ICH1 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

《ICH1 软件接口手册》主要描述了国产 IO 桥片的相关结构、片内设备等内容，并详细介绍了片内设备。

文档修订

文档更新记录	文档名	ICH1 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/

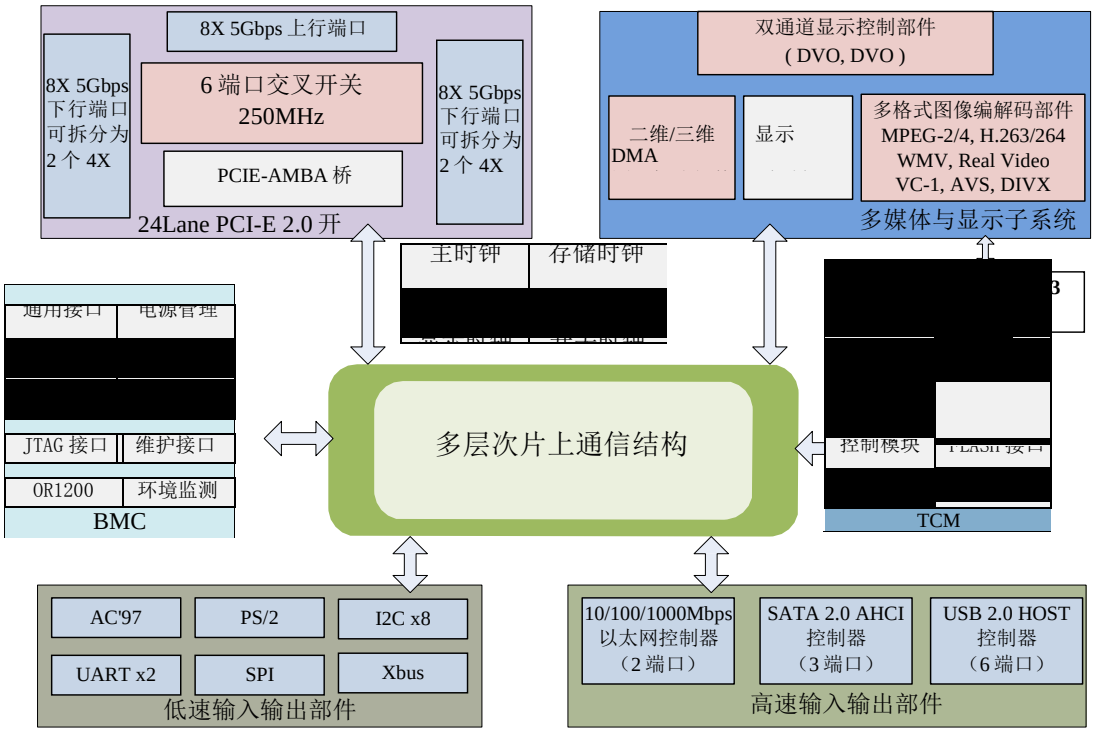
目　录

1 概述

本手册定义国产 IO 芯片 SWICH 同操作系统间的协议要求，用于国产 IO 芯片系统软件开 发。

SWICH 是一个 IO 控制型芯片，由多层次片上总线将 IO 控制部件互连起来。SWICH 的基 本结构如图 1-1 所示。

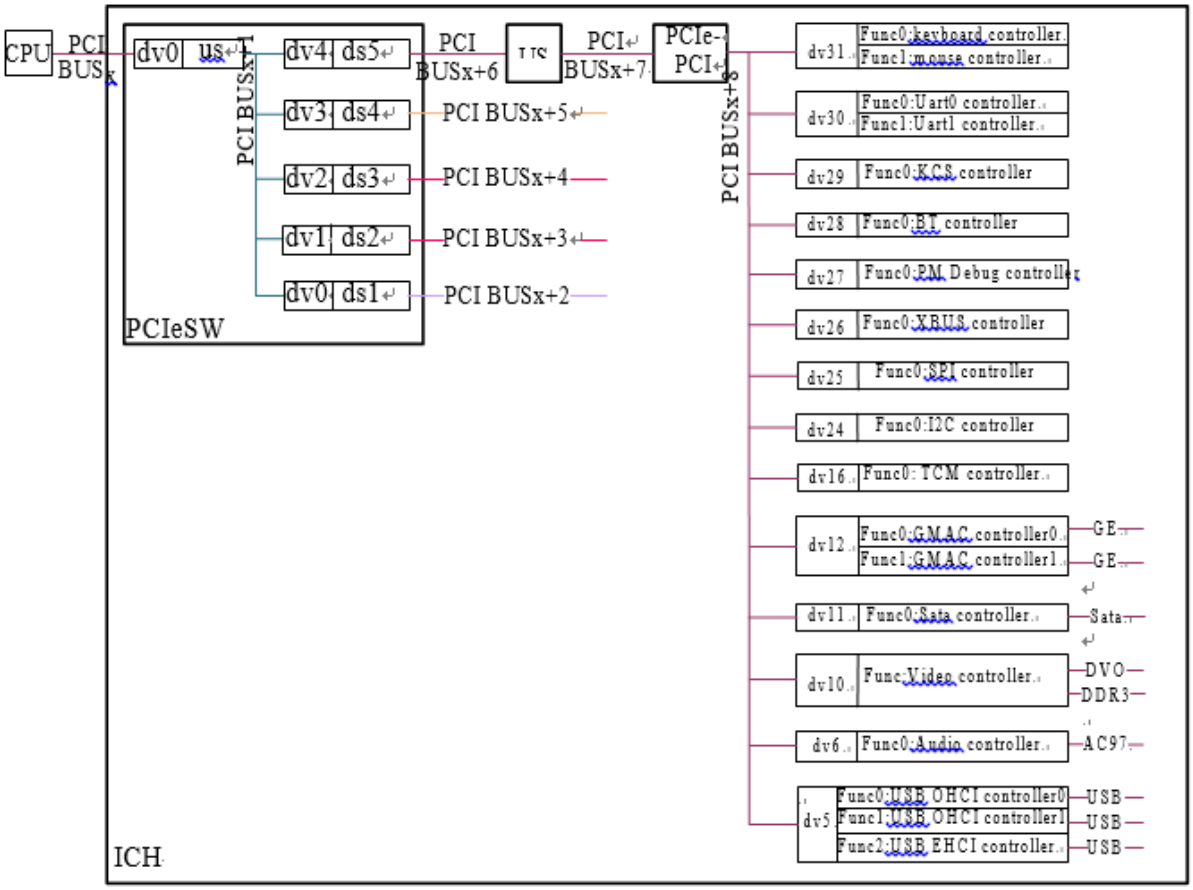
图 1-1 套片结构



2 套片空间编址

2.1 拓扑结构

图 2-1　套片拓扑



2.2 资源划分

表 2-1　套片 PCI 资源划分

Bus:Device:Function	功能描述
Busx:Device0:Function0	PCIeSW 的上游端口
Busx+1:Device4:Function0	PCIeSW 的下游端口 5
Busx+1:Device0:Function0	PCIeSW 的下游端口 1
Busx+1:Device1:Function0	PCIeSW 的下游端口 2
Busx+1:Device2:Function0	PCIeSW 的下游端口 3
Busx+1:Device3:Function0	PCIeSW 的下游端口 4
Busx+2: DeviceY:FunctionZ	PCIeSW 的下游端口 1 挂接的总线
Busx+3: DeviceY:FunctionZ	PCIeSW 的下游端口 2 挂接的总线
Busx+4: DeviceY:FunctionZ	PCIeSW 的下游端口 3 挂接的总线
Busx+5: DeviceY:FunctionZ	PCIeSW 的下游端口 4 挂接的总线
Busx+6:Device0:Function0	PCIe US IP
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:Device30:Function1	Uart1 控制器
Busx+8:Device29:Function0	KCS 控制器
Busx+8:Device28:Function0	BT 控制器
Busx+8:Device27:Function0	PMDebug 控制器
Busx+8:Device26:Function0	XBUS 控制器
Busx+8:Device25:Function0	SPI 控制器
Busx+8:Device24:Function0	I2C 控制器
Busx+8:Device16:Function0	TCM 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	视频控制器
Busx+8:Device6:Function0	AC97 音频控制器
Busx+8:Device5:Function0	USB OHCI 控制器 0
Busx+8:Device5:Function1	USB OHCI 控制器 1
Busx+8:Device5:Function2	USB EHCI 控制器

2.3 套片 PCIe 空间定义

- 支持系统对设备的 64 位地址 IO 访问，粒度为 1B、2B、4B；不支持带空洞的 IO 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。
- 套片内部实现的 PCI 设备占用系统 2GB+16MB 可预取 MEM 空间。
- BMC 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　套片 PCIe 64 位 BAR 地址空间定义

功能	空间大小及类型
BMC 的 PS2 键盘	8KB PRE-MEM
BMC 的 PS2 鼠标	8KB PRE-MEM
BMC 的 Uart0	8KB PRE-MEM
BMC 的 Uart1	8KB PRE-MEM
BMC 的 KCS	8KB PRE-MEM
BMC 的 BT	8KB PRE-MEM
BMC 的 PMDebug	8KB PRE-MEM
BMC 的 XBus	1MB PRE-MEM
BMC 的 SPI	8KB PRE-MEM
I2C 控制器	16KB PRE-MEM，约定： Address[13]=0 是 I2C0 Address[13]=1 是 I2C1
TCM 控制器	BAR0：1MB 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，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 EHCI 控制器	512KB PRE-MEM
USB OHCI 控制器 0	256KB PRE-MEM
USB OHCI 控制器 1	256KB PRE-MEM
PCIe-PCI 桥	BAR0：512KB PRE-MEM，空间为 PCIe-PCI 桥的寄存器

3 套片内设备

3.1 PCIe Switch

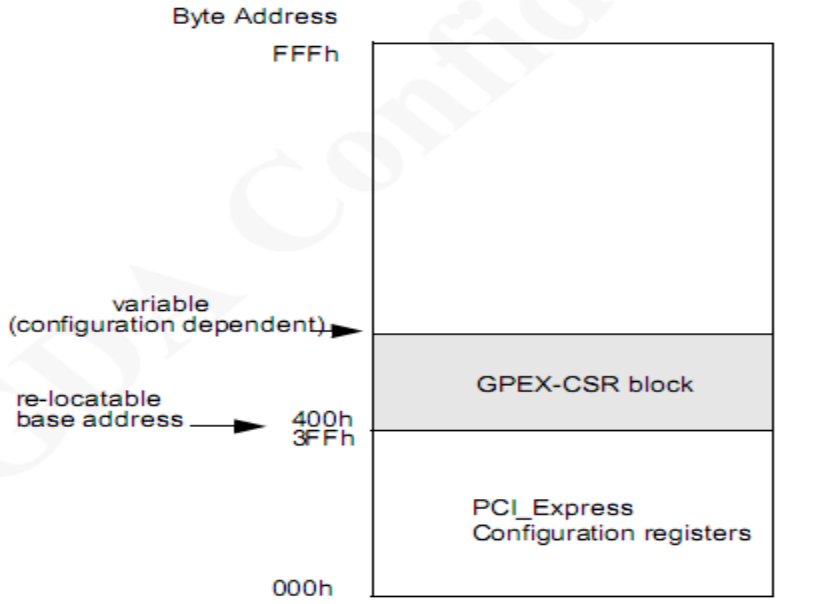
套片中的 PCIE switch 实现的功能为转发 Host 给下游设备的 IO 和配置请求以及 DMA 响应， 和下游设备给上游设备的 DMA 请求和配置及 IO 响应。同时转发 PCIE 协议规定的各类中断包 及电源管理等消息包。

3.1.1 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套片寄存器手册。

Figure 2 : PCI_Express Config Address Space



3.1.1.1 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。

3.1.1.2 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。

3.1.2 软件对 PCIE Switch 的相关操作

以下为软件对 PCIE 系统的主要操作。

3.1.2.1 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。

3.1.2.2 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）。

3.1.2.3 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。

3.1.2.4 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。

3.2 PCIe-AMBA 桥

PCIe-AMBA 桥是实现套片 PCI 拓扑，实现套片内部所有设备 PCI 功能框架，实现系统对 套片内部所有设备的初始化、枚举和驱动，实现所有设备中断，以及实现套片内部 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) 实现系统对套片设备的 IO 访问

系统对设备 IO 访问请求基于系统 IO 空间，该请求被 PCIe-AMBA 桥认领， PCIe-AMBA 桥将请求转换成套片 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) 实现套片设备对系统主存的 DMA 访问 套片各设备对系统主存的访问请求基于系统空间，这些请求在套片 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) 实现套片设备对系统的中断

PCIe-AMBA 桥中实现设备中断控制器，用于套片各个设备对系统中断请求的集中提交和处理。

语义上，所有挂接在套片 AMBA 总线上的需要由系统驱动和使用的 IO 设备都映射为套片 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套片 AMBA 总线上的 IO 设备。

3.2.1 桥的宏观特性

- 实现套片设备的 PCIe 拓扑结构
- 实现和 PCIe Switch 互连的 PCIe2.0 8X PIPE 直连通道
- 实现高效低延时的 PCIe 总线和 AMBA 总线间的协议转换
- 实现系统对套片设备的配置访问
- 实现系统对套片设备的 64 位地址 MIO 访问，支持系统对显示控制器 2GB 显存的 IO 访问， 套片设备的 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
- 实现套片设备多个相同 ID 的 AXI 总线读写事务的保序
- 实现套片设备对系统的 MSI 和 INTx 中断请求，实现可编程的中断优先级

- 实现套片设备对系统主存的 32 位地址 DMA 访问
- 实现套片 Legecy IO
- 支持系统 Max_Payload_Size: 128Bytes/256Bytes
- 支持系统 Max_Read_Request_Size: 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

3.2.2 桥的 IO 操作流程

IO 操作指由系统发起的对外部设备的 IO 读写请求，对于 PCI 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。系统对套片设备的 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 总线并由套片接收；
2) 套片 PCIeSW 的上游端口收到这个请求，经过 TYPE1 PCIe 配置空间的基限 BAR 判断该 TLP 请求是对 PCIeSW 的下游内部端口 PCIe-AMBA 桥的合法 IO/MEM 请求，并将该请求 转发给内部端口 PCIe-AMBA 桥；
3) 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；
4) 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，PCIe-AMBA 桥发给 PCIeSW，PCIeSW 转 发给 CPU；
5) 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；
6) 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；
7) 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；
8) 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，PCIe-AMBA 桥发给 PCIeSW，PCIeSW 转发给服务器 CPU。

3.2.3 桥的 DMA 操作流程

DMA 访问指由套片内设备发起的对系统主存的 DMA 读写请求，套片内设备 DMA 请求流 程是：

1) 设备根据需求（CPU 对 PCIe 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（没有真正完成）， 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（AHB 总线上的设备发起的请求对于 PCIe 总线来说，都是 HMX 桥发起的请求）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 包发到套片 PCIeSW 上游端口，套片 PCIeSW 上游端口通过 PCIe 总线发到服务器 CPU 主存；

