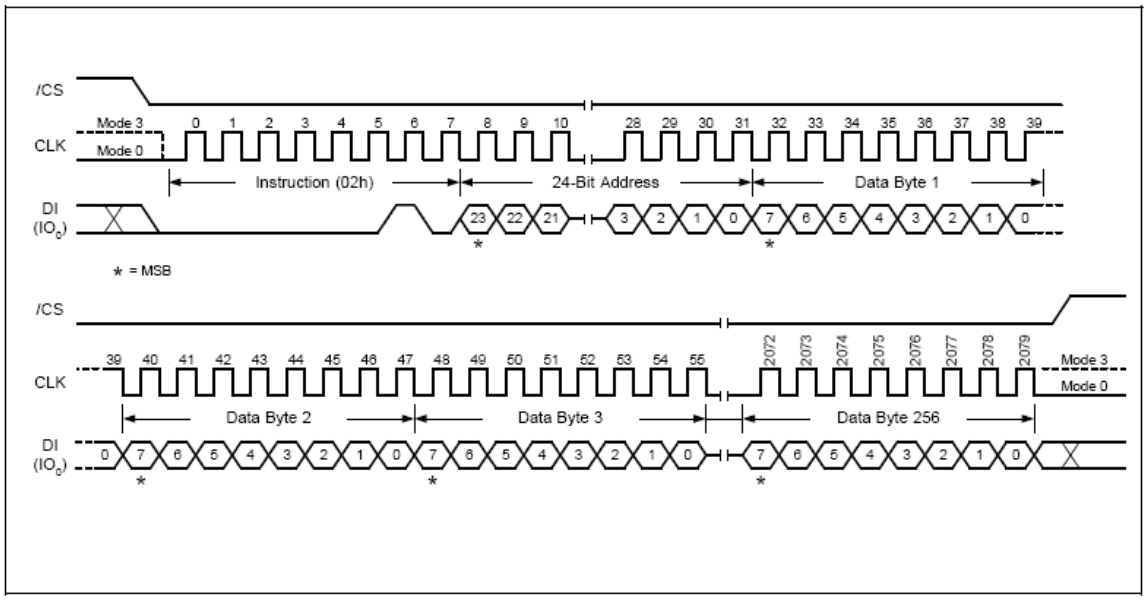


读数据命令序列（Read）

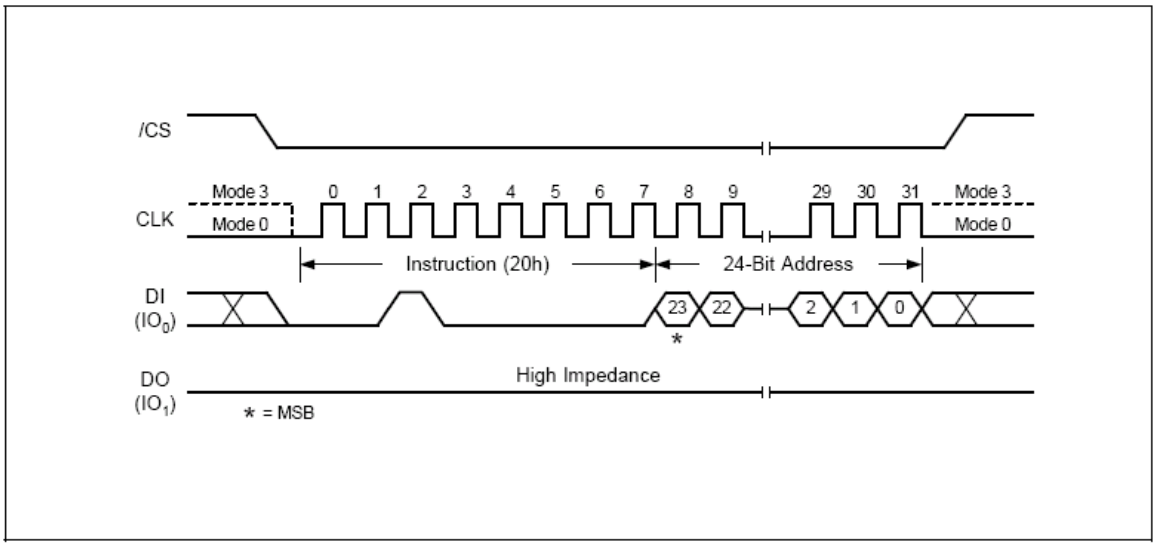


写使能命令序列（Write Enable）

在 Page Program 命令和 Erase 命令执行前都需要执行写使能命令。



页编程命令序列（Page Program） 支持一次进行 1~256 字节的数据编程。

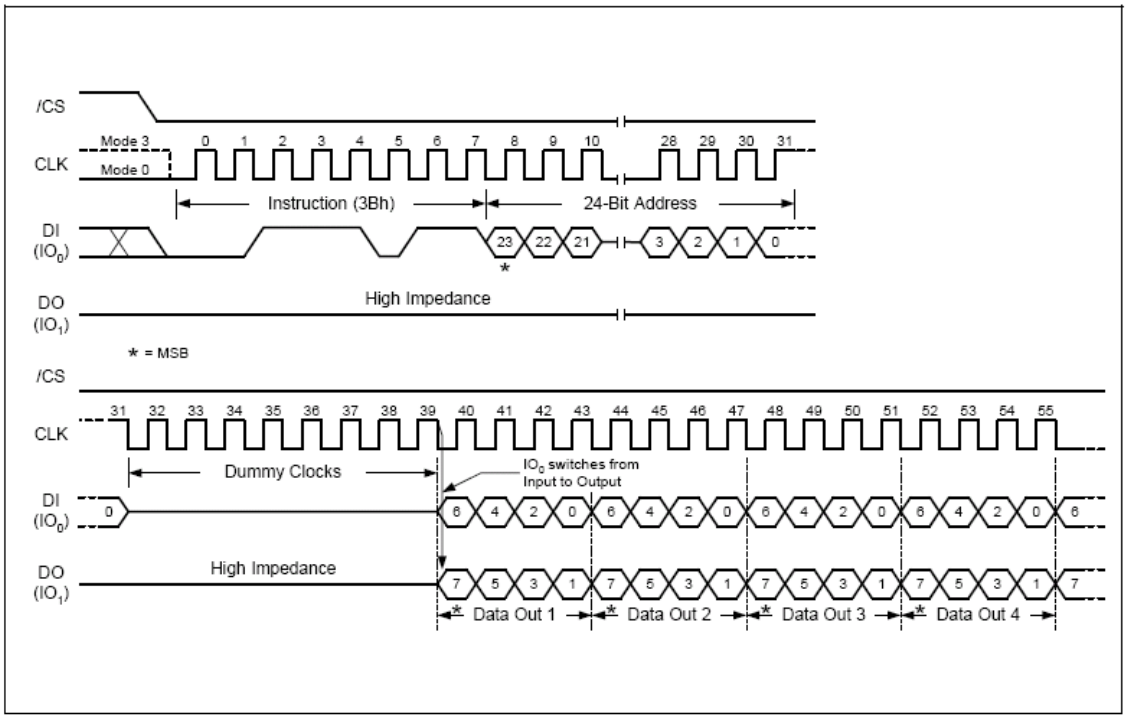


扇区擦除命令序列（Sector Erase） 擦除命令可以有扇区擦除，以可以有块擦除，以及芯片擦除，看不同的 flash 芯片类型和容量确定擦除命令，以及扇区、块定义的大小，擦除命令会把选择的区域全写成 1.在编程命令前需要进行 flash 擦除，否则写入的数据是和原来 flash 芯片中的数据进行与操作后才会写入，为了保证写入数据的正确性，先进行擦除成全 1 后再编程。

再以一个 xilinx 的 flash 模型为例，Read 命令操作码和序列同 winbond 芯片一致，但是写操作有不同，不需要执行写使能，首先执行命令 Buffer 1 Write，把要编程的数据写入 flash 芯片的一个临时缓冲，然后执行命令 Buffer 1 To Main Memory Page Prog With Erase，把临时缓冲中

