

# 第一章 S3C6410 处理器概述

S3C6410 是一个 16/32 位 RISC 微处理器，旨在提供一个具有成本效益、功耗低，性能高的应用处理器解决方案，像移动电话和一般的应用。它为 2.5G 和 3G 通信服务提供优化的 H/W 性能，S3C6410 采用了 64/32 位内部总线架构。该 64/32 位内部总线结构由 AXI、AHB 和 APB 总线组成。它还包括许多强大的硬件加速器，像视频处理，音频处理，二维图形，显示操作和缩放。一个集成的多格式编解码器（MFC）支持 MPEG4/H.263/H.264 编码、译码以及 VC1 的解码。这个 H/W 编码器/解码器支持实时视频会议和 NTSC、PAL 模式的 TV 输出。

S3C6410 有一个优化的接口连线到外部存储器。存储器系统具有双重外部存储器端口、DRAM 和 FLASH/ROM/DRAM 端口。DRAM 的端口可以配置为支持移动 DDR，DDR，移动 SDRAM 和 SDRAM。FLASH/ROM/DRAM 端口支持 NOR-FLASH，NAND-FLASH，ONENAND，CF，ROM 类型外部存储器和移动 DDR，DDR，移动 SDRAM 和 SDRAM。

为减少系统总成本和提高整体功能，S3C6410 包括许多硬件外设，如一个相机接口，TFT 24 位真彩色液晶显示控制器，系统管理器（电源管理等），4 通道 UART，32 通道 DMA，4 通道定时器，通用的 I/O 端口，IIS 总线接口，IIC 总线接口，USB 主设备，在高速（480 MB/S）时 USB OTG 操作，SD 主设备和高速多媒体卡接口、用于产生时钟的 PLL。

S3C6410 提供了丰富的内部设备，下面我们从它的整体特性、多媒体加速特性、视频接口、USB 特征、存储器设备、系统外设以及它的系统管理等方面来详细的介绍 S3C6410 处理器的特性：

## 1.1. S3C6410 体系结构

S3C6410 RISC 处理器特性包括：

- （1）基于 CPU 的子系统的 ARM1176JZF-S 具有 JAVA 加速引擎和 16KB/16KB I/D 缓存和 16KB/16KB I/D TCM。
- （2）在各自地 TBD V 和 TBD V 的 400/533/667MHz 操作频率。
- （3）一个 8 位 ITU 601/656 相机接口，用于缩放的高达 4M 像素，固定的 16M 像素。
- （4）多标准编解码器提供的 MPEG-4/H.263/H.264 编码和解码的高达 30 帧/s，VC1 视频解码、达到

30 帧/s。

- (5) 具有 BITBLIT 和轮换的 2D 图形加速。
- (6) AC-97 音频编解码器接口和 PCM 串行音频接口。
- (7) IIS 和 IIC 接口支持。
- (8) 专用的 IRDA 端口，用于 FIR，MIR 和 SIR。
- (9) 灵活配置 GPIO 。
- (10) 端口 USB 2.0 OTG 支持高速（480 MBPS，片上收发器）。
- (11) 端口 USB 1.1 主设备支持全速（12 MBPS，片上收发器）。
- (12) 高速 MMC / SD 卡支持。
- (13) 实时时钟，锁相环，具有 PWM 的定时器和看门狗定时器。
- (14) 32 通道 DMA 控制器。
- (15) 支持 8X8 键盘矩阵变换电路。
- (16) 用于移动应用的先进的电源管理。
- (17) 存储器子系统
  - 具有 8 倍或 16 倍数据总线的 SRAM/ROM/NOR Flash 接口。
  - 具有 16 倍数据总线的 MUXED，ONENAND 接口。
  - 具有 8 倍数据总线的 NANDFlash 接口。
  - 具有 16 倍或 32 倍数据总线的 SDRAM 接口。
  - 具有 16 倍或 32 倍数据总线（133Mb/s/引脚率）的移动 SDRAM 接口。
  - 具有 16 倍或 32 倍数据总线（266 Mb/s/引脚 DDR）的移动 DDR 接口。

## 2. ARM1176JZF-S 处理器

ARM1176JZF-S 处理器的特性包括：

- (1) TrustZone™安全扩展。
- (2) 具有超高速先进的微处理器总线架构（AMBA）、先进的可扩展接口（AXI）电平，两个接口支持的优先级顺序多处理机。
- (3) 8 阶管线。
- (4) 具有返回堆栈的分支预测。
- (5) 低中断延时配置。
- (6) 外部协处理器接口和协处理器 CP14 和 CP15 。

- (7) 指令和数据存储器管理单元 (MMUS)，通过一个统一的主 TLB 使用 MICROTTLB 结构管理。
- (8) 实际地索引和物理地址缓存。
- (9) 矢量浮点型 (VFP) 协处理器支持。
- (10) 外部协处理器的支持。
- (11) 追踪支持。

存储器子系统包括：

- (1) 高频宽存储器矩阵变换电路子系统。
- (2) 两个独立的外部存储器端口（一个静态混合的 DRAM 存储器端口和一个 DRAM 端口）。
- (3) 矩阵变换电路架构增加整体的带宽，具有同时访问的能力

### 3. 多媒体加速特性

多媒体加速特性包括：

#### (1) 照相机接口

- 支持 ITU - R 601/ITU-R 656 格式输入。支持 8 位输入。
- 对于 YCBCr 4:2:2 格式，相机输入分辨率高达  $4096 \times 4096$ 。
- $4096 \times 4096$  输入分辨率采取绕过硬件缩小尺度和预览单元，并且图象将以 JPEG 格式直接存储到存储器。
- 高达  $2048 \times 2048$  输入分辨率可以选择性的输入到硬件缩小尺度单元和预览单元。
- 分辨率缩小尺度，硬件支持的输入分辨率高达  $2048 \times 2048$ 。
- 编解码器/预览输出图像产生（16/18/24 位的 RGB 格式和 YCbCr 4: 2: 0/ 4:2:2 格式）。
- 图象窗口化和变焦的功能。
- 测试图案产生。
- 图像镜像和轮换支持 Y 轴镜像和 X 轴镜像，90 度、180 度和 270 度的轮换。
- H/W 色彩空间的转换。
- 支持 LCD 控制器直通。

#### (2) 多标准解码器 (MSC)

##### ①多标准视频编解码器

- MPEG-4 部分 II 简单协议规范编码/解码。
- H. 264/AVC 基线编码/解码。
- H. 263 协议规范 3 编码/解码。

- VC1 解码。
- 支持多部分电池和多标准。

#### ②编码工具

- 可变模块大小：16×16，16×8，8×16 和 8×8 。
- 自由的运动矢量。
- MPEG - 4 AC / DC 预测。
- H. 264/AVC 的帧内预测（固定模式决定）。
- 错误恢复工具。
- MPEG - 4 重新同步。具有 RVLC 的标记和数据分割。
- MPEG-4/AVC FMO 和 ASO。
- 位率控制（CBR 和 VBR）。

#### ③解码工具

支持所有标准功能。

#### ④前/后旋转/镜像

八个镜像/旋转模式。

#### ⑤性能

- 全双工的 VGA 30 fps 编码/解码。
- 半双工 720×480 30 帧/s（720×576 25f 帧/s）编码/解码。

#### （3）JPEG 解码器

- 压缩/解压缩达 65536×65536。
- 编码格式：YCbCr 4: 2: 2。
- 解码格式：YCbCr 4: 4: 4/ 4:2:2 / 4: 2: 0/ 4: 1: 1 或灰色。
- 支持压缩的内存数据在 YCbCr 4:2:2 或 RGB 565 格式。
- 支持一般用途的时钟转换器。

### 4. 显示控制

显示控制特性包括：

#### （1）TFT LCD 接口

- 320×240，640×480 或其他显示分辨率高达 1024×1024。
- 最大 2k × 2k 虚拟屏幕尺寸。



- 
- 支持五个窗口层作为 PIP 或 OSD。
  - 可编程 OSC 窗口定位。
  - 16 级 Alpha 混合。

#### (2) 视频后处理器

- 视频输入格式转换。
- 视频/图形缩放向上/向下或缩放输入/输出。
- 彩色空间的转换，从 YCbCr 到 RGB 和从 RGB 到 YCbCr。
- 专用本地接口显示。
- 专用定标器用作 TV 编码器。

#### (3) 具有图像增强的 TV (NTSC/PAL) 视频编码器

- ①支持 NTSC-M/PAL-B, D, G, H, I 兼容视频格式。
- ②支持 YCbCr 4: 2: 0/ 4:2:2 , 16/18/24 位 RGB 源格式。
- ③内置 MIE (移动图像增强器) 引擎
  - 黑色和白色延展。
  - 蓝色延展和 Flesh-Tone 校正。
  - 动态水平的尖峰与 LTI。
  - 黑色与白色噪音的降低。
  - 原始的, 全屏和宽屏视频输出。

## 5. 视频接口

视频接口特性包括:

#### (1) AC97 音频编解码器接口

- 可变采样率 ( 48 kHz 和低于 )。
- 1 通道立体声输入/1 通道立体声输出/1 通道麦克风输入。
- 16 位立体声 (2 声道) 音频。

#### (2) PCM 串行音频接口

- 主模式双向串行音频接口。
- 接受一个外部输入时钟来产生精确的音频时间。
- 可选的基于 DMA 的操作。

#### (3) IIS 总线立体声 DAC 接口

- 
- 1 通道总线作为音频编解码器接口。
  - 可选的基于 DMA 的操作。
  - 串行，每通道 8/16 位的数据传输。
  - 支持 IIS，合理的 MSB 和合理的 LSB 数据格式。
  - 可以在主或从模式下操作。
  - 支持多种位时钟频率和编解码器的时钟频率。
  - 16, 24, 32, 48fs 的位时钟频率和 256, 384, 512, 768fs 的编解码器的时钟频率。

## 6. USB 特性

USB 支持特性包括：

### (1) USB OTG2.0 高速

- 符合 OTG 规格 1.0 版本补充的 USB 2.0 协议的 2.0 版本。
- 配置只作为 OTG 设备，USB 1.1 设备，OTG 迷你主设备，或 USB 1.1 迷你主设备。
- 支持高速（480 Mb/s），全速（12 Mb/s）和低速（1.5Mb/s）。