5) 主存返回读响应，由 PCIe RC 经过 PCIe MEM 读完成事务发给套片 PCIeSW，PCIeSW 将 事务发给 PCIe-AMBA 桥接口；

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。

3.2.4 桥的配置寄存器定义

寄存器详细说明请参考套片寄存器手册。

3.2.5 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，转发给 PCIeSW 上游端口，对于 INTx，PCIeSW 上游端口进行规约，然后将中断消息通过 PCIe 总 线发到服务器 CPU；

5) 服务器 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，清除套片 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。

3.3 AC97 控制器

3. 3. 1 AC97 配置

3.3.1.1 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 控制器初始化按下表进行：

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

3.3.1.2 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 操作如下：

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 操作如下：

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 初始化如下：

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	

3.3.2 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

3.3.2.1 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

3.3.2.2 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。

为每个一个通道更新 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。

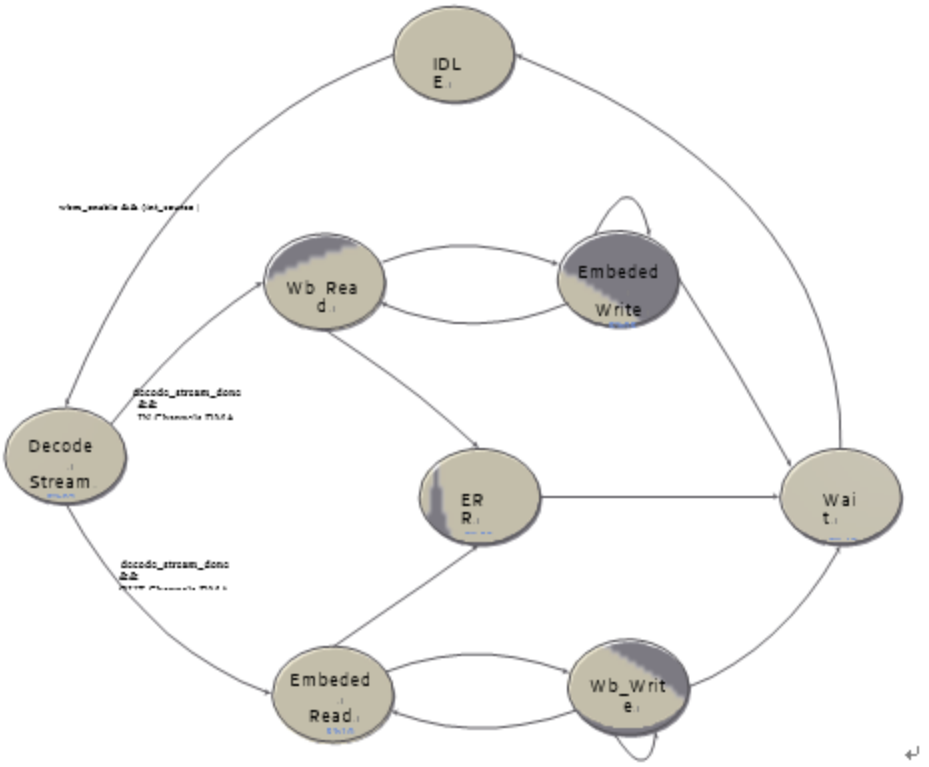
举例来说：发送给前置左右通道的单声道的数据流将置 INT_DMA_MASK 和 INT_DMA_TRIGGER 的 [1:0] 位为 1，而立体声会设置 INT_DMA_MASK 的 [1:0] 位和 INT_DMA_TRIGGER[1]。

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 GMAC 控制器

3.4.1 可编程寄存器

寄存器详细说明请参考套片寄存器手册。

3.4.2 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。

3.4.2.1 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：

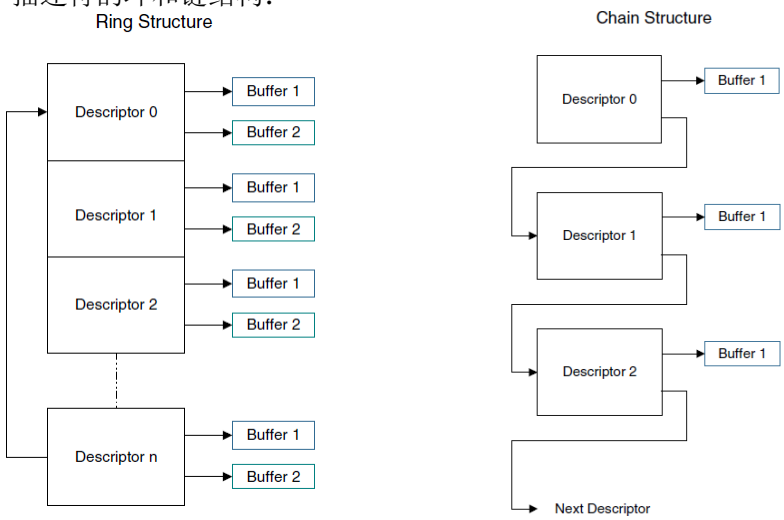
- 控制状态寄存器，Control and Status registers (CSR)
- 描述符和数据缓冲，Descriptor lists and data buffers

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：



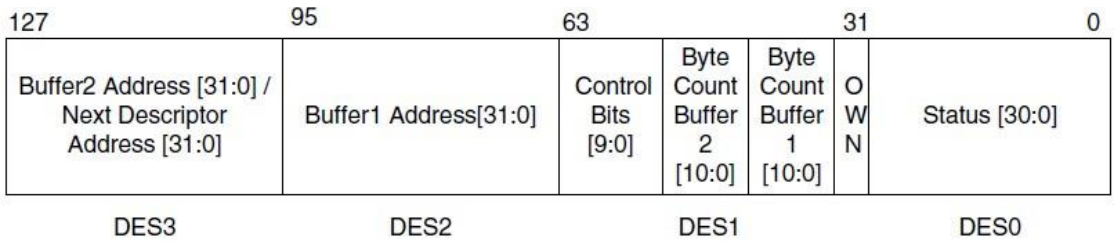
描述符列表驻留在 Host 物理存储地址空间中。每个描述符可以指向最大的 2 个缓冲。这可 以指向 2 个物理地址的 2 个缓冲，而不只是存储空间中的连续缓冲。一个主机物理地址空间中的数据缓冲有一个完整的帧或部分帧组成，但是不能超过一个帧。缓冲中只包含数据，缓冲的状态保存在描述符中。指向帧的数据链跨越多个数据缓冲，但是一个描述符不能跨越多个缓冲。 DMA 检测到 EOF 时，将忽略下一个帧缓冲。可以使用也可以不使用数据链。

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

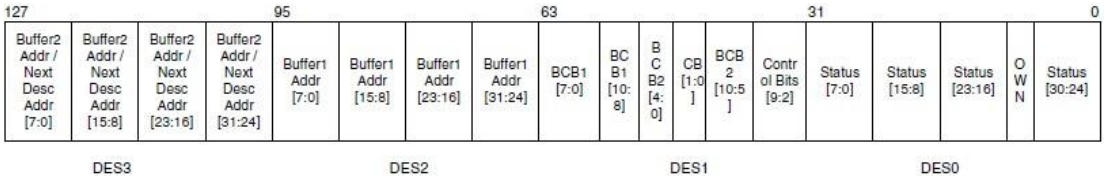
Little-Endian/Big-Endian ， 描 述 符 大 小 端 模 式 Reverse-Endian/Same-Endian ， 数 据 宽 度 有 32/64/128-bit。描述符的格式可以使用普通格式(16B)或可变/增强格式(32B)。

下面是 128 位数据总线的的描述符定义：

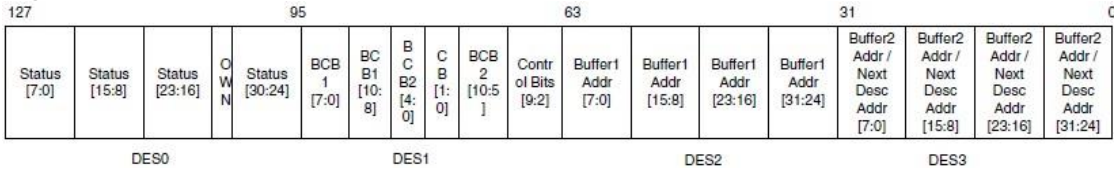
Little-Endian、Same-Endian 描述符：



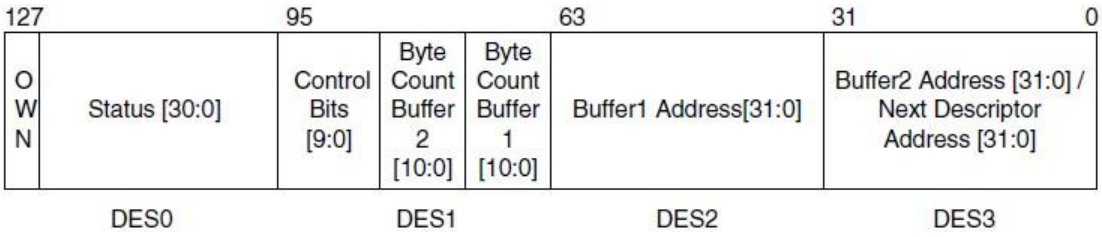
Little-Endian、Reverse-Endian 描述符：



Big-Endian、Same-Endian 描述符：



Big-Endian、Reverse-Endian 描述符：



3.4.2.2 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。

3.4.2.3 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。

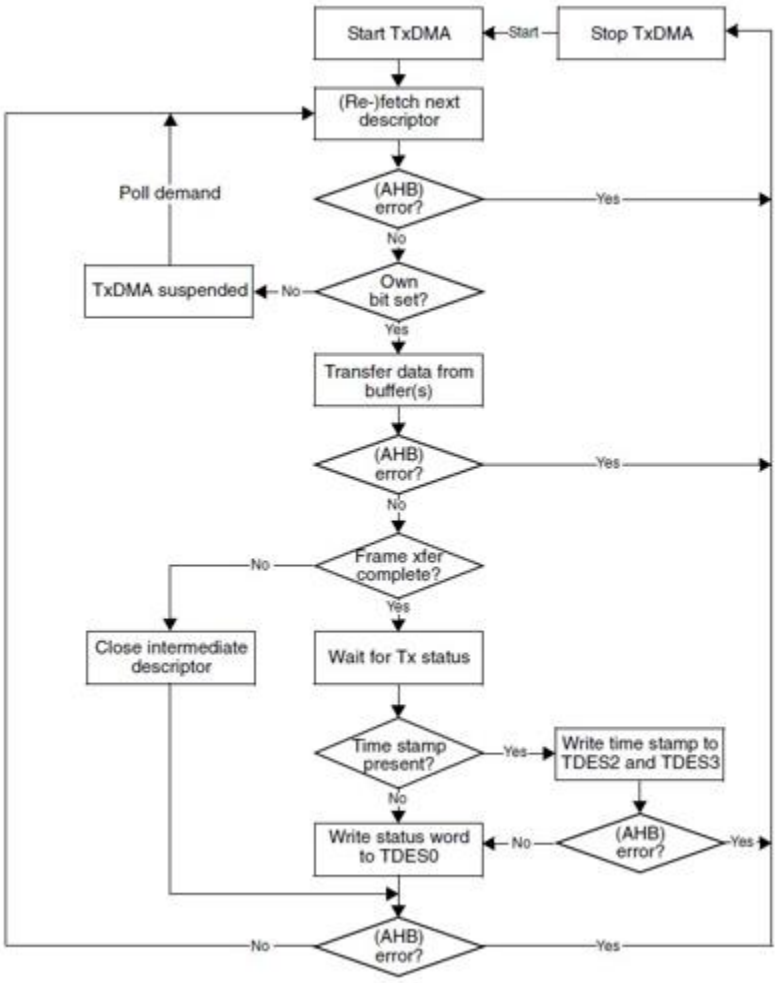
6） 如果以太网帧存储在数据缓冲在过个描述符中，DMA 关闭中间的描述符并取下一个描述符。重复第 3、4、5 步知道 EOF 数据传输给 MTL。

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。

具体流程图如下所示：



3.4.2.4 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。

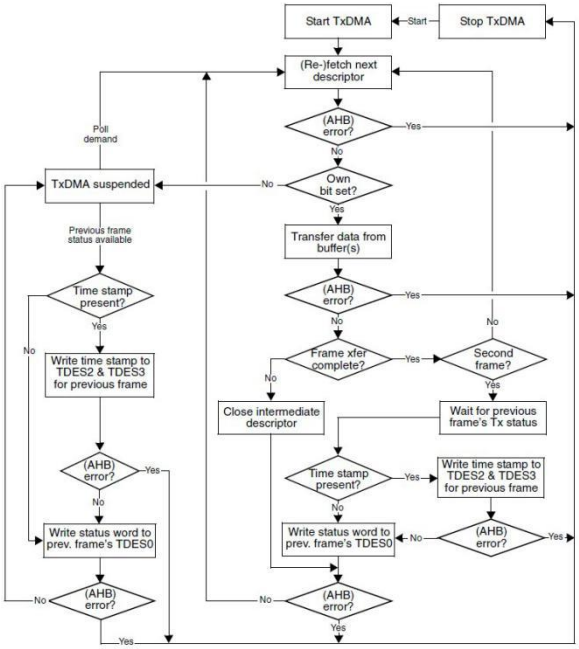
5） DMA 等待之前帧的帧传输状态和时间戳，一旦得到状态，DMA 写时间戳到 TDES2 和 TDES3，如果这样的时间戳可以得到（状态位描述的）。DMA 然后对对应的 TDES0 写状态， 清除占有位，