的数据真正写入存储器，并且写入前自动进行擦除操作。

从上面的两个不同的 spi flash 模型可以看到，尽管不同芯片的编程命令有所不同，只要 SSI 控制器的驱动改变一下就可以支持，即驱动和芯片类型紧密相关。但是，对于有些 flash 芯片的 dual 模式，甚至是 quad 模式，如下图所示的一款芯片支持的 dual read 模式：



上述的非 single 模式需要复用通路，而 DW_apb_ssi 这个 SSI 控制器 IP 并不支持，MOSI 和 MISO 信号都是单向的，所以该控制器只能支持单通道读写命令。

3.9.4.7 编程流程

按照 SSI 控制器操作模式的描述，启动编程过程的软件寄存器读写过程如下：

序号	操作
1	disable SSI 控制器，写‘h8 地址 SSIENR 寄存器值为 0x0
2	配置 TMOD 为 transmit only 模式，写‘h0 地址 CTRLR0 寄存器值为 0x1c7，配置 SCPH 和 SCPOL 也为 1，有的芯片只支持 MODE3，即前面两个值都为–1‖
3	配置波特率，写‘h14 地址 BAUD 寄存器值为 2
4	配置发送 FIFO 阈值，写‘h18 地址 TXFTLR 寄存器值为 0x4