### (2) USB 主设备

- 两个端口 USB 主设备。
- 符合 OHCI 1.0 版本。
- 符合 USB 规范 1.1 版本。
- 支持全速高达 12 Mb/s。

## 7. IrDA v1.1

IrDA v1.1 特性包括：

- (1) 专用的 IrDA 作为 v1.1（1.152Mb/s 和 4 Mb/s）。
- (2) 支持 FIR（4Mb/s）。
- (3) SIR（111.5kb/s）模式是由 UART 的 IrDA 1.0 模块支持的。
- (4) 内部 64 字节的 Tx/Rx FIFO。

串行通行特性：

### (1) UART

- 4 通道 UART 具有基于 DMA 或基于中断操作。
- 支持 5 位，6 位，7 位，或 8 位串行数据传输/接收。

- 
- 支持外部时钟用作 UART 操作（UCLK）。
  - 可编程波特率。
  - 支持 IrDA 1.0 SIR（115.2kb/s）模式。
  - 环回模式进行测试。
  - 每个通道都有内部 64 字节的 Tx FIFO 和 64 字节的 Rx FIFO。

#### （2）IIC 总线接口

- 1 通道多主设备 IIC 总线。
- 串行，8 位针对性和双向数据传输可在高达 100 kb/s 的标准模式下操作。
- 在快速模式高达 400 kb/s。

#### （3）SPI 接口

- 2 通道串行外设接口。
- 64 字节缓冲器用来接收/传送。
- 基于 DMA 或基于中断操作。
- 50Mb/s 的发送/接收（全双工）。

#### （4）MIPI HSI

- 单向高速串行接口。
- 支持发送和接收。
- 128 字节（32 位× 32）Tx FIFO。
- 256 字节（32 位× 64）RX FIFO。
- 发送：PCLK b/s，接收：高达 100 Mb/s。

### 8. 调制解调器接口

调制解调器接口特性包括：

并行调制解调器芯片接口

- 异步直接和间接 16 位 SRAM 式接口（i80 接口）。
- 片上 8KB 的双端口 SRAM 缓冲区直接接口。
- 片上写 FIFO 和读 FIFO（每 288 字），以支持间接脉冲数据传输。

---

## 9. GPIO

GPIO 特性包括：

188 个灵活配置的 GPIO。

输入设备特性：

(1) 便携式键盘接口

- 支持 8×8 键盘矩阵转换电路。
- 提供内部去抖滤波器。

(2) A/D 转换和触摸屏接口

- 8 通道复用 ADC。
- 最大 500k 采样/S 和 10 位分辨率。

## 10. 存储器设备

存储器设备特性包括：

MMC/SD 主设备

- 兼容多媒体卡协议版本 4.0。
- 兼容 SD 存储卡的协议版本 1.0。
- 128 字 FIFO 用作发送/接收。
- 基于 DMA 或基于中断操作。

## 11. 系统外设

系统外设特性包括：

(1) DMA 控制器

- 四个通用 DMA 嵌入式。
- 每个 DMA 有两个主端口。
- 每一个 DMA 支持 8 通道；完全支持 32 通道。
- 支持存储器到存储器，外设到存储器，存储器到外设，和外设到外设。
- 脉冲数据传输模式，以提高传输速率。

(2) 矢量中断控制器

- 支持 32 个矢量 IRQ 中断。
- 固定硬件中断优先级。

- 
- 可编程中断优先级。
  - 硬件中断优先级屏蔽。
  - IRQ 和 FIQ 生成。
  - 测试寄存器。
  - 原始中断状态。
  - 中断请求状态。
  - 支持 ARM v6 处理器 VIC 端口，在同步和异步模式，使其更快地中断服务。

#### (3) TrusZone 中断控制

- 在 TrustZone 设计中，提供了一个软件接口给安全中断系统的保护位。
- 在安全控制下，从任何系统中断源 nFIQ 生成。
- 从非安全中断控制器屏蔽的选择 nFIQ 中断。

#### (4) TrusZone 保护控制器

- 在 TrustZone 设计中，在一个安全的系统提供一个软件接口到保护位。
- AMBA APB 接口。

#### (5) 具有 PWM 的定时器（脉宽调制）

- 具有 PWM 的 4 通道 32 位定时器。
- 具有基于 DMA 或基于中断操作的 1 通道 32 位内部定时器。
- 可编程占空比周期，频率和极性。
- 死区生成。
- 支持外部时钟源。

#### (6) 16 位看门狗定时器

在超时时中断请求或系统复位。

#### (7) RTC（实时时钟）

- 完全时钟特性：毫秒，秒，分，时，天，星期，月，年。
- 32.768kHz 操作。
- 报警中断。
- 时间节拍中断。

12. 系统管理

系统管理特性包括：

系统操作频率

- ARM1176JZF - S 核心时钟频率最高是 667 MHz。
- 系统操作时钟产生。
- 三个片上 PLL ， APLL， MPLL 和 EPLL。
- APLL 生成一个独立 ARM 操作时钟。
- MPLL 生成系统参考时钟。
- EPLL 产生用作外设 IP 的时钟。

1.2 S3C6410 的引脚名称

为了能清楚地描述 S3C6410 的引脚信号，下面将先根据 S3C6410 的引脚定义图，详细的介绍各个引脚的标号与定义。

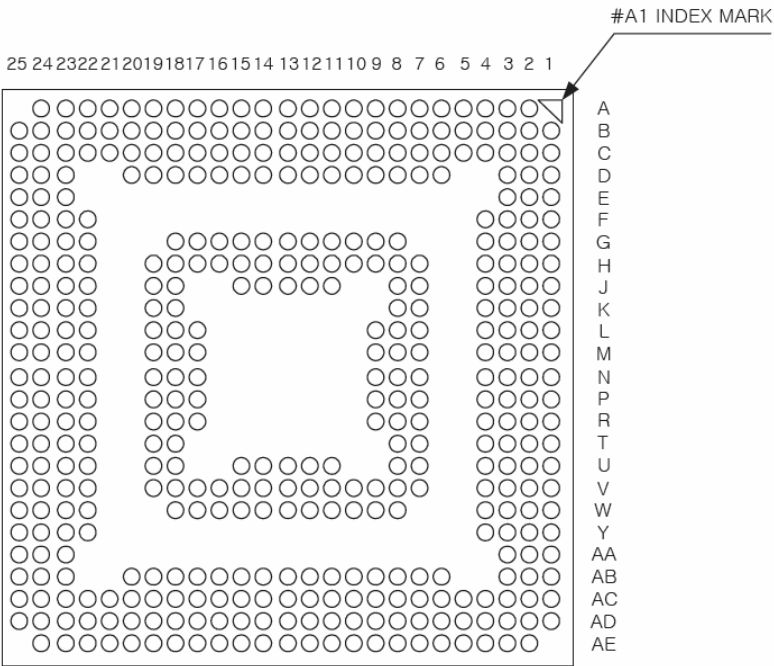


图 1-1 S3C6410 引脚图

按照图 1-1 所示的引脚顺序，各个引脚名称如表 1-1 所示。

表 1-1 引脚名称

引脚号	引脚名称	引脚号	引脚名称	引脚号	引脚名称
A2	NC_C	B3	XPCMEXTCLK1/GPE1	C3	XPCMSOUT1/GPE4
A3	XPCMSOUT0/GPD4	B4	XPCMSIN0/GPD3	C4	XPCMFSYNC1/GPE2
A4	VDDPCM	B5	XPCMEXTCLK0/GPD1	C5	XPCMDCLK1/GPE0
A5	XM1DQM0	B6	XM1DATA0	C6	XM1DATA4
A6	XM1DATA1	B7	XM1DATA3	C7	XM1DATA2
A7	VDDI	B8	VDDM1	C8	XM1DATA5
A8	VDDARM	B9	VDDM1	C9	XM1DATA7
A9	XM1DATA6	B10	XM1DATA13	C10	VDDARM
A10	XM1DATA9	B11	VDDARM	C11	XM1DATA14
A11	XM1DATA12	B12	XM1DATA16	C12	XM1DATA10
A12	XM1DATA18	B13	XM1DATA17	C13	XM1DATA19
A13	XM1SCLK	B14	XM1DQS2	C14	VDDM1
A14	XM1SCLKN	B15	XM1DATA22	C15	XM1DATA20
A15	XMMCDATA1_4/GHP6	B16	XMMCDATA1_2/GPH4	C16	XMMCDATA1_6/GPH8
A16	XMMCCMD1/GPH1	B17	VDDMMC	C17	XMMCDATA1_1/GPH3
A17	XMMCCDN0/GPG6	B18	XMMCDATA0_0/GPG2	C18	XMMCDATA0_2/GPG4
A18	XMMCCLK0/GPG0	B19	XSPIMISO1/GPC4	C19	XSPIMOSI1/GPC6
A19	XSPIMOSI0/GPC2	B20	XSPIMISO0/GPC0	C20	XSPICS0/GPC3
A20	XI2CSCL/GPC8	B21	XUTXD3/GPB3	C21	VDDEXT
A21	XUTXD2/GPB1	B22	XUTXD1/GPA5	C22	XURTSN1/GPA7
A22	XURTSN0/GPA3	B23	XCIYDATA7/GPF12	C23	XPWMECLK/GPF13
A23	XUTXD0/GPA1	B24	XCIYDATA5/GPF10	C24	XCIYDATA2/GPF7
A24	NC_D	B25	NC_F	C25	XCIYDATA0/GPF5
B1	NC_B	C1	XM0ADDR0	D1	XM0ADDR2
B2	XPCMSIN1/GPE3	C2	VDDARM	D2	XM0ADDR3