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）。



3.4.2.5 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

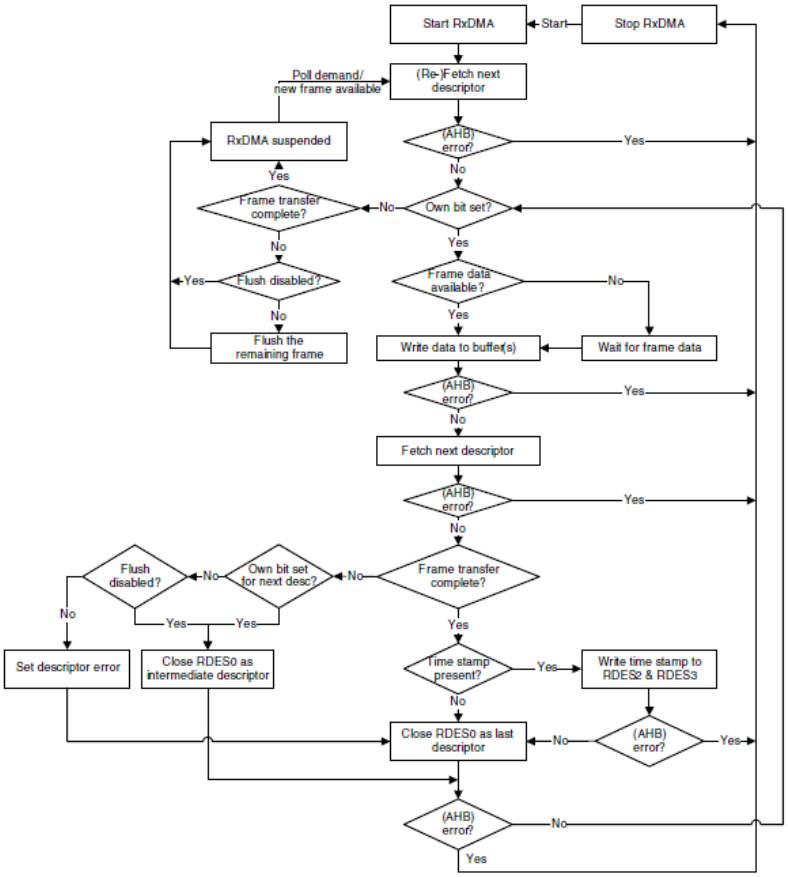
6） 如果当前帧传输完了，DMA 处理过程跳到第 7 步。如果 DMA 不占有下一个取到的描述符， 且帧传输没有完成（EOF 没有被传输），DMA 设置描述符错误位在 RDES0 中（除了刷新不 使能）。DMA 关闭当前描述符（清除占有位），通过清除最后段位（LS, Last Segment）在 RDES0 中的值标记它是中间的描述符，（如果刷新不使能，标志它是最后描述符，）然后跳 到第 8 步。如果 DMA 不占有下一个描述符但是当前帧传输没有完成，DMA 关闭当前描述 符作为中间的描述符，然后回到第 4 步。

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。



3.4.3 MC

3.4.3.1 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 BMC 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 BMC 或者 CPU 读取显存的 IO。

3.4.3.2 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。

3.5 SATA 控制器

3.5.1 可编程寄存器

寄存器详细说明请参考套片寄存器手册。

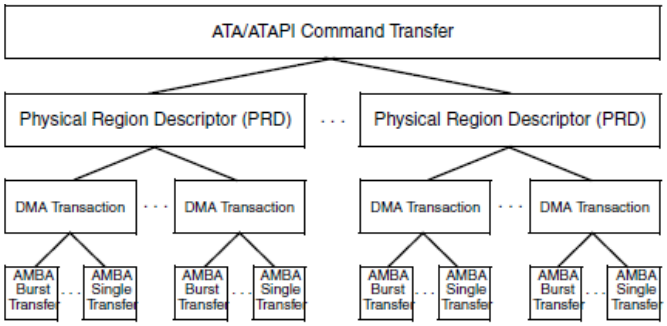
3.5.2 工作流程

基于以下规范:

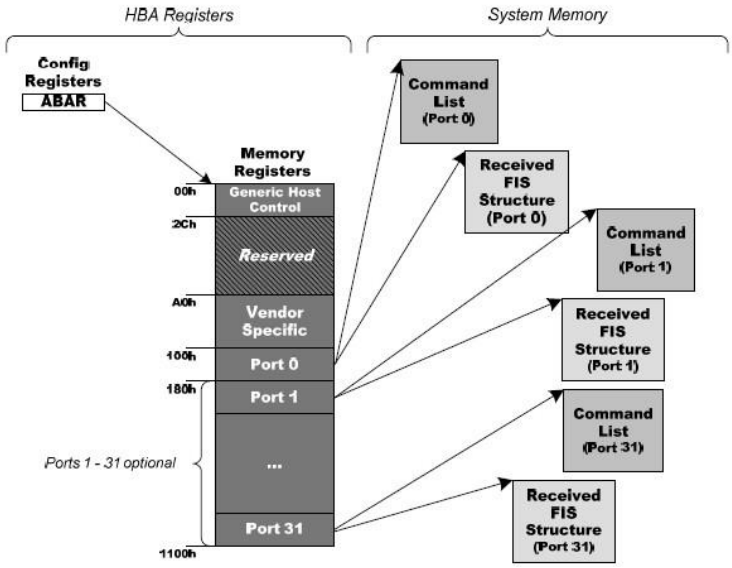
- DesignWare Cores SATA AHCI Databook, Chapter 2.5, Operation Details.
- Serial ATA AHCI 1.3 Specification.

3.5.2.1 3.5.2.1 数据结构

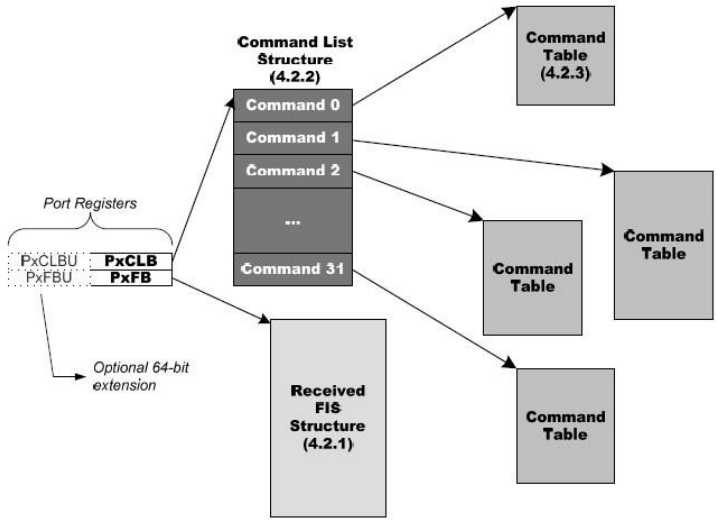
AHCI 数据传输层次如下图所示：



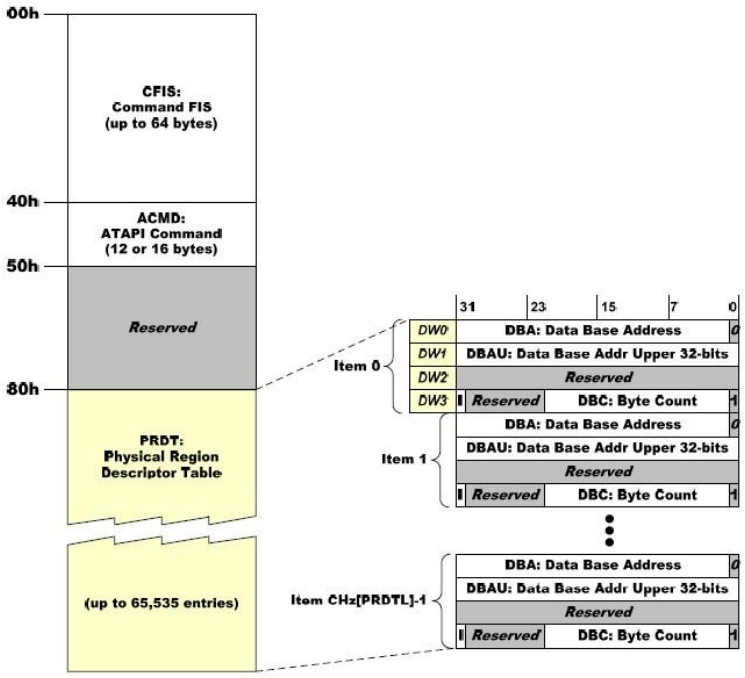
HBA 存储空间分配如下图所示：



端口存储空间分配如下图所示：



命令表结构如下图所示：



3.5.2.2 数据传输

3.5.2.2.1 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。

3.5.2.2.2 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 DMA Activate 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。

3.5.2.2.3 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。

3.5.2.2.4 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。

3.5.2.2.5 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）

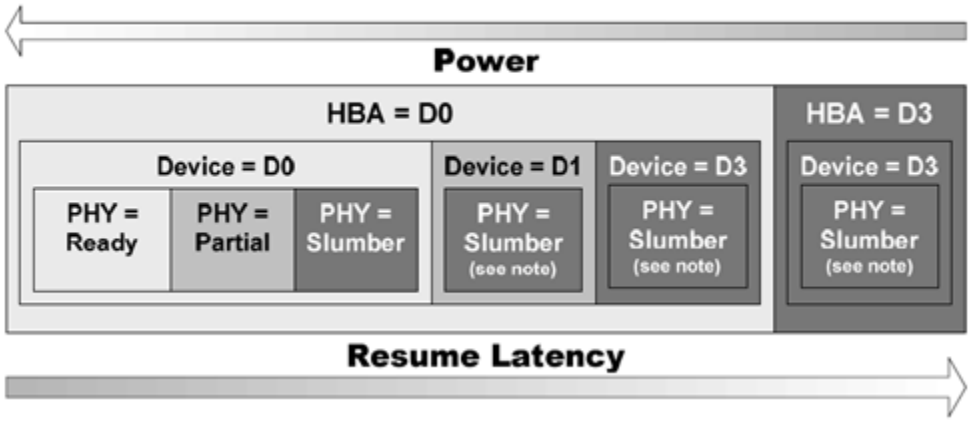
软件可以根据规范改变 RXTS/TXTS 值。通常最大传输大小是 FIFO 深度的一半，除了当 RX FIFO 的深度是 64 双字，这是受限于最小 af_level_s 值，这个值用于阻止 RX FIFO 由于 HOLD-HOLDA 延迟导致溢出。同样，在 AXI 总线环境中，RXTS/TXTS 值可以受限于 CC_MSTR_BURST_LEN 参数。在上电或者异步复位后，RXTS/TXTS 值设置成最大值。

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–N‖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。

3.5.2.4 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**

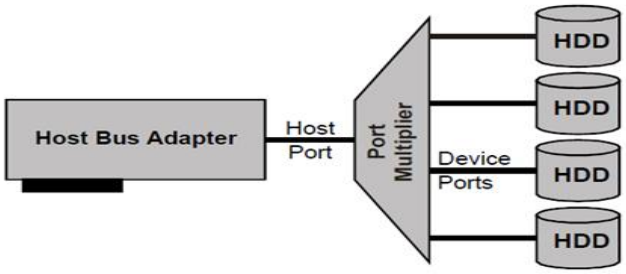
DWC_ahsata 可以通过设置参数 DEV_MP_SWITCH 支持机械在位切换。设计上会增加 1 位输入信号 p#_mp_switch 来表示机械在位切换的状态：

- 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：

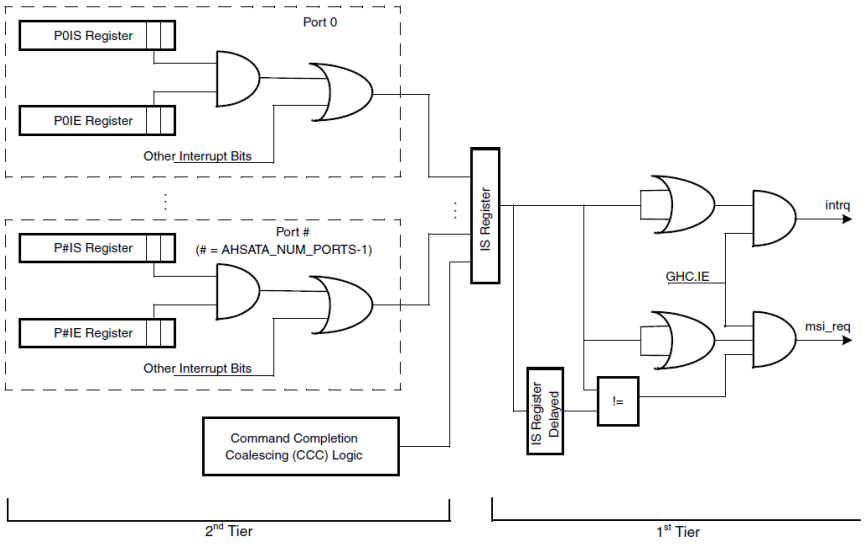


DWC_ahsata 支持基于命令切换的端口扩展器功能。当一个端口连接到端口扩展器上，软件 首先通过发出软件复位到端口复用器的端口 0Fh（控制端口）进行枚举。当对应端口扩展器返 回签名是-端口扩展器‖时，端口扩展器就被连上了对应的端口。如果返回的签名是其他的设备 类型，那么端口扩展器就没有被连接上。DWC_ahsata 支持命令列表越权控制的特征（当 CAP.SCLO=1），通过 P#CMD.CLO 帮助软件可靠地枚举端口扩展器。

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。

3.5.2.6 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。

3.5.2.7 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

3.5.2.8 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-1‖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。

3.5.2.9 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。

3.5.2.10 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。

3.5.2.11 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‘)

3.5.2.12 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。

3.5.2.13 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

3.5.2.14 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–reservedⅡ（读返回 0）

3.6 显示与多媒体子系统

3. 6. 1 DMAC

3.6.1.1 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：

- 要求描述符地址 32B 对界；
- 支持软件复位；
- 支持不同中断源的独立使能。

3.6.1.2 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（32B 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	DAR	目的地址寄存器
0x08	DLR	传输数据长度寄存器
0x0C	NDAR	下一个描述符地址寄存器
0x10	CTLR	DMA 事务控制寄存器，控制传输类型和描述符属性
0x14~0x1C	保留	-

- **单块传输方式**

软件可以通过配置 SAR、DAR、DLR 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
2	DAR	目的地址
3	DLR	传输数据长度
4	CTLR	CTLR[0]=1‘b1

3.6.1.3 中断流程

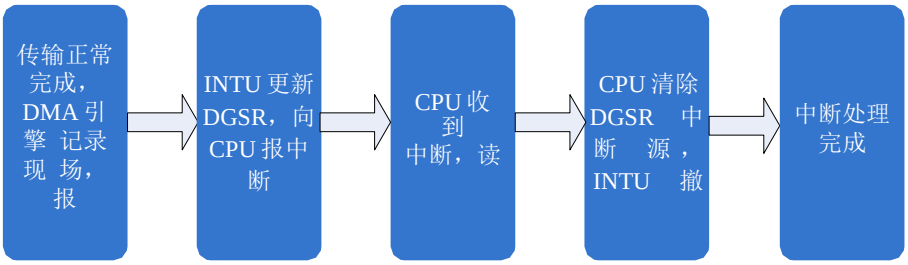
DMAC 通过中断方式与 CPU 交互，下表对不同中断进行了比较。需要注意：对于正常完 成中断，CPU 对 DMAC 进行一次 IO 读，对于其他类型中断，则进行两次 IO 读。