5	配置接收 FIFO 阈值，写‘h1c 地址 RXFTLR 寄存器值为 0x4
6	使能 slave 设备，写‘h10 地址 SER 寄存器值为 0x1
7	enable SSI 控制器，写‘h8 地址 SSIENR 寄存器值为 0x1
8	发送写使能命令，写‘h60 地址 DR 寄存器值为 0x6
9	发送擦除命令操作码，写‘h60 地址 DR 寄存器值为 0x20
10	发送擦除命令高段地址[23:16]，写‘h64 地址 DR 寄存器
11	发送擦除命令中段地址[15:8]，写‘h68 地址 DR 寄存器
12	发送擦除命令低段地址[7:0]，写‘h6c 地址 DR 寄存器
等待擦除完毕	
13	发送写使能命令，写‘h60 地址 DR 寄存器值为 0x6
14	发送编程命令操作码，写‘h60 地址 DR 寄存器值为 0x2
15	发送编程命令高段地址[23:16]，写‘h64 地址 DR 寄存器
16	发送编程命令中段地址[15:8]，写‘h68 地址 DR 寄存器
17	发送编程命令低段地址[7:0]，写‘h6c 地址 DR 寄存器
……	从‘h70 地址寄存器开始到‘hec，再从‘h60 到‘hec 地址寄存器，依次写入希望编程的数据
等待编程数据写入结束	

编程命令需要配置成 transmit only 传输模式，而不能配置成 transmit & receive 传输模式，否则在发送写命令和数据期间，接收 FIFO 会因为持续产生的串行时钟从串行输入数据总线上 取得数据写入 FIFO，造成非期望接收数据的写入。

3.9.4.8 EEPROM 读流程

序号	操作
1	disable SSI 控制器，写‘h8 地址 SSIENR 寄存器值为 0x0
2	配置 TMOD 为 eeprom read 模式，写‘h0 地址 CTRLR0 寄存器值为 0x3c7，配置 SCPH 和 SCPOL 也为 1，有的芯片只支持 MODE3，即前面两个值都为–1‖

3	配置波特率，写‘h14 地址 BAUD 寄存器值为 2
4	配置发送 FIFO 阈值，写‘h18 地址 TXFTLR 寄存器值为 0x4
5	配置接收 FIFO 阈值，写‘h1c 地址 RXFTLR 寄存器值为 0x4
6	配置接收数据帧数目 NDF，写‘h4 地址 CTRLR1 寄存器值为 ndf，接收的数据帧
	的数目从发送完 8-bit 操作码和 16-bit 地址后，间隔一个 8bit 串行数据收集时间后就开始增加，但是如果连接的芯片不是 16 地址，而是 24 位地址，那么接收数据帧数目的设定值 ndf 就不应该是期望接收数目-1，而应该恰好是期望接收数目的值，因为计数器相当于早 8 个串行时钟周期开始计数。
7	配置接收采样延迟，写‘hf0 地址 RSD 寄存器值为 0x0 或0x10，如果随命令后发送 16-bit 地址，则写入 0x10，如果发送 24-bit 地址，则写入 0x0，原因见下面的说明。
8	使能 slave 设备，写‘h10 地址 SER 寄存器值为 0x1
9	enable SSI 控制器，写‘h8 地址 SSIENR 寄存器值为 0x1
10	发送读数据命令操作码，写‘h60 地址 DR 寄存器值为 0x3
11	发送读数据命令高段地址[23:16]，写‘h64 地址 DR 寄存器
12	发送读数据命令中段地址[15:8]，写‘h68 地址 DR 寄存器
13	发送读数据命令低段地址[7:0]，写‘h6c 地址 DR 寄存器
等待读响应数据返回	
14	读地址‘h34 的原生中断状态寄存器的[4]位，判断是否为–1‖，如果为 1，则代表接收 FIFO 已经收到接收阈值数目以上的接收数据，可以启动系统总线读操作，连续的读‘h60~‘hec 地址数据寄存器的接收数据，直到该位始终保持为–0‖，代表已经读完 ndf 或 ndf-1 个数据帧。

EEprom read 模式因为确定传输的前 3 个字节分别是 opcode，addr[15:8]，addr[7:0]，所以 tmod==2‘b11 加入了条件 scph 和 spi1_control（发送缓冲非空）的判断，即确保控制器把 3 个字节的数据放到 SPI 总线上才会使能 FIFO 接收，但是还需要 8 个 ssi_clk 才能把第一个字节的输入数据存下来，所以还是需要加入 rx_sample_dl（y 设置成 16，因为波特率为 2，即控制器的 ssi_clk