表 1-1 引脚名称

D3	VDDARM	F1	XM0ADDR8/GPO8	G24	XM1DATA27
D6	XPCMFSYNC0/GPD2	F2	XM0ADDR6/GPO6	G25	XM1DATA30
D7	XPCMDCLK0/GPD0	F3	VDDARM	H1	VDDI
D8	VDDARM	F4	VDDM0	H2	XM0ADDR13/GPO13
D9	XM1DQS0	F22	XCIPCLK/GPF2	H3	XM0ADDR15/GPO15
D10	XM1DATA15	F23	XM1DATA24	H4	XM0ADDR12/GPO12
D11	XM1DATA11	F24	XM1DATA25	H7	XM0ADDR4
D12	XM1DATA8	F25	XM1DATA26	H8	VSSIP
D13	VDDI	G1	XM0ADDR11/GPO11	H9	XMMCDATA1_7/GPH9
D14	XM1DQM2	G2	XM0ADDR10/GPO10	H10	XMMCDATA1_3/GPH5
D15	XM1DATA21	G3	VDDM0	H11	XMMCDATA1_0/GPH2
D16	XM1DATA23	G4	XM0ADDR7/GPO7	H12	XSPICLK1/GPC5
D17	XSPICS1/GPC7	G8	XM1DQM1	H13	XMMCDATA0_1/GPG3
D18	VDDI	G9	XM1DQS1	H14	XSPICLK0/GPC1
D19	XURXD2/GPB0	G10	VDDM1	H15	XUCTSN1/GPA6
D20	XURXD0/GPA0	G11	XMMCDATA1_5/GPH7	H16	XPWMTOUT0/GPF14
D23	XPWMTOUT1/GPF15	G12	XMMCDATA0_3/GPG5	H17	XCIYDATA4/GPF9
D24	XCIVSYNC/GPF4	G13	XMMCCMD0/GPG1	H18	VSSPERI
D25	XCIHREF/GPF1	G14	XI2CSDA/GPC9	H19	XCIRSTN/GPF3
E1	XM0ADDR5	G15	XIRSDBW/GPB4	H22	XM1DQM3
E2	VDDARM	G16	XUCTSN0/GPA2	H23	XM1DATA31
E3	XM0ADDR1	G17	XCIYDATA6/GPF11	H24	XM1ADDR0
E23	XCIYDATA1/GPF6	G18	XCIYDATA3/GPF8	H25	XM1ADDR3
E24	XM1DATA28	G22	XCICLK/GPF0	J1	XM0AP/GPQ8
E25	XM1DQS3	G23	XM1DATA29	J2	XM0WEN



表 1-1 引脚名称

J3	VDDARM	K24	XM1ADDR12	M19	XM1WEN
J4	XM0ADDR14/GPO14	K25	XM1ADDR5	M22	VDDI
J7	VSSMEM	L1	XM0DQM0	M23	XM1ADDR10
J8	XM0ADDR9/GPO9	L2	XM0DATA13	M24	XM1CKE1
J11	XMMCCLK1/GPH0	L3	XM0SMCLK/GPp1	M25	XHIDATA17/GPL14
J12	VSSIP	L4	XM0OEN	N1	XM0DATA1
J13	VSSPERI	L7	XM0DATA10	N2	XM0DATA0
J14	XURXD3/GPB2	L8	XM0DATA12	N3	XM0DATA3
J15	XURXD1/GPA4	L9	VSSIP	N4	XM0DATA6
J18	VDDI	L17	VDDI	N7	XM0CSN0
J19	VDDM1	L18	XM1CSN1	N8	XM0CSN5/GPO3
J22	XM1ADDR9	L19	XM1ADDR4	N9	VSSIP
J23	XM1ADDR2	L22	XM1RASN	N17	XHIDATA16/GPL13
J24	XM1ADDR1	L23	XM1CSN0	N18	XHIDATA14/GPK14
J25	XM1ADDR6	L24	XM1CASN	N19	VDDUH
K1	XM0DATA15	L25	XM1ADDR15	N22	XUHDP
K2	VDDM0	M1	VDDM0	N23	XHIDATA15/GPK15
K3	VDDARM	M2	XM0DATA8	N24	XHIDATA13/GPK13
K4	XM0DATA14	M3	XM0DATA11	N25	XHIDATA12/GPK12
K7	XM0DQM1	M4	XM0DATA9	P1	VDDI
K8	VSSIP	M7	XM0DATA2	P2	XM0DATA5
K18	XM1ADDR7	M8	XM0DATA4	P3	XM0DATA7
K19	XM1ADDR11	M9	VSSMEM	P4	XM0CSN2/GPO0
K22	XM1ADDR13	M17	XM1ADDR14	P7	XM0CSN7/GPO5
K23	XM1ADDR8	M18	XM1CKE0	P8	XM0CASN/GPQ1

表 1-1 引脚名称

P9	VSSMEM	T4	XM0DQS0/GPQ5	U25	XHIADR8/GPL8
P17	VSSIP	T7	XEFFVDD	V1	VDDM0
P18	XHIDATA11/GPK11	T8	VSSMPLL	V2	XM0DQS1/GPQ6
P19	XHIDATA9/GPK9	T18	XHIADR7/GPL7	V3	XM0CKE/GPQ4
P22	XUHDN	T19	XHIADR9/GPL9	V4	XM0WEATA/GPP12
P23	XHIDATA10/GPK10	T22	XHIDATA1/GPK1	V7	VSSEPLL
P24	VDDHI	T23	XHIDATA3/GPK3	V8	XOM3
P25	XHIDATA8/GPK8	T24	XHIDATA2/GPK2	V9	XNRESET
R1	VDDM0	T25	XHIDATA0/GPK0	V10	XEINT1/GPN1
R2	XM0CSN3/GPO1	U1	XM0SCLKN/GPQ3	V11	XEINT6/GPN6
R3	XM0CSN1	U2	XM0RASN/GPQ0	V12	XEINT12/GPN12
R4	XM0WAITN/GPP2	U3	XM0WENDMC/GPQ7	V13	XVVD3/GPI3
R7	XM0INTATA/GPP8	U4	XM0INTSM1_FREN/GPP6	V14	XVVD8/GPI8
R8	XM0RDY0_ALE/GPP3	U7	XM0CDATA/GPP14	V15	XVVD12/GPI12
R9	VSSIP	U8	VSSMEM	V16	XVVD16/GPJ0
R17	VSSPERI	U11	VSSPERI	V17	VSSPERI
R18	VDDALIVE	U12	VSSPERI	V18	XHICSN_MAIN/GPM1
R19	XHIADR12/GPL12	U13	VSSIP	V19	XVCLK/GPJ11
R22	XHIDATA5/GPK5	U14	VSSPERI	V22	XHIOEN/GPM4
R23	XHIDATA4/GPK4	U15	VDDALIVE	V23	XHIADR6/GPL6
R24	XHIDATA6/GPK6	U18	XHIADR2/GPL2	V24	VDDHI
R25	XHIDATA7/GPK7	U19	XHIADR0/GPL0	V25	XHIADR5/GPL5
T1	XM0SCLK/GPQ2	U22	XHIADR4/GPL4	W1	VDDI
T2	XM0CSN6/GPO4	U23	XHIADR11/GPL11	W2	XM0RDY1_CLE/GPP4
T3	XM0CSN4/GPO2	U24	XHIADR10/GPL10	W3	XM0RESETATA/GPP9

---

W4	VSSAPLL	AA2	XM0INPACKATA/GPP10	AB25	XVVD20/GPJ4
W8	VSSMEM	AA3	XM0REGATA/GPP11	AC1	XADCAIN0
W9	XOM1	AA23	XHICSN/GPM0	AC2	XADCAIN1
W10	VDDALIVE	AA24	XVDEN/GPJ10	AC3	XADCAIN7
W11	XEXTCLK	AA25	XVHSYNC/GPJ8	AC4	VDDADC
W12	XEINT8/GPN8	AB1	VDDEPLL	AC5	VSSDAC
W13	XEINT14/GPN14	AB2	VDDMPLL	AC6	XDACOUT0
W14	XVVD1/GPI1	AB3	XM0OEATA/GPP13	AC7	XDACCOMP
W15	XVVD6/GPI6	AB6	VSSMEM	AC8	XUSBREXT
W16	XVVD11/GPI11	AB7	VSSOTG	AC9	VDDOTG
W17	XVVD14/GPI14	AB8	VSSOTGI	AC10	VDDOTGI
W18	XVVD22/GPJ6	AB9	XRTCXTI	AC11	VDDRTC
W22	XVVSYSYNC/GPJ9	AB10	XJTRSTN	AC12	XJTDO
W23	XHIADR3/GPL3	AB11	XJTCK	AC13	XOM2
W24	XHIADR1/GPL1	AB12	XJTDI	AC14	VSSPERI
W25	XHIIRQN/GPM5	AB13	XJDBGSEL	AC15	VDDSYS
Y1	XM0RPN_RNB/GPP7	AB14	XXTO27	AC16	XXTI
Y2	XM0ADRVAlID/GPP0	AB15	XXTI27	AC17	XXTO
Y3	XM0INTSM0_FWEN/GPP5	AB16	XSELNAND	AC18	XEINT5/GPN5
Y4	XPLLEFILTER	AB17	XEINT3/GPN3	AC19	XEINT7/GPN7
Y22	XVVD18/GPJ2	AB18	XEINT10/GPN10	AC20	VDDI
Y23	XHIWEN/GPM3	AB19	VDDALIVE	AC21	XVVD9/GPI9
Y24	XHICSN_SUB/GPM2	AB20	XVVD5/GPI5	AC22	XVVD10/GPI10
Y25	VDDI	AB23	XVVD23/GPJ7	AC23	VDDLCD
AA1	VDDAPLL	AB24	XVVD21/GPJ5	AC24	XVVD15/GPI15

表 1-1 引脚名称

AC25	XVVD19/GPJ3	AD25	NC_J		
AD1	NC_G	AE2	NC_H		
AD2	XADCAIN2	AE3	XADCAIN4		
AD3	XADCAIN3	AE4	XADCAIN6		
AD4	XADCAIN5	AE5	XDACOUT1		
AD5	VSSADC	AE6	XDACIREF		
AD6	VDDDAC	AE7	XDACVREF		
AD7	XUSBXTI	AE8	VSSOTG		
AD8	XUSBXTO	AE9	XUSBDM		
AD9	XUSBVBUS	AE10	XUSBDP		
AD10	XUSBID	AE11	XUSBDRVVBUS		
AD11	VDDOTG	AE12	XJTMS		
AD12	XRTCXTO	AE13	XJRTCK		
AD13	XOM0	AE14	XOM4		
AD14	XPWRRGTON	AE15	XNBATF		
AD15	XNWRESET	AE16	VDDI		
AD16	XNRSTOUT	AE17	XEINT0/GPN0		
AD17	XEINT2/GPN2	AE18	XEINT4/GPN4		
AD18	VDDSYS	AE19	XEINT9/GPN9		
AD19	XEINT11/GPN11	AE20	XEINT13/GPN13		
AD20	XEINT15/GPN15	AE21	XVVD0/GPI0		
AD21	XVVD4/GPI4	AE22	XVVD2/GPI2		
AD22	VDDLCD	AE23	XVVD7/GPI7		
AD23	XVVD13/GPI13	AE24	NC_J		
AD24	XVVD17/GPJ1				