DMAC 中断类型

中断类型	中断描述	中断产生部件	处理方式

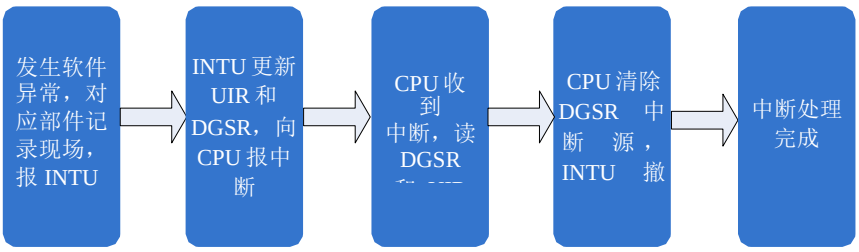
正常完成中断	DMA 传输完成，且传输过程中无错误发生	DMA 引擎（DMARU/DMAWU/DMAVU）	清除中断源
软件异常中断	IO 地址越界	AHB slave 接口/DMA 通道/GDU/AXI Master 接口	
	只读寄存器错误		
	描述符地址非 32B 对界	DMA 通道	
异常完成中断	源端传输错误	DMA 引擎（DMARU/DMAWU/DMAVU）	
	目的端传输错误		
	数据缓冲奇偶校验错误		
硬件故障中断	取描述符传输错误	DMA 通道	
	状态机非法状态错误	AHB slave 接口/DMA 通道/DMA 引擎/GDU/AXI Master 接口	软件复位

➢ 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：

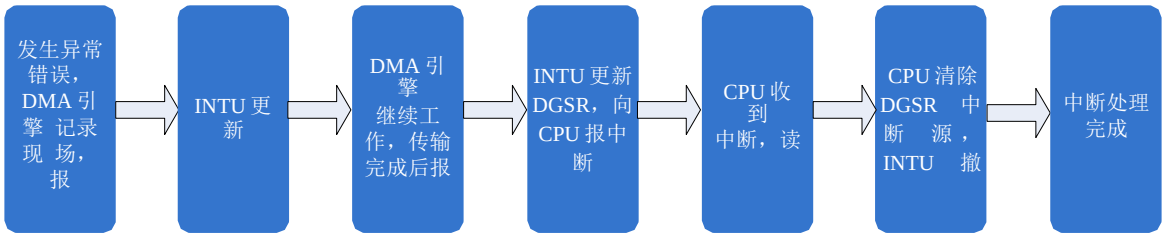
1) 当发生软件异常（IO 地址越界、只读寄存器错误、以及描述符地址非 32B 对界）时，对应 部件（AHB slave 接口或通道）记录中断现场，并向 INTU 发出相应的软件异常信号；

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；

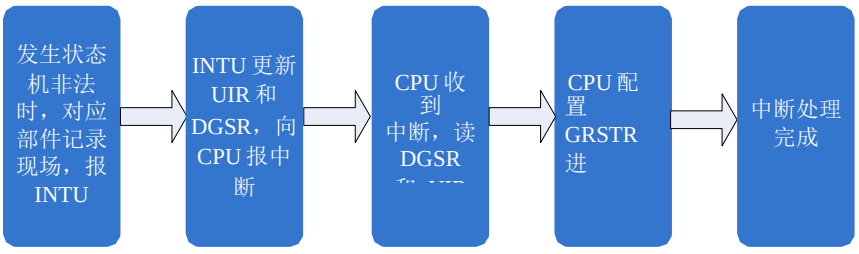
3) DMA 引擎继续后续数据传输，传输完成后向 INTU 发出传输完成中断信号；

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。

3.6.1.4 IO 寄存器

寄存器详细说明请参考套片寄存器手册。

3. 6. 2 DC

3.6.2.1 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。SWICH 中 DC 支持双 DVO 输出接口。DC 支持“交换（Switch）”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。

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

3.6.2.2 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 SWICH 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。

3.6.2.3 参考文档

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

3.6.3 GPU

3.6.3.1 基本功能

SWICH 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。

3.6.3.2 基本操作流程

SWICH 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。

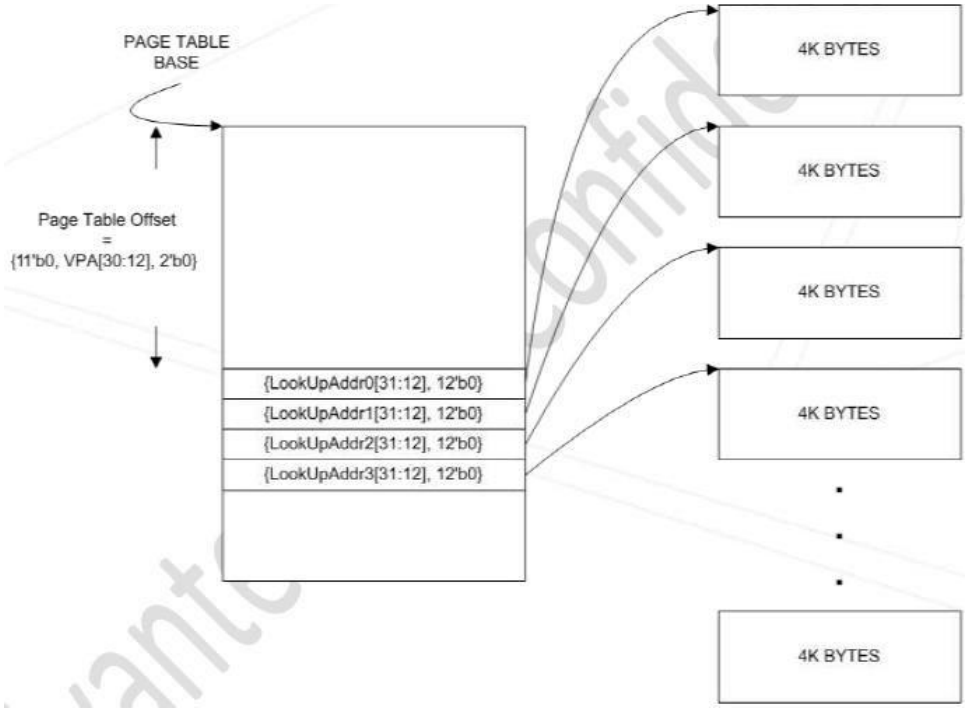
GPU 部件内部访存请求来源包括多个，其中支持虚拟地址访存包括：FE、TX、PE、PEZ 和 RA。每个对应一个页表基址寄存器。访问 Command Buffer 对应 FE 部件页表基址寄存器。

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：



1) 加载 Command Buffer 数据至主存中；
2) 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）。

3) 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。

4) 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。

5) 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。

6) 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。

3.6.3.3 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》。

3.6.4 VPU

3.6.4.1 基本功能

SWICH 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 SWICH 中需 通过 GPU 完成。

3.6.4.2 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 SWICH 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。

3.6.4.3 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 USB 控制器

主要特征如下：

- 支持 USB 2.0（high speed : 480Mbps）、USB1.1 协议（full speed : 12Mbps /low speed : ）
- 支持 AMBA 2.0 AHB 协议的总线接口
- AHB 总线支持 32 位的地址，支持 32bits、8bits 的数据访问宽度
- 实现了一个 EHCI 控制器（USB 2.0），两个 OHCI 控制器（USB1.1）
- 实现了六个 USB 接口，每个 OHCI 管理 3 个 USB 物理端口
- 主机控制器（Host Controller）和 PHY 的接口采用 UTMI+协议，该接口工作在 60MHz、8bits 数据宽度
- OpenHCI 1.0a compatible
- USB 1.1 compatible
- User-configurable Root Hub enables configuration of such parameters as the number of downstream ports and power switching options
- Support for both low-speed and full-speed USB devices
- No bidirectional or three-state buses
- No level-sensitive latches
- Simple application bus interface
- System Management Interrupt (SMI) pin support
- OHCI Legacy support

支持的协议标准版本如下：

- Universal Serial Bus Specification (Revision 2.0, April 27, 2000)
- Enhanced Host Controller Interface Specification for Universal Serial Bus (Revision 1.0, March 12, 2002)
- openHCI: Open Host Controller Interface Specification for USB (Release 1.0a, September 14, 1999)
- UTMI+ Specification, (Revision 1.0, February 25th, 2004)
- EHCI 1.1 Addendum (Revision v0.6, October 2007)
- AMBA™ Specification (Revision 2.0, May 13, 1999)

3.7.1 EHCI 控制器列表处理

List Processor 是主要的控制器，该模块实现了多个操作服务队列的状态机，这些服务队列 由 Host Controller Driver 根据操作寄存器中的相应位建立。

3.7.1.1 LPSMC

描述符预取

描述符预取由 LPLMU 模块初始化（启动）。数据结构 cache 和 EHCI 功能和操作寄存器（EOR）提供地址信息，LPLMU 模块提供控制信号来选择适当的地址。 **数据预取**

数据预取由 List Servicers（iTD、qTD、siTD）初始化启动，并且提供 byte 计数器。系统存 控 LPSMC 从 LPDSC 获得地址和偏移量。如有必要，LPSMC 可以把由 List Servicers 发出的请 求分为多个传输，根据起始地址、byte 计数器，和 OUT 阀值信息。这些多个传输（同时只有一 个处于激活状态），可被初始化为字节传输和双字传输。如果是跨页的，它将中断传输，并自动 的从一个适合的地址重新启动。

IN 数据传输

IN 数据传输由 LPSMC 自动启动。LPSMC 监控 Packet Buffer 接收的数据，当 IN data buffer 到达阀值时启动传输。（如果收到的数据的计数器的值小于 IN 的阀值，LPSMC 会保持 Root Hub 上的传输完成，才会初始化启动一些短包的传输）。LPDSC 提供地址和偏移量和数据预取的流一样格式的。

状态更新

状态更新的传输由 LPLMU 模块初始化启动，使用由 LPDSC 提供的地址和数据。一个传输 的 byte count 是基于状态更新传输的类型的(iTD，QH，qTD，siTD status updates，or writeback QH)

阀值信息

List Servicers 使用输出阀值来初始化到 Root Hub 的 OUT 传输。当 Packet Buffer 的输出数 据达到输出阀值时，该阀值是在 Synopsys specific register 中定义的，LPSMC 允许 List Servicers 启动到 Root Hub 的输出传输。开始数据预取后，当前传输的 byte 计数器达到比 OUT 阀值小两 个双字（2*32bits）时，阀值的检查结束。

在一些时刻以下评估将被执行：

如果以下两个条件中有一个成立，传输将被切断，当 Packet Buufer 可以 hold 住适当大小的 数据时自动重启：

◆ (The amount of remaining bytes to transfer >= OUT threshold value) AND (there is insufficient space available to hold the threshold value of data in the Packet Buffer)

◆ (The amount of remaining bytes to transfer < OUT threshold value) AND (there is insufficient space available to hold the number of remaining bytes to transfer)

LPSMC 使用 IN 阀值信息来启动到 MBIU 的数据传输。一旦 Packet Buffer 中的 IN 数据达 到阀值了，LPSMC 立刻启动到 MBIU 的数据传输。启动 IN 传输后，如果阀值监测发现目前的 传送计数器达到了比 IN 阀值小两个双字（2*32bits）时。此时，将会执行一下的评估：

如果以下的两个条件有一个成功了，就会中断传输，并自动重启在 Packet Buffer 中共有适 量的数据后。

◆ (The amount of remaining bytes to transfer >= IN threshold value) AND (there is not a threshold amount of data available in the packet buffer)

◆ (The amount of remaining bytes to transfer < IN threshold value) AND (the IN data packet has not yet been fully received)

在 Synopsys-specific register 中的 Break Memory Transfer bit 位，强制 host 中断 IN 或 OUT 的传输，当已经传输了阀值的大小的数据时。Break Memory Transfer 特制保证 LPSMC 不会发 出比阀值还要大的数据传输。

3.7.1.2 LPMCU

Master 控制模块实现了顶层的状态机，该状态机定义了 EHCI Host Controller 当前的状态。 当状态在可以一个操作状态，（Run bit 有效），LPMCU 控制 SOF 模块在每一个微包（micfram）的尾生成 micro SOF token。当 micro SOF 生成后，还将触发 List Management Unit 来服务这个 list， 来调度当前的 microframe。当 host 控制器在一个活动状态，micro SOF 生成的过程将在每个一 个 microframe 结束时反复执行，并通知操作过程中的 LPLMU。当 LPMCU 在未激活状态（Run bit 为 0），状态机一直在检测 Run bit 的有效就转换到 Active 状态。A Light Host Controller Reset， or a Host Controller Reset (HCRESET)可会造成这种情况。 Host controller 完成 the Light or HCRESET，并复位 Command register 中的相应位。

3.7.1.3 LPLMU

LPLMU 跟踪所有传输的微帧，为 LPSMC 和 LPDSC 提供控制信号，触发调用相关描述符服务模块(iTD， siTD， QH or qTD servicer)。LPLMU 控制状态机为每一个微帧采样 start-of-microframe 信号，基于传输的类型处理周期性的或非周期性的状态。如果同时同步和异 步的调度器都是使能的，所有同步调度完成后再处理异步的调度。If an isochronous schedule is enabled in microframe zero， the machine proceeds to a periodic state only after fetching the Frame List in a non-empty state.