是输出给 flash 的时钟频率的 2 倍）。NDF 的值同样也需要加 1. 前面这样设置的前提是 flash 只 支持 16 位地址，如果是支持 24 位地址，那么 eeprom read 的命令依然是 1 个字节的 opcode+3 个字节的地址，这样就不需要设置 rx_sample_dly，或写成 0。

其他的非 EEprom read 模式，由于输出数据的字节不定，没有办法进行延迟判断，所以都是在 发出第一个字节的数据后就置接收有效。所以每次需要根据发送的控制字节数（操作码和地址） 的不同修改 rx_sample_dly 的值（设置成 64，在 eeprom read 的基础上需要增加 3 字节的地址）。

3.9.4.9 正常工作读写流程

正常工作的写操作流程同上面的编程流程，而读操作，如果不通过 EEPROM 读模式，也可 以配置成 transmit & receive 模式，执行的过程类似 EEPROM 读。

同样的读操作，SSI 控制器的工作原理在 transmit & receive 模式和 EEPROM 读模式下是不 同的，EEPROM 读模式是按照 ndf 的设定值进行规定数据数目的接收。

而 transmit & receive 模式接收数据的数目是按照发送 FIFO 的有效数目来确定，只有发送 FIFO 持续的有数据发到串行总线上，才会有持续的 sclk_out，以此来控制接收数据的帧数。即 发送完操作码和地址后，还需要持续的发送期望接收帧数目的数据帧到串行总线上。接收 FIFO 写入的条目数和发送 FIFO 的发送条目数相同，即控制器真正收到的数据帧数=期望数据帧数+1个操作码+3 个地址码，即期望数+4.非期望数据的忽略通过采样延迟配置成 0x40 即可把前面的 64 拍 ssi_clk（4*8=32 个 sclk_out 时钟周期）时钟周期的等待接收数据时间去掉。 考虑写操作的执行时间比较长，而且读操作和写操作的传输模式也不同，需要分别配置，所以 尽量避免读写交叉，减少切换时间。

3.9.5 I2C

3.9.5.1 功能介绍

I2C 是一个两线接口，包括串行数据总线 SDA 和串行时钟 SCL。通过 I2C，各设备之间以 及设备与系统的其它部分之间可以互相通信。I2C 为系统和电源管理相关的任务提供一条控制 总线。一个系统利用 I2C 可以和多个设备互传信息，而不需要使用独立的控制线路。I2C 总线 涉及三类设备：

- ✧ 主设备，用来发布命令，产生时钟和终止发送的设备；
- ✧ 从设备，接收或响应命令的设备；
- ✧ 主机，是一种专用的主设备，必须具有主-从机功能，一个系统里只有一个主机。

I2C 接口控制器采用 DW_apb_i2c IP 实现。

3.9.5.2 工作流程

1）写-0Ⅱ到 DW_apb_i2c 的 IC_ENABLE 寄存器，关闭控制器 IP 使能，可以进行相关模式配置；

2)写 IC_CON 寄存器，可以设置最大的速度模式，7bit 还是 10bit 地址模式，IC_SLAVE_DISABLE 位设置成-1Ⅱ，MASTER_MODE 位设置成-1Ⅱ；

3）写 IC_TAR 寄存器，设置 I2C 目标设备地址；

4）写-1Ⅱ到 IC_ENABLE 寄存器，使能控制器 IP；

5）写 IC_DATA_CMD 寄存器，写入传输方向（读写类型）和写数据： i）如果是写请求，IC_DATA_CMD[8]位写-0Ⅱ，IC_DATA_CMD[7:0]为写数据； ii）如果是读请求，IC_DATA_CMD[8]位写-1Ⅱ。

6）如果传输的是写请求：当 I2C 控制器把写请求和数据从发送 FIFO 都发到总线上，控制器就 产生一个 STOP 中断，表示当前传输完成，即置寄存器 IC_RAW_INTR_STAT[9]位为-1Ⅱ，并通 知系统。驱动通过读中断源寄存器获取中断类型并处理，然后读寄存器 IC_CLR_STOP_DET 即 可清除该中断。

7）如果传输的是读请求：读请求命令发到 I2C 总线，控制器按照 I2C 协议接收总线返回的读响 应数据，并写入接收 FIFO，控制器置寄存器 IC_RAW_INTR_STAT[2]位为-1Ⅱ，代表 RX_FULL， 并发出中断通知系统。驱动读取中断源寄存器，确定中断类型，然后驱动读取 IC_DATA_CMD 寄存器获得读响应数据。RX_FULL 中断会被硬件自动清除当数据被逐个读出接收 FIFO 后。

说明：前 3 步是可选的，如果 I2C 控制器都使用复位默认值，而不需要进行配置修改，就直接 跳过这几步。