### 1.3 S3C6410 引脚信号描述

下面根据 S3C6410 引脚所能实现的不同功能来进行分类描述。

#### 1. 外部存储器接口

S3C6410 共享存储器端口（SR0MC/OneNAND/NAND/ATA/DRAM0）具体信号描述如表 1-2 所示。

表 1-2 S3C6410 共享存储器端口信号

信号	I/O	描述
ADDR[15: 0]	0	存储器端口 0 共同地址总线
DATA[15: 0]	0	存储器端口 0 共同数据总线
nCS[7: 6]	0	存储器端口 0DRAM 片选支持高达两个存储页
nCS[5: 4]	0	存储器端口 0SR0M/CF 片选支持高达两个存储页
nCS[3:2]	0	存储器端口 0SR0M/OneNAND/NAND Flash 片选支持高达两个存储页
nCS[1: 0]	0	存储器端口 0SR0M 片选支持高达两个存储页
nBE[1: 0]	0	存储器端口 0SR0M 字节有效
WAITn	I	存储器端口 0SR0M 等待
nOE	0	存储器端口 0SR0M/OneNAND 输出有效
new	0	存储器端口 0SR0M/OneNAND 写入有效
ADDRVALID	0	存储器端口 0OneNAND 地址有效
SMCLK	0	存储器端口 0OneNAND 时钟
RDY[0]	I	存储器端口 0OneNAND 组件 0 准备
RDY[1]	I	存储器端口 0OneNAND 组件 1 准备
INT[0]	I	存储器端口 0OneNAND 组件 0 中断
INT[1]	I	存储器端口 0OneNAND 组件 1 中断
RP	0	存储器端口 0OneNAND 复位
ALE	0	存储器端口 0 NAND Flash 地址锁存有效
CLE	0	存储器端口 0 NAND Flash 命令锁存有效

FWEn	0	存储器端口 0 NAND Flash 写入有效
FREn	0	存储器端口 0 NAND Flash 读有效
RnB	I	存储器端口 0 NAND Flash 准备/忙
nIORD_CF	0	存储器端口 0 CF 读选通作为 I/O 模式
nLOWR_CF	0	存储器端口 0 CF 写选通作为 I/O 模式
IORDY	I	存储器端口 0 CF 从 CF 卡等待信号
INT	I	存储器端口 0 CF 从 ATAPI 控制器中断请求
RESET	0	存储器端口 0 CF 卡复位
INPACK	I	存储器端口 0 CF 输入确认在 I/O 模式
REG	0	存储器端口 0 CF 从 CF 卡中断请求
WEn	0	存储器端口 0 CF 写入有效选通
OEn	0	存储器端口 0 CF 输出有效选通
CDn	I	存储器端口 0 CF 卡检测
DQM[1:0]	0	存储器端口 0 DRAM 数据屏蔽
RAS	0	存储器端口 0 DRAM 行地址选通
CAS	0	存储器端口 0 DRAM 列地址选通
SCLK	0	存储器端口 0 DRAM 时钟
SCLKn	0	存储器端口 0 DRAM 反转时钟的 Xm0SCLK
SCKE	0	存储器端口 0 DRAM 时钟有效
DQS[1: 0]	IO	存储器端口 0 DRAM 数据选通
WEn	0	存储器端口 0 DRAM 写入有效
AP	0	存储器端口 0 DRAM 自动预充电

S3C6410 共享存储器端口（SR0MC/ DRAM1）具体信号描述如表 1-3 所示。

表 1-3 S3C6410 共享存储器端口（SR0MC/ DRAM1）信号

信号	I/O	描述
Xm1CKE[1: 0]	0	存储器端口 1DRAM 时钟有效
Xm1SCLK	0	存储器端口 1DRAM 时钟
Xm1SCLKn	0	存储器端口 1DRAM 反转时钟的 Xm1SCLK
Xm1CSn[1: 0]	0	存储器端口 1DRAM 片选支持高达两个存储页
Xm1ADDR[15: 0]	0	存储器端口 1DRAM 地址总线
Xm1RASn	0	存储器端口 1DRAM 行地址选通
Xm1CASn	0	存储器端口 1DRAM 列地址选通
Xm1WEn	0	存储器端口 1DRAM 写入有效
Xm1DATA[15: 0]	IO	存储器端口 1DRAM 低于半数据总线
Xm1DATA[31: 16]	IO	可以作为存储器端口 1DRAM 高于半数据总线使用，通过吸同控制器设置
Xm1DQM[3: 0]	0	存储器端口 1DRAM 数据屏蔽
Xm1DQS[3: 0]	IO	存储器端口 1DRAM 数据选通

2. 串行通信

UART/IrDA/CF 具体信号描述如表 1-4 所示。

表 1-4 UART/IrDA/CF 信号

信号	I/O	描述
XuRXD[0]	I	UART 0 接收数据输入
XuTXD[0]	0	UART0 传输数据输出
XuCTS <sub>n</sub> [0]	I	UART 0 清除发送数据信号
XuRTS <sub>n</sub> [0]	0	UART 0 请求发送输出信号
XuRXD[1]	I	UART 1 接收数据输入
XuTXD[1]	0	UART 1 传输数据输出
XuCTS <sub>n</sub> [1]	I	UART 1 清除发送数据信号
XuRTS <sub>n</sub> [1]	0	UART 1 请求发送输出信号

XuRXD[2]	I	UART 2 接收数据输入
XuTXD[2]	O	UART 2 传输数据输出
XuRXD[3]	I	UART 3 接收数据输入
XuTXD[3]	O	UART 3 传输数据输出
XirSDBW	O	IrDA 收发控制信号（关机和带宽控制）
XirRXD	I	IrDA 接收数据
XirTXD	O	IrDA 发送数据
ADDR_CF[2:0]	O	CF 卡地址
EINT1[12:0]	I	外部中断 1

IIC 总线具体信号描述如表 1-5 所示。

表 1-5 IIC 总线信号

信号	I/O	描述
Xi2cSCL	IO	IIC 总线时钟
Xi2cSDA	IO	IIC 总线数据
EINT1[14: 13]	I	外部中断 1

SPI（2 通道）具体信号描述如表 1-6 所示。

表 1-6 SPI（2 通道）信号

信号	I/O	描述
XspiMISO[0]	IO	SPI MISO[0]。SPI 主设备数据输入线路
XspiCLK[0]	IO	SPI CLK[0]。SPI 时钟作为通道 0
XspiMOS[0]	IO	SPI MOS[0]。SPI 主设备数据输出线路
XspiCS[0]	IO	SPI 片选（只对于从模式）
XspiMISO[1]	IO	SPI MISO[1]。SPI 主设备数据输入线路
XspiCLK[1]	IO	SPI CLK[1]。SPI 时钟作为通道 1
XspiMOS[1]	IO	SPI MOS[1]。SPI 主设备数据输出线路
XspiCS[1]	IO	SPI 片选（只对于从模式）
ADDR_CF[2: 0]	O	CF 卡地址



EINT2[7: 2]	I	外部中断 2
XmmcCMD2	IO	命令/响应 (SD/SDIO/MMC 卡接口通道 2)
XmmcCLK2	0	时钟 (SD/SDIO/MMC 卡接口通道 2)

PCM (2 通道) /IIS/AC97 具体信号描述如表 1-7 所示。

表 1-7 PCM (2 通道) /IIS/AC97 信号

信号	I/O	描述
XpcmDCLK[0]	0	PCM 串行移动时钟
XpcmEXTCLK[0]	I	可选参考时钟
XpcmFSYNC[0]	0	PCM 同步指示字的开始
XpcmSIN[0]	I	PCM 串行数据输入
XpcmSOUT[0]	0	PCM 串行数据输出
XpcmDCLK[1]	0	PCM 串行移动时钟
XpcmEXTCLK[1]	I	可选参考时钟
XpcmFSYNC[1]	0	PCM 同步指示字的开始
XpcmSIN[1]	I	PCM 串行数据输入
XpcmSOUT[1]	0	PCM 串行数据输出
Xi2sLRCK[1:0]	IO	IIS 总线通道选择时钟
Xi2sCDCLK[1:0]	0	IIS 编解码器系统时钟
Xi2sCLK[1:0]	IO	IIS 总线串行时钟
Xi2sDI[1:0]	I	IIS 总线串行数据输入
Xi2sDO[1:0]	0	IIS 总线串行数据输出
X97BITCLK	I	AC-Link 位总线 (12.288MHz) 从 AC97 编解码器到 AC97 控制器
X97RESETn	0	AC-Link 复位至编解码器
X97SYNC	0	从 AC97 控制器 AC-Link 帧同步 (采样频率 48kHz)
X97SDI	I	AC-Link 串行数据输入从 AC97 编解码器
X97SDO	0	AC-Link 串行数据输出至 AC97 编解码器
ADDR_CF[2:0]	0	CF 卡地址
EINT3[4:0]	I	外部中断 3

USB 主设备具体信号描述如表 1-8 所示。

表 1-8 USB 主设备信号

信号	I/O	描述
XuhDN	IO	USB 数据引脚 DATA (-) 用作 USB1.1 主设备
XuhDP	IO	USB 数据引脚 DATA (+) 用作 USB1.1 主设备

USB OTG 具体信号描述如表 1-9 所示。

表 1-9 USB OTG 信号

信号	I/O	描述
XusbDP	IO	USB 数据引脚 DATA (+)
XusbDM	IO	USB 数据引脚 DATA (-)
XusbXTI	I	晶体振荡器 XI 信号
XusbXTO	I	晶体振荡器 XO 信号
XusbREXT	IO	外部 3.4k $\Omega$ (+/-1%) 电阻连接
XusbVBUS	IO	USB 迷你插座 Vbus
XusbID	I	USB 迷你插座标识
XusbDRVVBUS	O	驱动 Vbus 作为芯片外电荷泵