当控制状态机在一个周期性或非周期性的状态，它会预取一个活动的描述符，并执行这个 描述符调用相关的服务，并根据这个状态更新存储器。这些在 LPLMU 中都是由独立的状态机 管理的，保证同时三个小状态机（预取、处理、状态）只有一个有效

LPLMU 控制描述符预取、描述符处理启动，并更新存储器状态。LPLMU 还处理和状态机 无关的预取、处理和状态更新的逻辑。包括 iTD， QH， qTD， FSTN， and siTD fetch 的请 求。

LPLMU 通过状态机控制 siTD back pointer 处理。在异步调度的处理中，三个独立的状态机 跟踪 Async NAK Counter Reload， Async Active and Sleep State， and Async Advance Doorbell。 LMU 和 the System Memory Controller (LPSMC)， Master Controller Unit (LPMCU)， Data Structure Cache (LPDSC)， Descriptor RAM and Data RAM 有接口。LPLMU 完成实际的 list 处 理启动 USB 传输。

LPMCU 提供控制信号给 LPLMU 在启动 list 处理。主控制状态机管理周期和非周期的调用，启动预取状态机去预取相应的描述符 在配置过程中需要两个队列：处理队列和接收队列。处理队列都是由预取控制状态机写。

对于 IN 传输，检查描述符有存储空间就写这个队列。对于 OUT 传输，当从系统预取了所有数 据后写这个队列。处理状态机读处理队列并初始化启动传输。

结束了 USB 传输后，状态更新在本地描述符 RAM 中完成。已完成的 IN 传输放置在接收 队列中，并被接收状态机读取。对于 OUT 传输，状态由本地 RAM 中更新到系统存储器中。对于传到系统存储器中的 IN 数据，状态有本地描述符 RAM 更新到系统内存中。

3.7.1.4 iTD State Machine

iTD 服务模块在收 LPLMU 发来的 start_service 后开始进行同步传输的服务。如果是一个 OUT 传输，它将通过 LPSMC 接收来着主存的数据。在开始启动 Root Hub 的事务前，它将检查是否有足够时间完成该微帧的传输。如果时间足够，iTD 服务模块检查 FIFO 的阀值，再启动到RH 再到 USB 接口的传输。当从 RH 接收到传输完成信号，iTD 服务模块将 IN 事务的数据和事 务的状态更新到数据结构 Cache 的描述符中事务状态域。它还提供传输完成信号给 LMU。如果

（已传的）字节数没达到要求，它将启动另外一个传输来，重复这个动作。

对于 OUT 事务，iTD 服务模块需要等待数据从系统主存预取来。使用到 FIFO 的阀值，这 个值也是可以用户配置的。举例来说阀值为 64 字节，而事务的大小是 512 字节，包缓冲中的数 据加上 RH 内的 32 字节数据达到 64 字节，iTD 服务模块立刻启动 RH 的传输。

对于 IN 事务，LPSMC 自动启动数据传输。iTD 服务模块在 IN 数据都传输完成后发出 Transfer Done 信号到 LPLMU。

下面列举一下 iTD 服务模块处理的主要错误类型：

- 数据预取跨过了微帧或帧的边界，USB 的传输就不能执行了，访存请求也立刻结束。如果 事务在 USB 接口上已经启动了，（原因是 OUT 阀值比事务的长度小），会引起一个缓冲错。
- 一个 IN 传输跨过了微帧或帧的边界，是不作为出错处理的。因此，iTD 服务模块等到数据 都传给主存了将发一个 Service Done 信号给 LPLMU
- 在缓冲 underrun/overrun 的情况下，Buffer Error 位会置起来。（对于 IN 传输，所有在包缓 冲里的数据都将传给主存）
- 如果 IN 数据发生了 CRC 错误，数据还是传出去，状态寄存器中的 Xact Error 位将置起
- 过超时或 Bad PID 响应，就不会传数据了，状态寄存器中的 Xact Error 位将置起
- 当包紊乱，数据还是传输的，状态寄存器中的 Babble Detected 位将置起
- 当包短时，收到的任何数据都传到主存去。
- CRC 错，Bad PID, Timeout, the Packet Babble 等错误，USB 中断位将置起，产生中断。

3.7.1.5 siTD Servicer

该模块处理同步的分块传输描述符，操作类似于 iTD。LPLMU 处理一个 siTD 描述时，将 激活 siTD 服务于当前的微帧。执行结束后，控制权还给 LPLMU。LPSITD 根据相应的 split mask 和描述符中的 SplitXstate 执行 Start Split 和 Complete Split 传输。对于一个 OUT 传输，将有一 个或多个 Start Split（SSPLIT），对于一个 IN 传输，有一个 Start Split（SSPLIT）或者一个或多 个 Complete Splits（CSPLIT）。

OUT 事务从主存预取数据，给 RH 模块相关控制信号。RH 使能后，LPSITD 状态机等 RH 完成 USB 端口的传输。在控制器交给 LPLMU 前，结果状态和其他请求描述符域更新到 DSC 中。对于一个 IN 事务，RH 从端点接收数据，当 RH 的事务结束或者数据大小达到包缓冲的阀值后，立刻开始到主存的数据传输。为 OUT 事务预取数据时，LPSITD 状态机检测当前微帧是 否时间足够，再启动 RH 的传输。如果时间不足，而描述符中完成后中断使能位置上了，LPSITD 状态机将生成一个中断。

在一些特殊的条件下，LPSITD 处理一个后向指针的描述符。当取一个后向指针的描述符时，

LPSITD 将控制权交给 LPLMU，带上一个预取标志。LPLMU 预取描述符，并启动一个新的传 输。LPLMU 给出其他模块需要的控制信息。如果一个后向指针 siTD 激活了，它将执行下去。 如果没有激活，或者当前的传输将其激活为设置为 0，LPLMU 将控制权交给 LPSITD 模块来继续进行原来的 siTD 处理。

LPSITD 处理的主要的错误包括：

- 一个 IN CSPLIT，LPSITD 状态机收到一个 ERR 握手包，将设置状态位的 ERR 位为 1，并 将激活位设置为 0
- 在 IN CSPLIT 过程中，数据 CRC 错，超时，LPSITD 状态机将置 Xact 错误位。超时或 CRC 错，将重试两次，然后将激活位置 0
- 如果是包 babble 错，数据将数据发送给主机，状态寄存器中的 Babble Detected 位将置起
- 如果一个 OUT 传输微帧的数据预取越界了，访存传输立刻结束。如果该传输由于大小小于 OUT 阀值，已经在 USB 上启动了，报缓冲错结果
- 一个 IN 数据传输微帧的数据预取越界了，是不做错误处理的。

3.7.1.6 qTD 和 QH Servicer（LPQTD）

qTD 模块控制由 QH 表和队列传输描述符（qTD）定义的传输、控制传输和中断传输。qTD 数据结构在系统主存中是大小为 32 字节、连续的、32 字节对界的，一个 qTD 定义的传输最大可以传的数据大小为 20480Bytes（5*4096），有五个缓冲指针。qTD 访问的的缓冲必须是连续的，可以是任何字节对界的。qTD 退出的条件只有如下几条：

- 全部缓冲都传输完成后
- 错误计数器（Cerr）由 1 到 0 的跳变

当队列管理模块读到一个 qTD 传输描述符后就触发该模块进行处理。在处理过程中，LPLMU 既可能遇到周期性列表中的中断传输，也可能遇到非周期性列表中的快传输或控制传 输。一旦传输完成，LPQTD 就将控制权转交给 LPLMU。

LPQTD 基本功能包括：

- 初始化启动到 Root Hub 的块、控制、中断传输
- 在系统主存中取得 OUT 事务的数据
- 当事务的数据大小大于 MaxPacketSize 时，传输多个包
- 在数据结构 cache 中更新描述状态
- 给 LPLMU 提供传输的状态
- 如果完成时生成中断信号有效，生成控制信号生成一个中断请求
- 重新更新 NAK 计数器
- 支持块、控制、中断的分裂的事务。

QH 服务模块在收到 LPLMU 的 start_service 信号后开始启动周期性/非周期性的传输。QH 首先检查端点的传输速率，然后确定是否有需要 split 的传输。如果端点是全速或低速的，QH为当前的端点提供 split 传输。如果端点是高速的，QH 服务处理一个普通的 QH 传输。对于块、 控制传输，QH 服务模块必需确定是否需要生成一个 PING token，或者 NackCount 是否需要重 新开始计数，这些都基于 QH 中的描述符。参考 EHCI 手册，这个过程中断传输是不需要的。

对于 OUT 传输，QH 服务模块从系统主存控制器取得数据。在启动到 Root Hub 的事务前， 它将检查是否有足够的时间完成该微帧的传输。如果时间足够，QH 服务模块检查 FIFO 的阀值并开始 Root hub 的传输。，根据从 Root Hub 收到的完成信号，QH 服务模块检查目前传输的状态。如果状态检查返回包 babble 或短包错误， QH 服务模块设置描述符中的状态位，更新 LPDSC 中的描述符，并给 LPLMU 列表服务完成信号。如状态结果中 STALL pID 或 CERR 从 1变为 0，QH 服务模块将描述符的暂停（Halt）位设置为 1，活动位（Active）设置为 0，更新 LPDSC 中的状态，并发 List Service Done 给 LPLMU。如状态检查返回无错，且事务计数器（对于中断传输）或者 park mode 模式计数器（对于块传输）为非零，QH 服务模块启动另外一个传输。当一个传输完成，更 LPDSC 中的描述符的 Transaction Status 域，并发 List Service Done 给 LPLMU。

QH 服务模块执行以下传输，根据 QH 描述符中的域：

- Interrupt Transfer: sets the S-mask field for the current microframe
- Split Transfer: sets the SplitXstate descriptor for full or low speed devices
- Ping Transfer: defines the ping state descriptor for high-speed endpoints
- Control Transfer: sets the Control endpoint flag in the descriptor
- Bulk Transfer: sets the Bulk endpoint flag in the descriptor

qTD 服务模块处理以下错误：

- 如取数据的边界超过了微帧（或帧）的边界，该传输不会在 USB 总线上执行，访存事 务也会被立刻停止。如果事务以及在 USB 总线上启动了，报 buffer error 结果。
- 如果到主存的 IN 数据传输超过了微帧（或帧）的边界，是不作为错误处理的。qTD 等待所有的数据都传到主存了，给 LPLMU 发送 qtd_srvc_done 信号。
- 如果缓冲不满或溢出了，qTD 服务模块将状态中的 Buffer Error 置为有效
- 如果 IN 数据 CRC 错，qTD 服务模块传输该数据，并将 Xact error 位设置为有效
- 如果有超时或 bad PID 响应，qTD 服务模块描述符状态中的 Babble Detected 位设置为 有效
- 如果有一个短包，qTD 将数据从 USB 设备传到主存
- 如果有 CRC 错，Bad PID, Timeout, Packet Babble or CERR 值变为 0，如果报错中断使 能，qTD 服务模块生成中断。

3.7.1.7 Operational Registers

该模块存储着 EHCI 的功能和操作寄存器，Auxiliary Well中也是操作寄存器的一部分，但 在别的模块中实现

3.7.1.8 Start-of-Frame (SOF) Generator

帧启始包（SOF packets），生成时包括一个 SOF 计数值，并为微帧生成微 SOF。微帧的长 度（duration）由 the Frame Length Adjustment (FLADJ) register 的值决定，FLADJ 寄存器由 the Application Strap Signals 配置，通过自陷配置的 FLADJ 值必须是相同的。这样就保证了 host 的 微帧长度和每个端口的微帧长度相同。

3.7.1.9 Packet Buffer (PBUF)

包 BUF 只能在 CONFIG1 状态访问，在 CONFIG2 状态，使用外部数据和描述符 RAM。 包 BUF 为 IN/OUT 数据提供存储和控制。在 OUT 事务过程中，LPSMC 从系统主存中预取数据并写到 BUF 中。在 IN 事务过程中，数据由 Root Hub 写入。coreConsultant 可以配置包 BUF 的大小，默认的大小是 128 x 32 (512 bytes)。

3. 7. 2 OHCI 控制器列表处理

List 处理模块是 OHCI 中的主要控制模块。有多个状态机来实现 list 服务流程，list 优先级，USB 状态，ED（Endpoint descriptor）和 TD（Transfer descriptor）服务，状态回写，和 TD 结束。另外，该模块还作为 HCI master 和主机串行接口引擎（HSIE）的接口，完成 USB 到主存、主 存到 USB 的数据传输。
List 处理模块包括以下子模块：

- USB States Block
- List Service Flow Block
- ED-TD Block
- HCI Master Interface Logic
- Data Read-Write Logic

3.7.2.1 List Service Flow Block

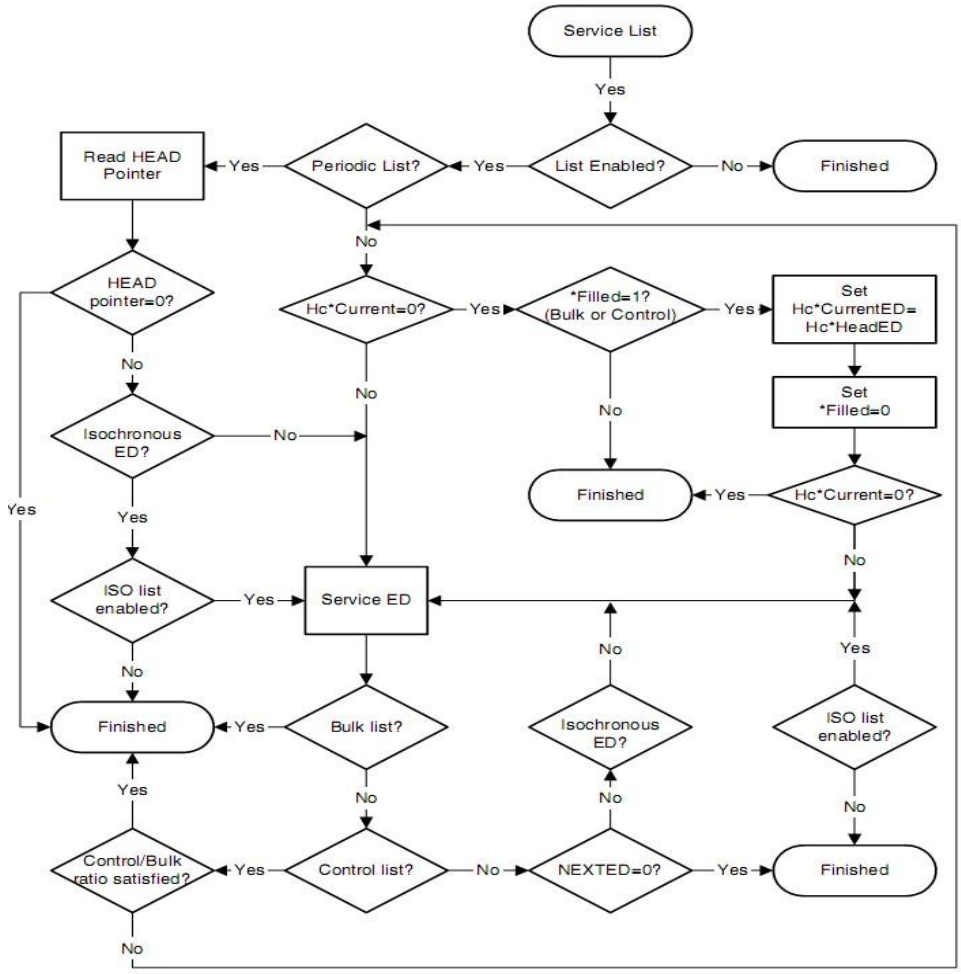
该模块实现了一个 list 服务流程状态机。在每一帧中当 SOF 发送到 USB 后，该状态机由 USB

状态状态机触发（进行状态转换）。一旦确定要服务哪个 list，LSF 状态机就触发 ED-TD状态机去服务传输（list）中的一个包（paket）。一个包传完后，ETD 状态机再控制 LSF 状态机。这个流程在每个包每个传输都是重复的，直到当前的帧传输完成再或者到了帧尾（EOF）。到 EOF 后，控制器权又交给了 USB 状态状态机。

该模块还实现优先级逻辑，确定服务哪一个 list (periodic/nonperiodic)，基于 HCI 的寄存器 的内容。在非周期性 list，基于 HCI 的 Control Bulk Service Ratio (CBSR)确定哪种类型的传输（控 制或批量）。

根据优先级逻辑的提示，LSF 状态机处理数据结构，来确定是否有一个端点（ED）。如果 无 ED，LSF 状态机告知优先级逻辑。如果存在 ED，LSF 状态机提供有效信息给 ED 状态机来 服务 ED。