### 3. 并行通信

外部中断具体信号描述如表 1-10 所示。

表 1-10 外部中断信号

信号	I/O	描述
XEINT[15: 0]	I	外部中断
XkpROW[7: 0]	I	便携式键盘 I/F 行
ADDR_CF[2: 0]	O	CF 卡地址

#### 4. 调制解调器接口

主设备 I/F/HIS (MIPI) /Key I/F/ATA 具体信号描述如表 1-11 所示。

表 1-11 主设备 I/F/HIS (MIPI) /Key I/F/ATA 信号

信号	I/O	描述
XhiCSn	I	片选，通过调制解调器芯片驱动
XhiCSn_main	I	片选作为主 LCD 旁路，通过调制解调器芯片驱动
XhiCSn_sub	I	片选作为子 LCD 旁路，通过调制解调器芯片驱动
XhiWEn	I	写入使能，通过调制解调器芯片驱动
XhiOEn	I	读使能，通过调制解调器芯片驱动
XhiINTR	O	调制解调器芯片中断请求
XhiADDR[12: 0]	I	地址总线，通过调制解调器芯片驱动
XhiDATA[17: 0]	IO	数据总线，通过调制解调器芯片驱动
XEINT[27: 16]	I	外部中断
XkpCOL[7: 0]	O	便携式键盘接口列输出
XhrxREADY	O	准备信号指示传输一个新的物理层帧可以开始
XhrxWAKE	I	唤醒信号用来指示接收器发射将开始一个传输
XhpROW[7: 0]	I	便携式键盘接口行输入
DATA_CF [15: 0]	IO	CF 卡数据
CE_CF[1: 0]	O	CF 卡使能选通
IORE_CF	O	CF 读选通为 I/O 模式
IOWR_CF	O	CF 写选通为 I/O 模式
IORDY_CF	I	CF 从 CF 卡等待信号
ADDR_CR[2: 0]	O	CF 卡地址

PWM 具体信号描述如表 1-12 所示。

表 1-12 PWM 信号

信号	I/O	描述
XpwmECLK	I	PWM 定时器外部时钟
XpwmTOUT[1: 0]	O	PWM 定时器输出
XCLKOUT	O	时钟输出信号
EINT4[13]	I	外部中断 4

5. 图像/视频处理

相机接口具体信号描述如表 1-13 所示。

表 1-13 相机接口信号

信号	I/O	描述
XciCLK	O	主时钟相机处理器 A
XciHREF	I	水平同步，通过相机处理器 A 驱动
XciPCLK	I	像素时钟，通过相机处理器 A 驱动
XciVSYNC	I	垂直同步，通过相机处理器 A 驱动
XciRSTn	O	软件复位到相机处理器 A 驱动
XciYDATA[7: 0]	I	在 8 位模式，像素数据为 YCbCr，或在 16 位模式下为 Y，通过相机处理器 A 驱动
EINT4[12: 0]	I	外部中断 4

6. 显示器控制

2 通道 DAC 具体信号描述如表 1-14 所示。

表 1-14 2 通道 DAC 信号

信号	I/O	描述
XdacVREF	AI	参考电压输入
Xdac1REF	AI	外部寄存器连接
XdacCOMP	AI	外部电容器连接
XdacOUT_0	AO	DAC 模拟输出
XdacOUT_1	AO	DAC 模拟输出

ADC 具体信号描述如表 1-15 所示。

表 1-15 ADC 信号

信号	I/O	描述
Xdac_AIN [7: 0]	AI	ADC 模拟输入

PLL 具体信号描述如表 1-16 所示。

表 1-16 PLL 信号

信号	I/O	描述
Xp11EFILTER		环路滤波器电容器

7. 存储设备

MMC 2 通道具体信号描述如表 1-17 所示。

表 1-17 MMC 2 通道信号

信号	I/O	描述
XmmcCLK0	0	时钟（SD/SDIO/MMC 卡接口通道 0）
XmmcCMD0	IO	命令/响应（SD/SDIO/MMC 卡接口通道 0）
XmmcDAT0[3: 0]	IO	数据（SD/SDIO/MMC 卡接口通道 0）
XmmcCDN0	I	卡删除（SD/SDIO/MMC 卡接口通道 0）
XmmcCLK1	0	时钟（SD/SDIO/MMC 卡接口通道 1）
XmmcCMD1	IO	命令/响应（SD/SDIO/MMC 卡接口通道 1）
XmmcDAT1[7: 0]	IO	数据（SD/SDIO/MMC 卡接口通道 1）
XmmcCLK2	0	时钟（SD/SDIO/MMC 卡接口通道 2）
XmmcCMD2	IO	命令/响应（SD/SDIO/MMC 卡接口通道 2）
XmmcDAT2[3: 0]	IO	数据（SD/SDIO/MMC 卡接口通道 2）
ADDR_CF[2: 0]	0	CF 卡地址
EINT5[6: 0]	I	外部中断 5
EINT 6[9: 0]	I	外部中断 6

## 8. 系统管理器

复位具体信号描述如表 1-18 所示。

表 1-18 复位信号

信号	I/O	描述
XnRESET	I	XnRESET 暂停任何操作在处理和取代 S3C6410 到一个已知的复位状态。对于复位，XnRESET 必须保持 L 电平至少四个 FCLK，在处理器功率稳定下来之后
XnWRESET	I	系统热复位。当维护 SDRAM 内容时复位整个系统
XsRSTOUTn	O	外部设备复位控制（sRSTOUTn = nRESET & nWDTRST & SW_RESET）

时钟具体信号描述如表 1-19 所示。

表 1-19 时钟信号

信号	I/O	描述
XrtcXTI	I	RTC 32kHz 晶体输入
XrtcXT0	O	RTC32kHz 晶体输出
X27mXTI	I	显示器模式 27MHz 晶体输入
X27mXT0	O	显示器模式 27MHz 晶体输出
XXTI	I	内部振荡器电路晶体输入
XXT0	O	内部振荡器电路晶体输出
XEXTCLK	I	外部时钟源

JTAG 具体信号描述如表 1-20 所示。

表 1-20 JTAG 信号

信号	I/O	描述
XjTRSTn	I	XjTRSTn (TAP 控制器复位) 在开始复位 TAP 控制器。如果使用调试器，一个 10k $\Omega$ 上拉电阻必须被连通。如果不使用调试器，XjTRSTn 引脚必须在 L 或低又小脉冲
XjTMS	O	XjTMS (TAP 控制器模式选择) 控制 TAP 控制器状态的顺序。一个 10k $\Omega$ 的上拉电阻必须被连接到 TMS 引脚
XjTCK	I	XjTCK (TAP 控制器时钟) 提供 JTAG 逻辑的时钟输入。一个 10k $\Omega$ 的下拉电阻必须被连接到 TMS 引脚

XjRTCK	0	XjRTCK（TAP 控制器返回的时钟）提供 JTAG 逻辑时钟输出
XjTDI	I	XjTDI（TAP 控制器数据输入）是测试指令和数据的串行输入。一个 10k 的上拉电阻必须连接到 TDI 引脚
XjTDO	0	XjTDO（TAP 控制器数据输出）测试指令和数据的串行输入。它可能通过 GPIO 电阻控制下拉
XjDBGSEL	I	JTAG 选择。1：外设 JTAG，0：ARM1176JZF-S 核心 JTAG

MISC 具体信号描述如表 1-21 所示。

表 1-21 MISC 信号

信号	I/O	描述
XOM[4: 0]	I	操作模式选择。
XPWRRGTON	0	功率调节器使能
XSELNAND	I	选择 Flash 存储器。1：OneNAND，1： NAND
XnBATF	I	电池故障指示

9. 电源组

VDD 具体信号描述如表 1-22 所示。

表 1-22 VDD 信号

信号	I/O	描述	电压
VDDALIVE	P	带电组件的内部 VDD	1.0
VDDARM	P	ARM1176 核和缓存的内部 VDD	TBD
VDDINT	P	逻辑的内部 VDD	TBD
VDDMPLL	P	MPLL 核的 VDD	TBD
VDDEPLL	P	APLL 核的 VDD	TBD
VDDOTG	P	EPLL 核的 VDD	3.3
VDDOTGI	P	USB OTG PHY 的 VDD	1.0
VDDMMC	P	USB OTG PHY 的内部 VDD	2.5~3.3
VDDHI	P	SDMM 的 IO VDD	2.5~3.3
VDDLCD	P	主设备 I/F 的 IO VDD	2.5~3.3

VDDPCM	P	LCD 的 IO VDD	2.5~3.3
VDDEXT	P	PCM 的 IO VDD（音频 I/F-I <sup>2</sup> S，AC97）	3.3
VDDSYS	P	外部 I/F 的 IO VDD（UART， I <sup>2</sup> C，相机 I/F，USB 主设备等）	3.3
VDDADC	P	ADC 核和 IO 的 VDD	3.3
VDDDAC	P	DAC 核和 IO 的 VDD	3.3
VDDRTC	P	RTC 逻辑和 IO 的 VDD	2.5
VDDM0	P	存储器端口 0 的 IO VDD	1.8~2.5
VDDM1	P	存储器端口 1 的 IO VDD	1.8~2.5

VSS 具体信号描述如表 1-23 所示。

表 1-23 VSS 信号

信号	I/O	描述
VSSIP	G	内部逻辑接地&ARM1176 核和缓存
VSSMEM	G	存储器端口 0 和 1 的 IO 接地
VSSOTG	G	USB OTG PHY 的接地
VSSOTGI	G	USB OTG PHY 的内部接地
VSSPERI	P	USB 主设备，SDMMC，主设备 I/F，LCD，PCM，外部 I/F 和系统控制器
VSSAPLL	G	APLL 核接地
VSSMPLL	G	MPLL 核接地
VSSEPLL	G	EPLL 核接地
VSSADC	G	ADC 核接地
VSSDAC	G	DAC 核接地

注意：

- (1) I/O 表示输入/输出；
- (2) AI/AO 表示模拟输入/输出；
- (3) ST 表示施密特触发；
- (4) P 表示电源。



## 2 存储器映射

S3C6410 支持 32 位物理地址域，并且这些地址域分成两部分，一部分用于存储，另一部分用于外设。

### 2.1 存储器系统模块图

通过 SPINE 总线访问主存，主存的地址范围是 0x0000\_0000~0x6FFF\_FFFF。主存部分分成四个区域：引导镜像区、内部存储区、静态存储区和动态存储区。

引导镜像区的地址范围是从 0x0000\_0000~0x07FF\_FFFF，但是没有实际的映射内存。引导镜像区反映一个镜像，这个镜像指向内存的一部分区域或者静态存储区。引导镜像的开始地址是 0x0000\_0000。