下图介绍了 LSF 状态机确定是否存在 ED 需要服务。

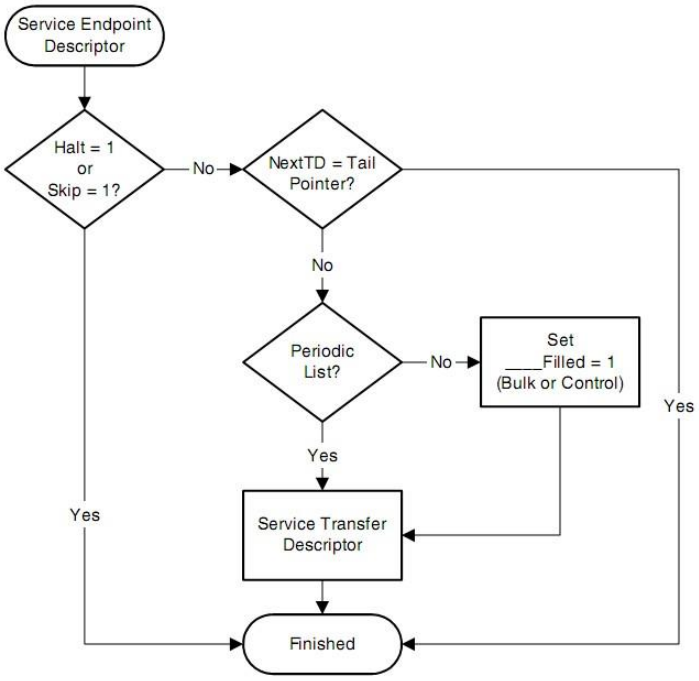


3.7.2.2 ED-TD 模块

每个包传输时，该模块都会触发 LSF 状态机。ETD 状态机的逻辑可以分为以下两个功能模块：

ED 服务流程图

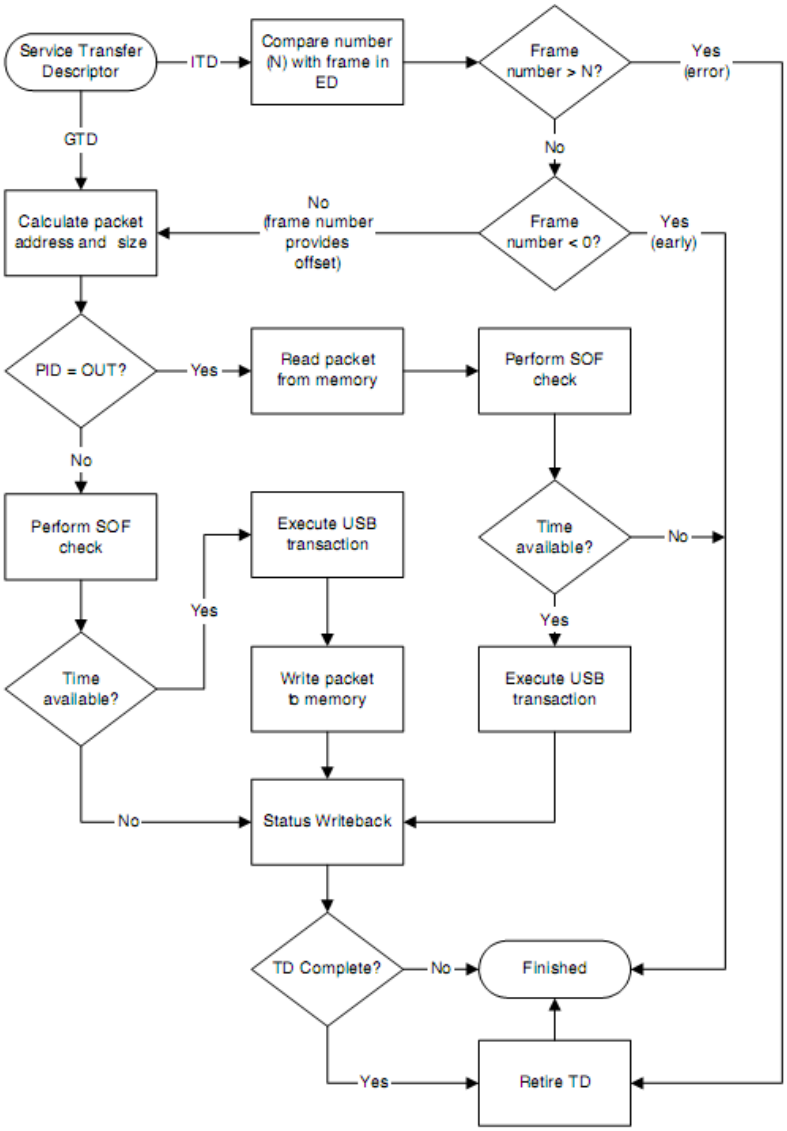
端点描述符（ED）状态机确定一个 TD（传输描述符）是否被执行。如果一个 TD 不能执 行，ED 状态机将通知 LSF 状态机。如果一个 TD 存在，ED 状态机给 TD 执行需要的信息。



TD 服务流程

TD 状态机根据 ED 提供的信息开始工作，当从系统主存取到传输描述符，TD 谈自己和 Framing 模块和地址、大小计算模块有接口，来确定传输的同步帧号是否正确，以及在 USB 端 口上传输这个帧的时间是否足够（in the 1ms frame period）。如果帧号的数值和传输时间要求可 以满足，TD 状态机启动 USB 端口的传输，方法是给 RH 和 HSIE 模块相关请求并读写 FIFO。 如果不满足要求，USB 上的传输不会被初始化。

在成功的或不成功的 USB 传输结束时，TD 将控制权交给状态回写模块（the Status Writeback block），将状态更新到系统主存中



3.7.2.3 HCI-Master 接口逻辑

该总线接口模块，实现了数据和地址的 MUX，生成读写系统主存的地址和数据。该模块还 实现了从系统主存取来的 ED (4-DWORD)和 TD (8-DWORD)的寄存器。另外，如果该模块实现 了一下三种任务：状态回写、TD 退出、地址和包大小计算。

状态回写：当 TD 状态机结束到 USB 的数据传输，状态回写信号更新系统主存中传输的报告**TD 退出**：当一个 TD 的所有的数据都成功的传输了，或者出错了，TD 退出逻辑将 TD 放入 TD 完成队列。

地址和包大小计算：该模块执行 TD 地址和包大小计算，该信息传给 TD 状态机

3.7.2.4 数据读写逻辑

数据读写（DRW）逻辑实现了一个状态机控制数据在 USB 和系统主存间的传输。它将主 机串行接口引擎（Host Serial Interface Engine -HSIE）和 HCI master 同步起来。

串行接口引擎把 IN 事务的从端点读到的数据包存储在数据 FIFO 里（DFIFO）。当所有的数 据都接收了（for GTD/ITD），或者 DFIFO 中的数据都到达一个阀值了（≥ 16 bytes），ETD 状 态机就触发 DRW 状态机发出主存写命令。DRW 状态机提供地址和数据大小（bytes），触发 HCI Master 的写周期。如果需要写超过 16 字节，DRW 状态机执行多个 16 字节的周期，最后一个周 期写剩余的数据。DRW 为每个周期进行地址递增计算。它重复该序列，直到 RH_HskRdy(该信 号表示所有的数据都从 USB 得到了)，然后停止写周期直到 DFIFO 为空了。在传输的最后，它 将控制权交给 ETD 状态机。

类似的，对于 OUT 包，ETD 状态机触发 DRW 状态机从主存读数据。HCI Master 接口逻 辑提供地址和要读的数据的大小（bytes）。再次，如果数据是大于 16bytes 的，DRW 状态机提 供多个 HCI Master 周期去（从主存）读。一个传输结束后，DRW 将控制权交给 ETD。

地址和数据大小将路由到 HCI Master 模块，HCI Master 模块将地址和数据多选一。该状态 机只处理段端口相关的读写。而 HCI Master 逻辑处理所有的其他的读写（如状态回写和 list 头 指针的读）。

3.7.2.5 Root Hub and HSIE Blocks

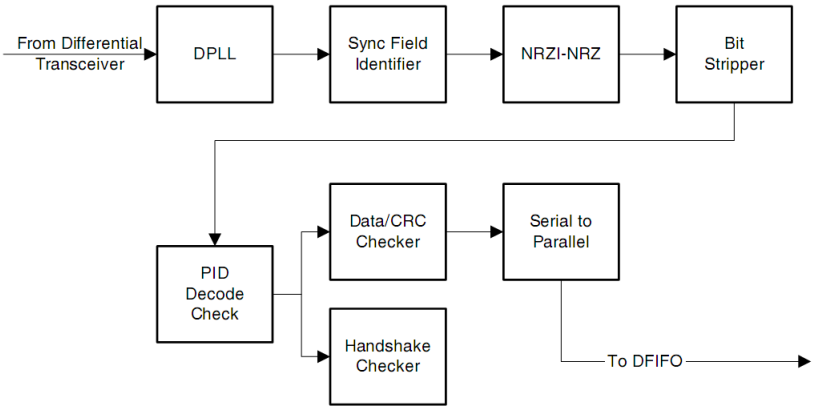
- Reset_Resume

当有一个 SOF 发送给全速端口时，该复位、唤醒子模块实现一个状态机来生成和传播低速 保持活动信号给所有的低速端口。低速端口需要保持活动，因为它们不会译码 SOF。该子模块 还实现了一个计数器运行在 12MHz 时钟，并为每一个毫秒生成一个周期宽度的脉冲。在端口配 置模块中的服务和唤醒计数器使用这个脉冲来进行复位和唤醒计时。

- Digital Phase-Locked Loop
- Host Serial Interface Engine (HSIE) State Machine

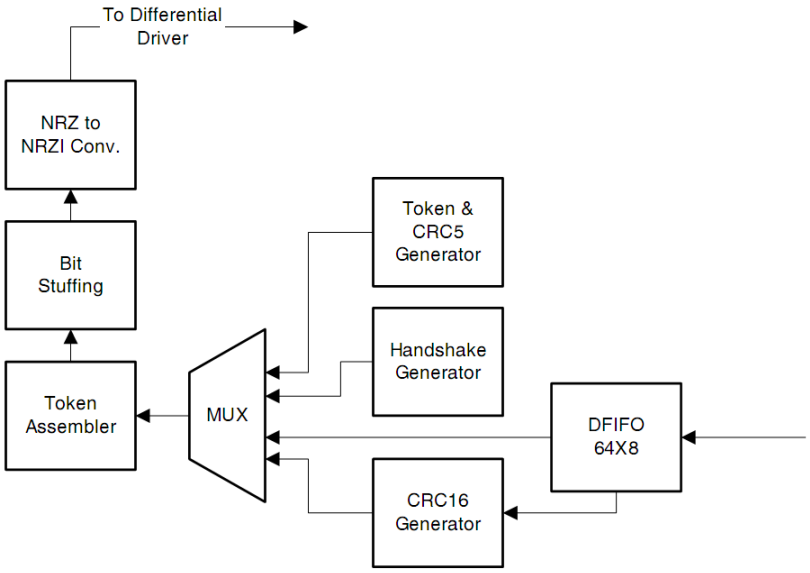
主机串行接口引擎通过 D+和 D-信号，接收和传输 USB 协议的数据。在 USB 数据接收过 程中，D+和 D-信号从差分接收器传过来，是一个 single-ended bit stream，该数据流按顺序经过 PLL 模块解析出外部的时钟和数据信息。PLL 模块将时钟和数据信息送给 SIE 模块，识别出同 步范式，并进行 NRZI-NRZ 转换。NRZ 结果数据经过比特剥离（Bit Stripper），该剥离器将插入的 0 去掉。数据流首先结果 PID 译码检查来识别出不同的 PID。而 HSIE 值模块根据 PID 的类型管理协议。如果是一个数据 PID，串行数据将编译为字节的模式，而接收数据的 CRC 是在数 据接收的同时计算的，然后将数据存入 DFIFO。接收引擎的数据流程如下：

HSIE Receive Data Path （HC 的输入）



在 HC 数据发送过程中，HSIE 模块负责将 DFIFO 中取出的字节格式的数据，再将其转换 为串行数据。如果数据位填充，从 NRZ 转换为 NRZI 格式，从差分驱动器（HC 外部）传输给 USB 端口。当有一个握手包时，SIE 模块集合其适当的握手包，并将其通过 USB 端口发出。该 发送引擎的数据流程图如下：

HSIE Transmit Data Path（HC 的输出）



HSIE 包括以下三个子模块：

- 接收模块
- 同步模块
- 传输模块
  - ❖ RCFG_Dpls：Multiplexed D+；This is the bit-wise AND of all D+ lines from different ports.
  - ❖ RCFG_Dmns：Multiplexed D–；This is the bit-wise OR of all D– lines from different ports.
  - ❖ RemoteWakeUp (upstream Resume from suspended peripherals) ： Remote Wakeup Signal from

Downstream Ports

This signal is a simple OR of remote wakeup events from all ports.