内部存储区用于启动代码访问内部 ROM 和内部 SRAM，也被称做 Steppingstone。每块内部存储器的起始地址是确定的。内部 ROM 的地址范围是 0x0800\_0000~0x0BFF\_FFFF，但是实际存储仅 32KB。该区域是只读的，并且当内部 ROM 启动被选择时，该区域能映射到引导镜像区。内部 SRAM 的地址范围是 0x0C00\_0000~0x0FFF\_FFFF，但是实际存储仅 4KB。该区域能被读和写，当 NAND 闪存启动被选择时能映射到引导镜像区。

静态存储区的地址范围是 0x1000\_0000~0x3FFF\_FFFF。通过该地址区域能访问 SRAM、NOR Flash、同步 NOR 接口设备、和 Steppingstone。每一块区域代表一个芯片选择，例如，地址范围从 0x1000\_0000~0x17FF\_FFFF 代表 X<sub>m</sub>OCS<sub>n</sub>[0]。每一个芯片选择的开始地址是固定的。NAND Flash 和 CF/ATAPI 不能通过静态存储区访问，因此任何 X<sub>m</sub>OCS<sub>n</sub>[5:2] 映射到 NFCON 或 CFCON，相关地址区域应当被访问。一个例外，如果 X<sub>m</sub>OCS<sub>n</sub>[2] 用于 NAND Flash，Steppingstone 映射到存取区从 0x2000\_0000~27FF\_FFFF。

动态存储区的地址范围是 0x4000\_0000~0x6FFF\_FFFF。DMC0 有权使用地址 0x4000\_0000~0x4FFF\_FFFF，并且 DMC1 有权使用地址 0x5000\_0000~0x6FFF\_FFFF。对于每一块芯片选择的起始地址是可以进行配置的。

外设区域通过 PERI 总线被访问，它的地址范围是 0x7000\_0000~0x7FFF\_FFFF。这个地址范围的所有 SFR 能被访问。而且如果数据需要从 NFCON 或 CFCON 传输，这些数据需要通过 PERI 总线传输。

存储器系统模块的地址映射图，如图 2-1 所示。

	AXI Remap = 0				AXI Remap = 1				
0xFFFF_FFF F	Reserved				Reserved				
0x8000_0000									
	SFRs				SFRs				
0x7000_0000									
	DMC1				DMC1				
0x5000_0000	DMC0				DMC0				
0x4000_0000	SRAM5			CF	SRAM5			CF	
0x3800_0000	SRAM4			CF	SRAM4			CF	
0x3000_0000	SRAM3		One NAND1	NAND1	SRAM3		One NAND1	NAND1	
0x2800_0000	SRAM2		One NAND0	Boot Loader	SRAM2		One NAND0	NAND0	
0x2000_0000	SRAM1				SRAM1				
0x1800_0000	SRAM0	External ROM			SRAM0	External ROM			
0x1000_0000	Boot Loader				Boot Loader				
0x0C00_0000	Internal ROM				Internal ROM				
0x0800_0000									
0x0000_0000	SRAM0	External ROM	One NAND0	Boot Loader	Internal ROM				

图 2-1 地址映射

## 2.2 特殊设备地址空间

如表 2-1 所示，显示了特殊设备地址空间的描述。

表 2-1 特殊设备地址空间

地址		大小 (MB)	描述	备注
0x0000_0000	0x07FF_FFFF	128MB	Remap 0 : SRAM0 或 Boot Loader Remap 1 : 内部 ROM	被映射区域
0x0800_0000	0x0BFF_FFFF	64MB	内部 ROM	
0x0C00_0000	0x0FFF_FFFF	64MB	Stepping Stone (Boot Loader)	
0x1000_0000	0x17FF_FFFF	128MB	SMC Bank 0	
0x1800_0000	0x1FFF_FFFF	128MB	SMC Bank 1	
0x2000_0000	0x27FF_FFFF	128MB	SMC Bank 2	

0x2800_0000	0x2FFF_FFFF	128MB	SMC Bank 3	
0x3000_0000	0x37FF_FFFF	128MB	SMC Bank 4	
0x3800_0000	0x3FFF_FFFF	128MB	SMC Bank 5	
0x4000_0000	0x47FF_FFFF	128MB	存储器端口 1 DDR/SDRAM Bank0	
0x4800_0000	0x4FFF_FFFF	128MB	存储器端口 1 DDR/SDRAM Bank1	
0x5000_0000	0x5FFF_FFFF	256MB	存储器端口 2DDR/SDRAM Bank0	
0x6000_0000	0x6FFF_FFFF	256MB	存储器端口 2DDR/SDRAM Bank1	

如表2-2所示，显示了AHB总线存储器映射。

表 2-2 AHB 总线存储器映射

		描述	备注
0x7000_0000	0x700F_FFFF	SROM SFR	
0x7010_0000	0x701F_FFFF	OneNAND SFR	
0x7020_0000	0x702F_FFFF	NFCON SFR	
0x7030_0000	0x703F_FFFF	CFCON SFR	
0x7040_0000	0x70FF_FFFF	保留	
0x7100_0000	0x710F_FFFF	TZIC0	
0x7110_0000	0x711F_FFFF	TZIC1	
0x7120_0000	0x712F_FFFF	INTC0	
0x7130_0000	0x713F_FFFF	INTC1	
0x7140_0000	0x71FF_FFFF	保留	
0x7200_0000	0x72FF_FFFF	保留	
0x7300_0000	0x7300_0FFF	ETB 存储器	
0x7310_0000	0x731F_FFFF	ETB寄存器	
0x7320_0000	0x73FF_FFFF	保留	

0x7400_0000	0x740F_FFFF	间接主机I/F	
0x7410_0000	0x741F_FFFF	直接主机I/F	
0x7420_0000	0x742F_FFFF	保留	
0x7430_0000	0x743F_FFFF	USB Host	
0x7440_0000	0x744F_FFFF	MDP I/F	
0x7450_0000	0x74FF_FFFF	保留	
0x7500_0000	0x750F_FFFF	DMA0	
0x7510_0000	0x751F_FFFF	DMA1	
0x7520_0000	0x752F_FFFF	保留	
0x7530_0000	0x753F_FFFF	保留	
0x7540_0000	0x75FF_FFFF	保留	
0x7600_0000	0x760F_FFFF	保留	
0x7610_0000	0x761F_FFFF	2D图形	
0x7620_0000	0x762F_FFFF	TV编码器	
0x7630_0000	0x763F_FFFF	TV定标器	

如表7-3所示，显示了APB总线存储器映射。

表 2-3 APB 总线存储器映射

地址		描述	备注
0x7640_0000	0x76FF_FFFF	保留	
0x7700_0000	0x770F_FFFF	Post处理器	
0x7710_0000	0x771F_FFFF	LCD控制器	
0x7720_0000	0x772F_FFFF	旋转器	

0x7730_0000	0x77FF_FFFF	保留	
0x7800_0000	0x783F_FFFF	相机I/F	
0x7840_0000	0x787F_FFFF	保留	
0x7880_0000	0x78BF_FFFF	JPEG	
0x78C0_0000	0x78FF_FFFF	保留	
0x7900_0000	0x79FF_FFFF	保留	
0x7A00_0000	0x7AFF_FFFF	保留	
0x7B00_0000	0x7BFF_FFFF	保留	
0x7C00_0000	0x7C0F_FFFF	USB OTG	
0x7C10_0000	0x7C1F_FFFF	USB OTG SFR	
0x7C20_0000	0x7C2F_FFFF	SD-MMC 控制器 0 (高速/CE-ATA)	
0x7C30_0000	0x7C3F_FFFF	SD-MMC 控制器 1 (高速/CE-ATA)	
0x7C40_0000	0x7C4F_FFFF	SD-MMC 控制器 2 (高速/CE-ATA)	
0x7C50_0000	0x7C5F_FFFF	保留	
0x7D00_0000	0x7D0F_FFFF	D&I (安全总线系统配置) SFR	
0x7D10_0000	0x7D1F_FFFF	AES_RX	
0x7D20_0000	0x7D2F_FFFF	DES_RX	
0x7D30_0000	0x7D3F_FFFF	HASH (SHA/PRNG)_RX	
0x7D40_0000	0x7D4F_FFFF	RX FIFO SFR	
0x7D50_0000	0x7D5F_FFFF	AES_TX	
0x7D60_0000	0x7D6F_FFFF	DES_TX	
0x7D70_0000	0x7D7F_FFFF	HASH(SHA/PRNG)_TX	
0x7D80_0000	0x7D8F_FFFF	TX FIFO SFR	
0x7D90_0000	0x7D9F_FFFF	RX_FIFO	
0x7DA0_0000	0x7DAF_FFFF	TX_FIFO	

0x7DB0_0000	0x7DBF_ FFFF	SDMA0	
0x7DC0_0000	0x7DCF_ FFFF	SDMA1	

如表2-4所示，显示了APB总线存储器映射。

表 2-4 APB 总线存储器映射

地址		描述	备注
0x7DD0_0000	0x7DFF_FFFF	保留	
0x7E00_0000	0x7E00_0FFF	DMC0 SFR	
0x7E00_1000	0x7E00_1FFF	DMC1 SFR	
0x7E00_2000	0x7E00_2FFF	MFC SFR	
0x7E00_3000	0x7E00_3FFF	保留	
0x7E00_4000	0x7E00_4FFF	看门狗定时器	
0x7E00_5000	0x7E00_5FFF	RTC	
0x7E00_6000	0x7E00_6FFF	HSI TX	
0x7E00_7000	0x7E00_7FFF	HIS RX	
0x7E00_8000	0x7E00_8FFF	保留	
0x7E00_9000	0x7E00_9FFF	保留	
0x7E00_A000	0x7E00_AFFF	键盘I/F	
0x7E00_B000	0x7E00_BFFF	ADC/触摸屏	
0x7E00_C000	0x7E00_CFFF	ETM	
0x7E00_D000	0x7E00_DFFF	Key	
0x7E00_E000	0x7E00_EFFF	芯片 ID	
0x7E00_F000	0x7E00_FFFF	系统控制器	
0x7F00_0000	0x7F00_0FFF	TZPC	
0x7F00_1000	0x7F00_1FFF	AC97	
0x7F00_2000	0x7F00_2FFF	IIS 通道0	
0x7F00_3000	0x7F00_3FFF	IIS 通道1	
0x7F00_4000	0x7F00_4FFF	IIC	