- RCFG_RmtWkp = | UP_RmtWkp (where UP* are of width NDP)

3.8 可信计算模块 TCM

处理器通过高速密码算法模块和命令通道模块与 TCM 通信。高速密码模块采用 DMA 控制 器实现。DMA 控制器为专用控制器，共设置 7 路通道负责算法模块的数据传送。通道由软件配 置，支持单块传输模式、基本链表传输模式和扩展链表传输模式。命令通道模块主要用于处理 器与 TCM 之间的请求和应答通信。

3.8.1 DMA 传输模式

DMA 通道用于控制 DMA 的传输，支持三种传输模式，单块传输模式、基本链表模式和扩 展链表模式。

3.8.1.1 单块传输模式

单块传输模式不需要从软件中读取配置数据，相应的传输控制参数由软件直接配置到通道中，其工作步骤如下，

1) 确认当前通道空闲（DGSR[0]）；
2) 配置 CHx_CR、CHx_ASR/CHx_DAR 和 CHxDLR； 3) 选择单块传输模式；
4) 启动 DMA 通道（清除并写通道使能位）；
5) 开始传输；
6) 传输完毕置中断信号。

3.8.1.2 基本链表模式

基本链表模式需要软件首先初始化一个通道描述符表，通道描述符包含通道的控制信息以及下一个通道描述符的地址。软件先将第一个通道描述符的地址写到 CHx_CLNDAR 中，然后 启动 DMA。DMA 通道先从地址 CHx_CLNDAR 中读取通道描述符，根据描述符的内容配置通道参数和下一个描述符地址（CHx_NLNDAR）并开始传输。传输完毕后判断当前通道描述符是 否是最后一个，如果是

则 DMA 传输完成，如果不是则将 CHx_NLNDAR 的内容拷贝到CHx_CLNDAR 中继续读取下一个通道描述符进行传输。

1) 初始化通道描述符表；

2) 选定空闲通道；

3) 将第一个通道描述符的地址写到 CHx_CLNDAR 中；

4) 选择基本链表模式；

5) 启动 DMA 通道，读取第一个通道描述符并配置通道参数；

6) 开始传输直到所有通道描述符传输完毕；

7) 置中断信号。

3.8.1.3 扩展链表模式

在扩展链表模式中，软件需要首先建立通道描述符表和链表描述符表，链表描述符包括当前链表的第一个通道描述符地址和下一个链表描述符的地址。软件将第一个链表描述符地址写 入 CHx_CLSDAR 然后启动 DMA。DMA 从地址 CHx_CLSDAR 中取得第一个链表描述符，包 括第一个链表的第一个通道描述符地址和下一个链表描述符的地址（写入 CHx_NLSDAR 中）， 然后 DMA 读取第一个通道描述符进行传输。当完成一个通道描述符表时，判断如果当前通道 描述符为链表描述符表的最后一项的最后一个通道描述符，则本次 DMA 传输完成，如果不是 则将 CHx_NLSDAR 的地址拷贝到 CHx_CLSDAR 中继续读取下一个链表描述符进行传输。

1) 初始化通道描述符表和链表描述符表；

2) 选定空闲通道；

3) 将第一个链表描述符的地址写到 CHx_CLSDAR 中；

4) 选择扩展链表模式；

5) 启动 DMA 通道，读取第一个链表描述符；

6) 据链表描述符读取第一个通道描述符；

7) 开始传输直到完成所有通道描述符；

8) 置中断信号。

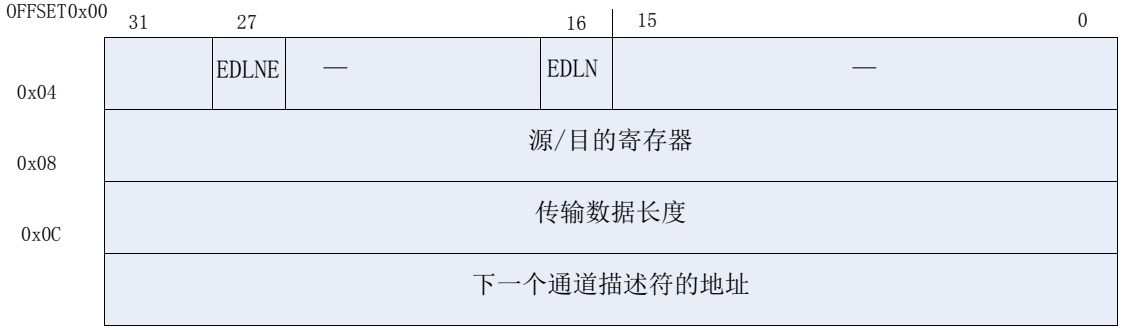
3.8.2 DMA 描述符

描述符分为通道描述符和链表描述符，通道描述符控制通道传输中的地址、长度等具体信息，其它的控制参数需要软件配置。链表描述符描述了通道描述符表的起始地址和有关该链表

的一些控制参数。

3.8.2.1 通道描述符

通道描述符在内存中的结构见下图，DMA 传输时，软件还需要对通道控制寄存器（除了通 道描述符已定义的控制位）的其它控制位进行配置。



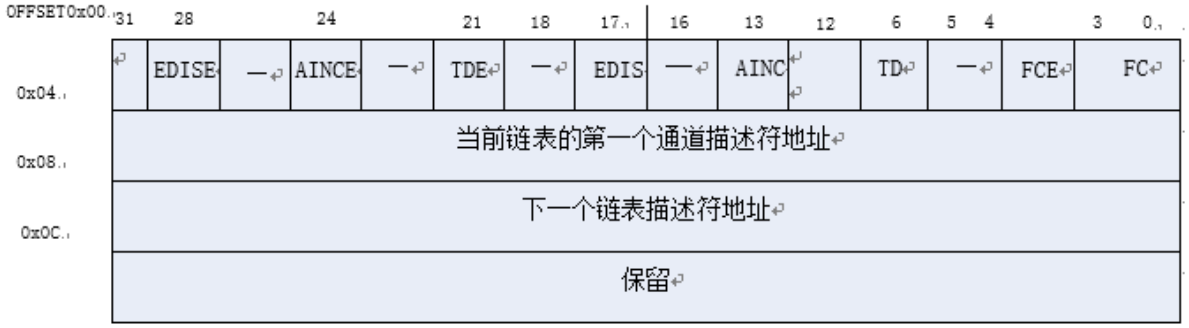
通道描述符控制字段的定义。

通道描述符控制字段定义

比特	名字	定义
0-15	—	保留
16	EDLN	为 1 表示当前通道描述符为链表的最后一个描述符
17-26	—	保留
27	EDLNE	EDLN 写使能，高有效
31-28	—	保留

3.8.2.2 链表描述符

下为链表描述符在内存中的结构。DMA 传输时，软件还需要对通道 控制寄存器（除了链表描述符和通道描述符已定义的控制位）的其它控制位进行配置。



链表描述符各字段的定义如下：

比特位	名字	描述
2-0	FC	传输控制，该字段决定了一个通道占用传输引擎一次最多能传多少个字节，传输完 Chx_CR[FC]个字节后传输引擎使用权将被收回，仲裁器重新仲裁通道请求。 1 16 bytes 2 32 byste 10 64 bytes 11 128 bytes 100 256 bytes 101 512 bytes 110 1024 bytes 111 通道独占，不进行仲裁
3	FCE	FC 字段的写使能，高有效
5-4	—	保留
6	TD	传输方向， 0 将数据由源/目的地址寄存器传输到算法模块 1 将数据由算法模块传输到源/目的地址寄存器
7-12	—	保留
13	AINC	地址增量控制 0 地址递增 1 地址固定
14-16	—	保留
17	EDLS	为 1 则表示当前链表描述符为最后一个描述符

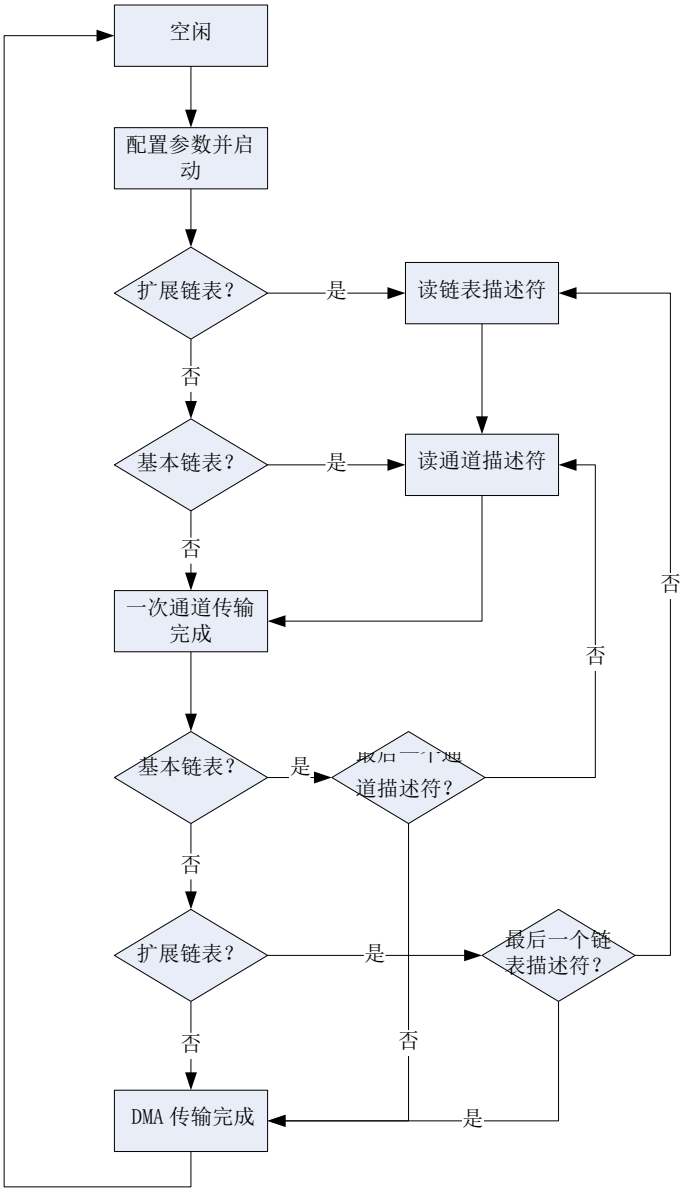
18-20	—	保留
21	TDE	TD 写使能，高有效
20-23	—	保留
24	AINCE	AINC 写使能，高有效
25-27	—	保留
28	EDLSE	EDLS 写使能，高有效
18-31	保留	—

3.8.3 DMA 通道工作流程

当软件调用密码服务时，首先需要确定空闲通道，然后配置通道控制寄存器，选择传输模 式，启动通道。通道发出总线使用仲裁，仲裁器分配总线使用权，得到授权的通道开始传输数 据。为了最大利用总线性能，传输时做如下约定：

1) 通道向算法传送数据时，如果接收 FIFO 内有最大总线突发字节数(256 字节)的空间且 本次通道传输还未完成，则通道向仲裁器发出仲裁请求；
2) 通道从算法接收数据时，如果发送 FIFO 内有最大总线突发字节数(256 字节)的数据或 是最后一次突发传输时（最后一次传输发送 FIFO 中的数据有可能达不到最大总线突 发长度），则通道向仲裁器发出仲裁请求；

通道突发是指通道获得授权后不超过流控字段（CHx_CR[FC]）规定的最大数据长度的传输。 通道每次能够传输的最大字节数目由流控字段（CHx_CR[FC]）决定，通道则按照尽量以最大总 线突发传输数据。当流控字段（CHx_CR[FC]）大于一次总线突发时则分多次传输，不够一次总 线突发时则按照实际数目发送。一旦通道将 CHx_CR[FC]规定的长度传输完毕，就将总线的使用权交出。如果此时通道传输传输完毕（CHx_CDLR 为 0），则本次通道传输完成，否则根据收发 FIFO 的状态继续发出仲裁请求。算法要求数据长度必须是 128bit 的整数倍，因此通道只支 持总线 SIZE 以 128bit 的方式发送，4 字节算法属性字应填充至 16 字节。通道工作流程图如下。



通道工作流程图

3.8.4 3.8.4 DMA 中断寄存器

高速密码模块以中断的方式和 CPU 交互，CPU 检测到来自 TCM 的中断信号后，首先查询 DMA 全局状态寄存器（DGSR），以确定发出中断的部件，然后再进一步查询具体的中断信息（CHx_SR）。

中断产生：

1) OUT_FIFO 有数中断；
2) DMA 传输完毕中断；
3) 编程异常中断；
4) 由 OPENRISC 发出的中断。 **中断撤销：**

向中断寄存器对应位写 1 撤销该中断。

3.8.5 DMA 编程接口地址

下表为高速密码模块的编程接口地址。

高速密码模块编程接口地址

addr[31:16]	addr[15:12]	addr[11:0]	含义
0x0090	**全局寄存器**		
	0x0	0x000	DGSR
		0x004	ARCR
		0x008	ASR
		0x00C	AFSR
	CH0		
	0x1	0x000	CH0_CR
		0x004	CH0_SR
		0x008	CH0_CLNDAR
		0x00C	CH0_NLNDAR
		0x010	CH0_CLSDAR
		0x014	CH0_NLSDAR
		0x018	CH0_SAR/CH0_DAR
		0x01C	CH0_DLR
		0x020	CH0_CDLR
	CH1		
	0x2	0x000	CH1_CR
		0x004	CH1_SR
		0x008	CH1_CLNDAR
		0x00C	CH1_NLNDAR

		0x010	CH1_CLSDAR
		0x014	CH1_NLSDAR
		0x018	CH1_SAR/CH1_DAR
		0x01C	CH1_DLR
		0x020	CH1_CDLR
	CH2		
	0x3	0x000	CH2_CR
		0x004	CH2_SR
		0x008	CH2_CLNDAR
		0x00C	CH2_NLNDAR
		0x010	CH2_CLSDAR
		0x014	CH2_NLSDAR
		0x018	CH2_SAR/CH1_DAR
		0x01C	CH2_DLR
		0x020	CH2_CDLR
	CH3		
	0x4	0x000	CH3_CR
		0x004	CH3_SR
		0x008	CH3_CLNDAR
		0x00C	CH3_NLNDAR
		0x010	CH3_CLSDAR
		0x014	CH3_NLSDAR
		0x018	CH3_SAR/CH1_DAR
		0x01C	CH3_DLR
		0x020	CH3_CDLR
	CH4		
	0x5	0x000	CH4_CR
		0x004	CH4_SR
		0x008	CH4_CLNDAR
		0x00C	CH4_NLNDAR
		0x010	CH4_CLSDAR
		0x014	CH4_NLSDAR
		0x018	CH4_SAR/CH1_DAR

		0x01C	CH4_DLR
		0x020	CH4_CDLR
	CH5		
	0x6	0x000	CH5_CR
		0x004	CH5_SR
		0x008	CH5_CLNDAR
		0x00C	CH5_NLNDAR
		0x010	CH5_CLSDAR
		0x014	CH5_NLSDAR
		0x018	CH5_SAR/CH1_DAR
		0x01C	CH5_DLR
		0x020	CH5_CDLR
	CH6		
	0x7	0x000	CH6_CR
		0x004	CH6_SR
		0x008	CH6_CLNDAR
		0x00C	CH6_NLNDAR
		0x010	CH6_CLSDAR
		0x014	CH6_NLSDAR
		0x018	CH6_SAR/CH6_DAR
		0x01C	CH6_DLR
		0x020	CH6_CDLR
命令通道			
0x1	0x0	xxx	IN_FIFO，只写
	0x1	xxx	OUT_FIFO，只读
0x0	0x8	000	一拍高清除命令通道发出的可读标记
		004	命令通道 FIFO 的清除（一拍高）
		008	TCM 状态寄存器（寄存器说明详见表 C-1 ）

3.8.6 DMA 编程接口定义

详见 ICH 寄存器手册 TCM 部分

3.8.7 命令通道的基本工作流程

处理器发送一个新的命令包时。读取命令通道得状态寄存器，判断可写标记，），如果可 写（INFIFO 为空），直接发送最大数据量为 4KB 的数据包到命令缓存 INFIFO 中， 一旦 INFIFO 非空， OPENRISC 端的可读标记置为有效，而 OPENRISC 端则在读取到可读标记时， 开始读取 INFIFO 的数据，处理器数据的写入和 OPENRISC 端数据的读取是同时进行的。不管 是处理器还是 OPENRISC 端，都由软件通过读取状态寄存器来控制命令的收发状态转换，硬件 只是完成一个通道的功能。