0x7F00_5000	0x7F00_5FFF	UART	
0x7F00_6000	0x7F00_6FFF	PWM定时器	
0x7F00_7000	0x7F00_7FFF	IrDA	
0x7F00_8000	0x7F00_8FFF	GPIO	
0x7F00_9000	0x7F00_9FFF	PCM通道0	
0x7F00_A000	0x7F00_AFFF	PCM通道1	
0x7F00_B000	0x7F00_BFFF	SPI0	
0x7F00_C000	0x7F00_CFFF	SPI1	
0x7F00_D000	0x7F00_DFFF	保留	
0x7F00_E000	0x7F00_EFFF	保留	
0x7F00_F000	0x7F00_FFFF	保留	

## 3 系统控制器

本小节主要介绍系统控制器在 S3C6410 RISC 微处理器中的功能和使用。系统控制器由两部分组成：分别是系统时钟控制和系统电源管理控制。系统时钟控制逻辑，在 S3C6410 中生成所需的系统时钟信号，用于 CPU 的 ARMCLK，AXI/AHB 总线外设的 HCLK 和 APB 总线外设的 PCLK。在 S3C6410 中有三个 PLL。一个仅用于 ARMCLK，一个用于 HCLK 和 PCLK，最后一个用于外设，特别用于音频相关的时钟。通过外部提供的时钟源，时钟控制逻辑产生慢速时钟信号 ARMCLK，HCLK 和 PCLK。该每个外设块的时钟信号可能被启用或禁用，由软件控制以减少电源消耗。

在电源控制逻辑中，S3C6410 有多种电源管理方案，以保持电力系统的最佳消耗，用于一个给定的任务。在 S3C6410 中，电源管理由四个模块组成：通用时钟门控模式，空闲模式，停止模式和睡眠模式。

在 S3C6410 中，通用时钟门控模式用来控制内部外设时钟的开/关。可以通过用于外设所要求的特定应用提供时钟，使用通用时钟门控模式来优化 S3C6410 的电源消耗。例如：如果定时器没有要求，则可以中断时钟定时器，以降低功耗。

闲置模式仅中断 ARMCLK 到 CPU 内核，它提供时钟给所有外设。通过使用闲置模式，电力消耗通过 CPU 内核而减少。

停止模式通过禁用 PLL 冻结所有时钟到 CPU 以及外设。在 S3C6410 中，电力消耗仅因为漏电流。

睡眠模式断开内部电源。因此，电力消耗因为除了唤醒逻辑，CPU 和内部逻辑将为零。为了使用睡眠模式，两个独立的电源是必需的。两个电源中的一个用于唤醒逻辑提供电力，另一个提供其他内部逻辑，包括 CPU 和为了旋转开/关所必须进行的控制。

### 3.1 系统控制器的特性

系统控制器包含的特性有以下几个方面：

- 三个 PLL：ARM PLL，主 PLL，额外的 PLL（这些模块用于使用特殊频率）。
- 五种省电模式：正常，闲置，停止，深度停止和睡眠。
- 五种可控制的电源范围：domain-V，domain-I，domain-P，domain-F，domain-S。
- 内部子块的控制操作时钟。



- 控制总线优先权。

### 3.2 功能描述

这部分主要介绍 S3C6410 系统控制器的功能。包含时钟的体系结构，复位设计和电源管理模式。

**1. 硬件体系结构**

如图 2-2 所示，说明了 S3C6410 的结构框图。S3C6410 是由 ARM1176 处理器，几个多媒体协处理器和各种外设 IP 组成的。ARM1176 处理器是通过 64 位 AXI 总线连接到几个内存控制器上的。这样做是为了满足带宽需求。多媒体协处理器分为五个电源域，包括 MFC（多格式编解码器），JPEG，Camera 接口，TV 译码器等等。当 IP 没有被一个应用程序所要求时，五个电源域可以进行独立的控制，以减少不必要的电力消耗。

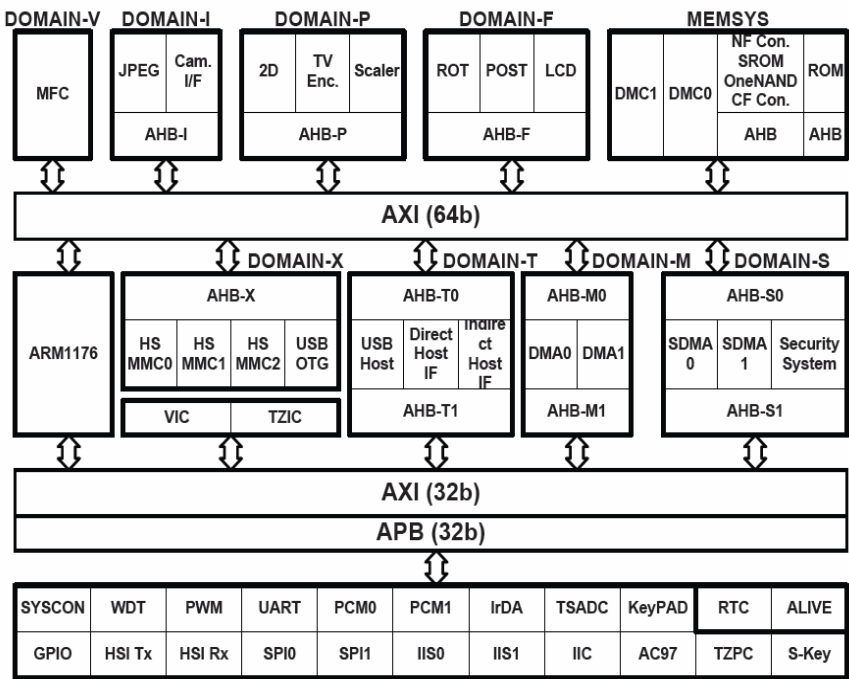


图 3-1 S3C6410 的结构框图

### 2. 时钟体系结构

如图 3-2 所示，说明了时钟发生器模块的结构框图。时钟源在外部晶体 (XXTIp11) 和外部时钟 (XEXTCLK) 两者之间进行选择。该时钟发生器由三个 PLL（锁相环）组成，产生高频率的时钟信号可以达到 1.6GHz。

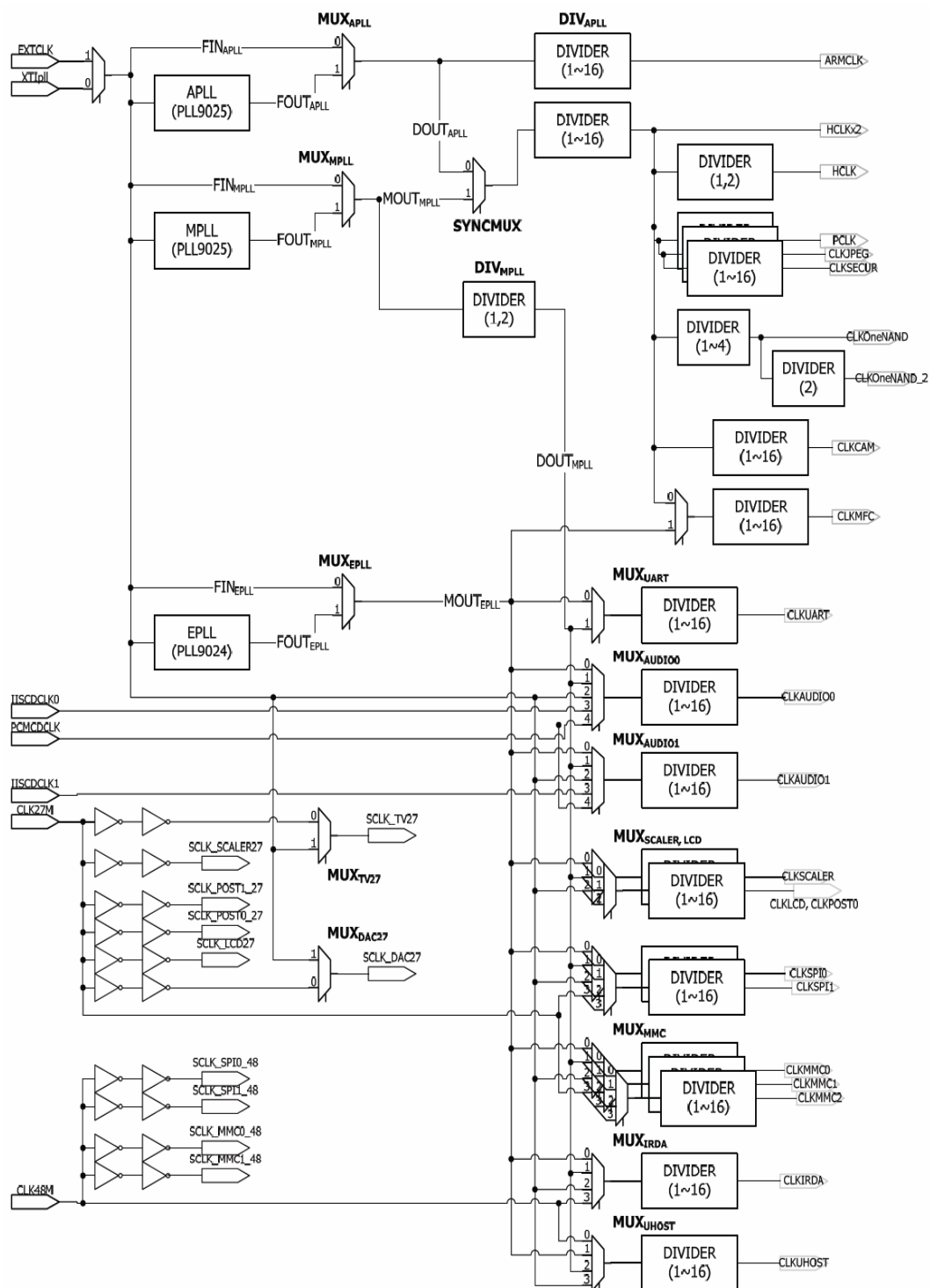


图 3-2 时钟发生器的结构框图

3. 时钟源的选择

内部时钟会产生用于外部的时钟源，其说明如表 3-1 所示。当外部复位信号被声明时，OM[4:0]引脚决定了 S3C6410 的操作模式。OM[0]引脚选择外部时钟源，例如，如果 OM[0]是 0，则 XXTIp11（外部晶体）被选择。否则，XEXTCLK 被选择。