处理器读取命令包的过程：

命令处理完毕，OPENRISC 端检测到可写标记有效（OUTFIFO 为非满），将应答包的数据 按照最大数据量为 4KB 的数据包写入到 OUTFIFO 中，写入完毕后，给处理器一个可读标记， 处理器接收到该标记后，将数据从 OUTFIFO 读出。OPENRISC 端不断的将应答包的数据写入 OUTFIFO。整个过程中，WB 总线在 FIFO 空满情况下，总线会不断 RTY，从而保证数据有效 传输，所以 OPENRISC 端没有复杂的数据流控制，可以实现 OPENRISC 端的全流水操作。

3.8.8 命令通道中断

TCM 向处理器端发两个中断：中断管理模块在 DMA 控制器模块中实现。

1) 命令应答中断：处理器接收到该中断开始读取 OUTFIFO 的应答包数据。
2) DMA 启动中断：处理器在与 TCM 命令交互过程中，如果要调用 DMA 方式的内部密码服 务，此时，TCM 端在解析完命令后给处理器发送一个应答中断，处理器开始启动 DMA 操 作调用内部密码服务。

3.8.9 命令通道编程接口定义

命令通道作为 AMBA 总线的从设备仅开放存储缓存空间和一个状态寄存器空间。 其中状态寄存器得具体说明如下表 c-1：

表 c-1

寄存器的值	含义
t_cmdfifo_st [1]=0	INFIFO 为非空，此时处理器不能写数据
t_cmdfifo_st [1]=1	INFIFO 为空，此时处理器可以往 INFIFO 写数据
t_cmdfifo_st [3]=0	数据尚未写入完毕，处理器不可读
t_cmdfifo_st [3]=1	数据写入完毕，处理器可读
t_cmdfifo_st [8]=0	OR1200 无效
t_cmdfifo_st [8]=1	OR1200 有效
t_cmdfifo_st [16]=0	BIOS 校验失败
t_cmdfifo_st [16]=1	BIOS 校验成功

3.9 BMC（维护控制模块）

3.9.1 CPU 可见的 BMC 地址空间

模块名	Func	地址空间	地址空间大小	实用	中断需求	Legacy IO 地址
XBUS	0	0x000000~0x0FFFFF	1MB	1MB	No	NVRAM
PS/2	1	0x100000~0x101FFF	8KB	8B	Yes	60h（键盘，鼠标）
	2	0x102000~0x103FFF	8KB	8B	Yes	64h（键盘，鼠标）
UART1	3	0x104000~0x105FFF	8KB	8B	Yes	3F8h – 3FFh (COM3)
UART2	4	0x106000~0x107FFF	8KB	8B	Yes	2F8h – 2FFh (COM2)
IPMI-KCS	5	0x108000~0x109FFF	8KB	4B	Yes	CA2h－CA3h
IPMI-BT	6	0x10A000~0x10BFFF	8KB	4B	Yes	E4~E6h
电源管理和复位端口	7	0x10C000~0x10DFFF	8KB	4B	Yes	80h，88h，8ch，90h
SPI 编程	8	0x10E000~0x10FFFF	8KB	1B	No	S_FLASH

1) LegacyIO 的低位地址偏移[12:0]在 BMC 内部编制时保持不变，对 BMC 来说，只要在上述

“对应 BMC 内部地址空间”的基址上加 LegacyIO 的 12 位偏移地址，来实现这些 IO 寄存 器编制即可，例如 PS2 的 LegacyIO 64h 在 BMC 内的编制为 0x1000-0064。这样既能兼容 软件的 MMIO 访问，又能兼容软件的 LegacyIO 访问，因为 PCIE-AMBA 桥来能够将 CPU 发出的 MMIO 访问或 LegacyIO 访问转换成相同的 AMBA Paddr[31:13]地址给 BMC。

2) IPMI 通信渠道

a) 包括 1 个 KCS 端口，1 个 BT 端口，对外呈现 3 个 Function；每个端口需要 8 字节 I/O 空间，需要支持字节访问。

b) 原则上 Host 不直接访问风扇、传感器等，而是通过与 BMC 通信间接访问。

3) Super I/O 功能

a) 包括 1 个 PS/2 模块，2 个 UART 串口模块，对外呈现 3 个 Function。 b)

每个模块需要 8 字节的 I/O 空间，需要支持字节访问。

4) 电源管理和复位端口

a) 用于 Host 发起关电、复位操作,也用于自检信息输出（有如 X86 系统中的 Port 80h）。

b) 对外呈现 1 个Function。 5)

扩展总线（XBUS）功能

a) 用于在外部扩展一片 FPGA，以应对 CPU 维护接口的变化，提高套片适应能力；

b) 用于外接 BMC 工作所需的高速存储器和外设；

c) 用于外接 NVRAM（数控项目需要，CPU 可见），此功能需要 Host 能访问挂在扩展总 线上的 NVRAM。

6) 与 SW-X 的中断接口

a) BMC 上述 7 个 SW-X 可见的中断，只需在 BMC 顶层给出 7 根中断连线即可（高电平 有效，硬件置，软件清）

b) 这 7 根中断线将连接的套片内部的中断处理模块进行进一步的处理，由它决定生成 INT 类还是 MSI 中断给处理器。

7) 对 CPU 发来的不存在的地址读写请求，BMC 会在 APB 总线上返回 ERROR 响应。

3.9.2 CPU 可见的寄存器及功能

3.9.2.1 复位初始化

复位初始化功能	BMC 内部地址	位宽	功能说明

80H	0x10C080	8	电源复位控制寄存器 0
88H	0x10C088	4	唤醒中断使能寄存器
8CH	0x10C08C	2	面板 LED 双色灯
90H	0x10C090	1	蜂鸣器发声使能

注：80H、88H、8CH、90H 这四个寄存器对处理器可见。

表：80H 寄存器域描述

名 称	范围	类型	描 述
CPU_RST_OP	[7:0]	RW,0x55	处理器复位控制

注：该寄存器的读写都只能采用字节读和字节写。

1） 软件通过对 80h 端口（BMC 内部地址为 0x4200-0080）的寄存器进行操作，来实现对 BMC 复位的控制。

2） 处理器在 80h 端口写入 0xff【与 off 形似】后，会形成中断给 OR1200，OR1200 处理中断时读取 80h 中的值，如果为 0xff，执行关机操作；SW 写入 0x00【从头开始】后，会形成 中断给 OR1200，OR1200 处理中断时读取 80h 中的值，如果为 0x00，执行系统复位操作。 SW 写入 0xEE【与 SLEEP 形似】后，且唤醒中断有效，也会形成中断给 OR1200，OR1200 读取 80h 中的值，如果为 0xEE，执行唤醒操作（由 BMC 给逐个核心发唤醒中断）。

3） 这三种操作都通过中断方式，通知 OR1200 来进行处理，OR1200 读取 80h 端口来决定具体 的操作类型。待机复位后，80h 端口值为 0x55。

4） 睡眠唤醒能够通过设置 88H 端口的唤醒中断使能寄存器来设置那些设备可以实现唤醒操作。

表：88H 寄存器域描述

名 称	范围	类型	描 述
—	[31:7]	—	保留。
WAKEUP_INT_MASK	[6:0]	RW,0x3F	唤醒中断使能寄存器

注：1）需采用 32 位读写操作，对处理器可见。

2）各使能位都是高有效，为低时对应中断被屏蔽

表：8CH 寄存器域描述

名 称	范围	类型	描 述
—	[31:2]	—	保留。
PWR_LED	[1:0]	RW,0x0	面板双色 LED 灯

注：需采用 32 位读写操作，对处理器可见。

表：90H 寄存器域描述

名 称	范围	类型	描 述
—	[31:1]	—	保留。
BEEP_EN	[0]	RW,0x0	唤醒中断使能寄存器

注：需采用 32 位读写操作，对 SW-X 可见。

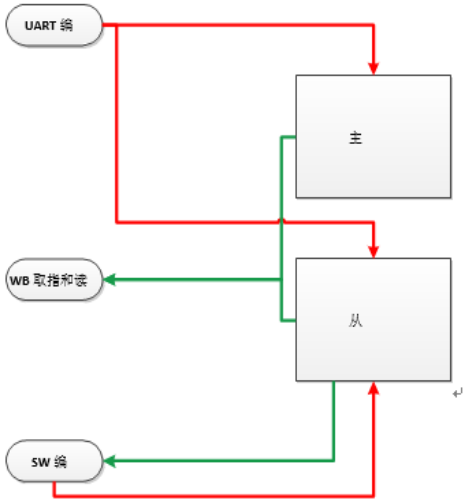
接口信号	信号说明	屏蔽位
Gmac0_pmt_int_o	GMAC 控制器 0 电源唤醒信号，高电平有效	0
Gmac1_pmt_int_o	GMAC 控制器 1 电源唤醒信号，高电平有效	1
OHCI02INT_int_n	USB 键盘鼠标(OHCI0)的中断请求线，低电平有效。	2
OHCI12INT_int_n	USB 键盘鼠标(OHCI1)的中断请求线，低电平有效。	3
KEY2INT_int	BMC 的 PS2(Key)中断请求线，高电平有效。	4
MOUSE2INT_int	BMC 的 PS2(mouse)中断请求线，高电平有效。	5
PMBUG2INT_int	电源唤醒中断事件	6

3.9.2.2 FLASH 编程接口

套片 BMC 支持主从双 FLASH，在 BMC 地址空间的编址如下：

模块名	地址	中断需求
SPI-FROM-MASTER	0000-0000~00FF-FFFF	No
SPI-FROM-SLAVER	0100-0000~01FF-FFFF	No

注：现阶段采用的 FLASH 芯片地址位宽只有 24 位，容量为 4MB~8MB，所以软件访问从 FLASH 时，地址 [24]必须为 1。两片 FLASH 的 SPI 的编程和取值访问通路如下：



注：红色为编程，绿色为读数。

使用主板上的跳线信号 UART_HDPLX 来区分各种不同的编程和访问方式，当 UART_HDPLX 为低有效时，只允许 UART 接口对两个 FLASH 进行编程，不能进行其它操作（此时通常是主板调试初期）。当 UART_HDPLX 为高无效时，UART 编程禁止，此时设置为 OR1200 能够对主从两片 FLASH 进行读访问（包括取值），SW-X 只能对从 FLASH 进行编程读写；此时 **需要避开 OR1200 和 SW-X 都对从 FLASH 进行操作（OR1200 读，SW-X 进行编程）**。

串口能对主从两片 FLASH 进行编程，而 CPU 只能对第二片从 FLASH 进行编程操作，两者采用同样的命令包格式和协议，只是接口方式上有差别，命令包格式如下：

字节序号	字节内容	
1	写长度（WLEN）	命令包
2	读长度（RLEN）	
3	写数据 1	
…		
W	写数据 W（最后一个）	
W+1		回的响应包
…		
W+R		

FLASH 读写长度编码规则如下

1） **读写长度的高 4 位（RLEN[7:4]或 WLEN[7:4]）为 2 的指数：**

0~3 —— 保留，视为 0;

4 —— 2^4，长度为 16;

5 —— 2^5，长度为 32;

6 —— 2^6，长度为 64;

7 —— 2^7，长度为 128;

8 —— 2^8，长度为 256;

9~15—— 如果只有一片 FLASH，这些值保留，视为 0;

要访问第 2 颗 FlashROM 芯片（即嵌入式 CPU 看到的 8MB~16MB 地址空间，Addr[23]=1），在读写长度信息中做如下规定：

操作类型	WLEN		RLEN		备注
	[7:4]	[3:0]	[7:4]	[3:0]	
对主 FLASH 的读	<=8	任意值	<=8	任意值	
对主 FLASH 的写	<=8	任意值	<=8	任意值	
对从 FLASH 的读	**0xF**	任意值	<=8	任意值	0XF 作为访问从 FLASH 的标志
对从 FLASH 的写	<=8	任意值	**0xF**	任意值	0XF 作为访问从 FLASH 的标志

注：

1） 软件组命令包时每次只能发读命令包或写命令包，不会有同一个命令包内既写 256 字节， 又读 256 字节的要求，通常只做读（命令和地址的长度要算在 WLEN 内），或只做写。

2） 命令包内的地址只有 24 位有效，所以这也必须要求使用上表的规范才能读写第二片 FLASH。

2） 读写长度的低 4 位（RLEN[3:0]或 WLEN[3:0]）为正常的数值：

0 ——0;

1 ——1;

…

15 ——15。

注意：总的长度为高 4 位表示的长度和低 4 位标识的长度之和。暂时只支持最大 256+15 字节的命令包或响应包。

写数据和读数据长度均为 0 的命令包为 NOP 命令。出现混乱时，可用这种包来重建秩序。

PRG_SPI	BMC 内部地址	位宽	功能说明
PRG_BUF	0x10E000	8	计时器控制寄存器
PRG_STAT	0x10E004	1	是否正在编程

表：SPI_BUF 寄存器域描述

名 称	范围	类型	描 述

	[31:8]		保留
SPI_BUF	[7:0]	RW,0x0	SPI 命令 BUFFER 读写接口

PRG_STAT 寄存器域描述

名 称	范围	类型	描 述
	[31:1]		保留
PRG_STAT	[0]	RW,0x0	SPI 接口是否正在编程

增加了从 SPI 接口(wb 请求)过来的对 FLASH 的烧写逻辑。SW-X 和 OR1200 可以通过字节写 0x10E000 地址的 SPI_BUF 寄存器来生成 FLASH 擦除和编程的命令包（编程最大 256 字节）， 命令包格式与 UART 接口编程 FLASH 相同。读 FLASH 状态和数据时，也通过字节写 0x10E000（命令 BUFFER 地址）来组读命令包，读出的数据（长度为 WLEN）放在读缓冲内（最大 256 字节），软件通过字节读 0x10E000（读数据 BUFFER 地址）来得到读数据。

建议采用 Fast Read Data（0Bh）命令来读取 FLASH 中的数据，因为普通的 Read Data（03h） 命令受器件 50MHz 频率的限制，用 Fast Read Data 命令时，接收到的第一个字节无效（DummyClocks 时的数据）。

3.10 I2C 接口

I2C 接口有以下特性：

- 两线的 I2C 串行接口，包括串行数据总线（SDA）和串行时钟（SCL）
- 支持两种速率
  - Standard mode (100 Kb/s)
  - Fast mode (400 Kb/s)
- 时钟同步
- Master 或 Slave 的 I2C 操作
- 7 或 10 位地址
- 7 或 10 位地址组合格式的传输
- 批量传输模式（Bulk transmit）
- 忽略 CBUS 地址（CBUS 是 I2C 总线的老版本，可以共享 I2C 总线）
- 接收和发送缓冲
- 中断或 polled-mode 操作
- 可处理所有总线速率的 Bit 或 Byte 的等待
- 软件接口简单，系统接口为 APB 总线

- 支持由软件配置参数
- 和 DW_ahb_dmac 控制器有 DMA 握手接口
- 数据（SDA）保持时间可编程（tHD;DAT） 寄存器详细说明请参考套片寄存器手册。