表 3-1 启动时设备操作模式的选择

OM[4:0]	启动设备	功能	时钟源
0000X	NAND	AdvFlash=0, AddrCycle=3	如果OM[0] 是0, XXTIp11 被选择。  如果 OM[0] 是 1,XEXTCLK 被选择。
0001X		AdvFlash=0, AddrCycle=4	
0010X		AdvFlash=1, AddrCycle=4	
0011X		AdvFlash=1, AddrCycle=5	
0100X	SROM	—	
0101X	NOR (26 位)	—	
0110X	OneNAND	—	
0111X	MODEM	—	
RESERVED	保留	—	
1111X	内部 ROM	—	

操作模式根据启动设备主要分为六种类别。启动设备可以是 NAND，SROM，NOR，OneNAND，MODEM 和内部 ROM 其中的的一种。当启动设备是 NAND 时，可以选择的额外的特征如表 6-108 所示。当 NAND Flash 设备被使用时，XSELNAND 引脚必须是 1，即使它用来作为启动设备或存储设备。当 OneNAND Flash 设备被使用时，XSELNAND 引脚必须是 0，即使它用来作为启动设备或存储设备。当 NAND/OneNAND 设备不使用时，XSELNAND 可以是 0 或 1。

4. 锁相环（PLL）

S3C6410 内部的三个 PLL，分别是 APLL，MPLL 和 EPLL。带有一个参考输入时钟操作频率和相位的同步输出信号。在这个应用当中，包括基本模块的说明，如图 3-3 所示。电压控制振荡器 (VCO) 产生的输出频率成正比，输入到直流电压。通过 P，前置配器划分输入频率（FIN）。通过 M，主分频器分割 VCO 的输出频率，用于输入到相位频率检测器，（PFD）。通过 S，post 定标器划分为 VCO 的输出频率。相位差探测器计算相位差和电荷泵的增加/减少输出电压。每个 PLL 的输出时钟频率是可以计算的。

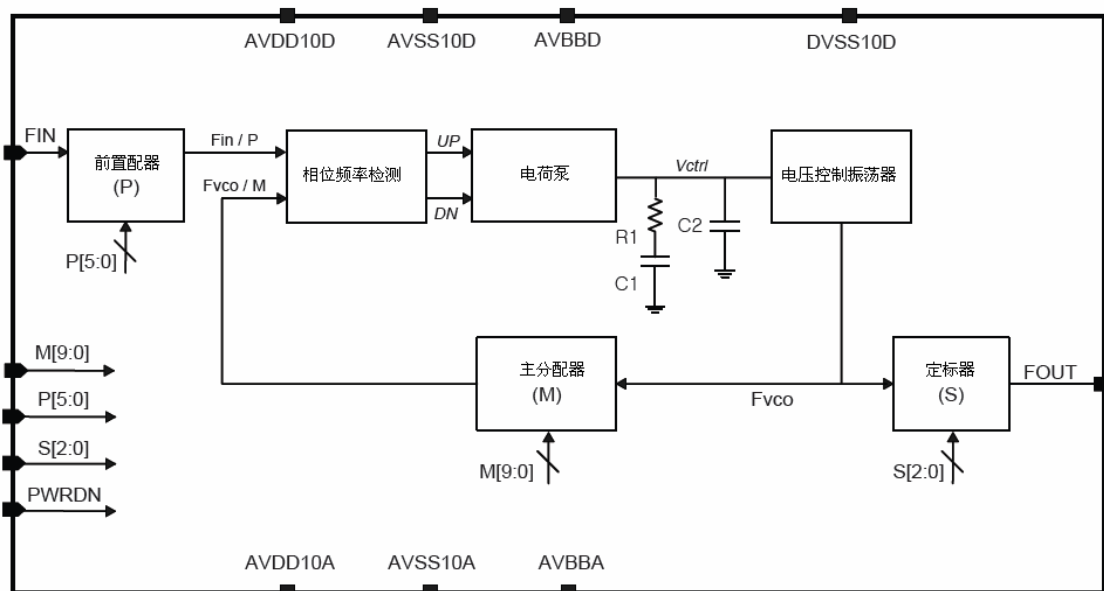


图 3-3 PLL 结构框图(只有 APLL, MPLL)

## 5. PLL 和输入参考时钟之间时钟选择

如图 3-4 所示，说明了时钟发生器逻辑。S3C6410 有三个 PLL，APLL 用于 ARM 时钟操作，MPLL 用于主时钟操作，EPLL 用于特殊用途。时钟操作被分为三组。第一组是 ARM 时钟，从 APLL 产生。MPLL 产生主系统时钟，用于操作 AXI，AHB 和 APB 总线操作。最后一组是从 EPLL 产生的，产生的时钟主要用于外设 IP's，例如，UART，IIS 和 IIC 等等。

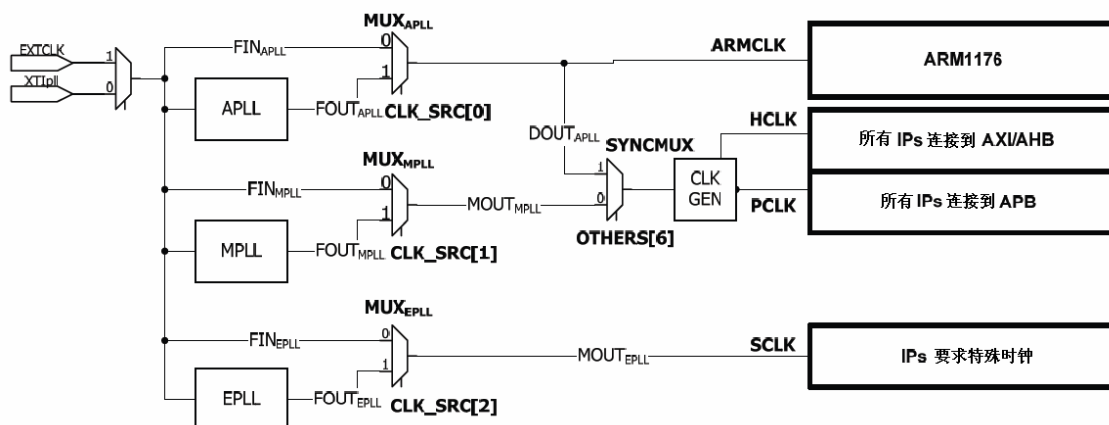


图 3-4 从 PLL 输出时钟发生器

CLK\_SRC 寄存器的最低三位控制三组时钟源。当位为 0 时，则输入时钟绕过组，否则，PLL 输出将被应用到组。

## 6. ARM 和 AXI/AHB/APB 总线时钟发生器

S3C6410 的 ARM1176 处理器运行时最大可达 227MHz。操作频率可以通过内部时钟分频器来控制，DIVARM，没有改变 PLL 频率。该分频器的比率从 1~8 不同。ARM 处理器降低了运行速度，以减少功耗。

S3C6410 由 AXI 总线，AHB 总线和 APB 总线组成，以优化性能要求。内部的 Ips 连接到适当的总线系统，以满足 I/O 带宽和操作性能。当在 AXI 总线或 AHB 总线上时，操作速度可以最大达到 133MHz。当在 APB 总线上时，最大的操作速度可以达到 66MHz。而且，总线速度在 AHB 和 APB 之间有高度依赖同步数据传输。如图 3-5 所示，该图说明了总线时钟发生器部分满足总线系统时钟的要求。

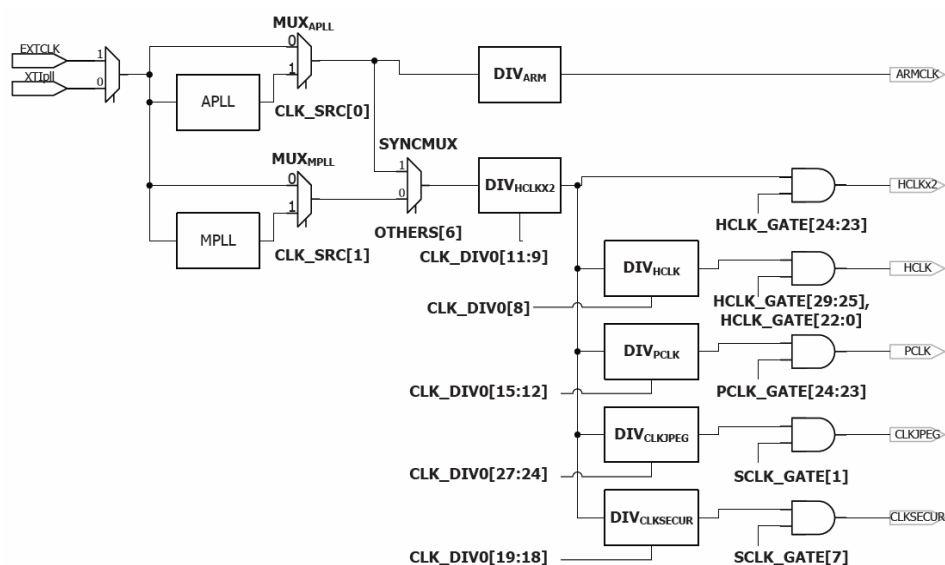


图 3-5 ARM 和总线时钟发生器

S3C6410 的 HCLKX2 时钟提供了两个 DDR 控制器，DDR0 和 DDR1。操作速度可以达到最高 266MHz，通过 DDR 控制器发送和接收数据。当操作没有被请求时，每个 HCLKX2 时钟可独立地屏蔽，以减少多余的功率耗散在时钟分配网络上。所有的 AHB 总线时钟都是从 DIVHCLK 时钟分频器中产生的。产生的时钟可以独立地屏蔽，以减少多余的功率耗散。HCLK\_GATE 寄存器控制 HCLKX2 和 HCLK 的主机操作。

通过 APB 总线系统，低速互连 IP 传输数据。运行中的 APB 时钟高达 66MHz，并且是从 DIVPCLK 时钟分频器产生的。也可以屏蔽使用 PCLK\_GATE 寄存器。作为描述，频率比率在 AHB 时钟和 APB 时钟之间必须有一个整数值。例如，如果 DIVHCLK 的 CLK\_DIV0[8] 位为 1，则 DIVPCLK 的 CLK\_DIV0[15:12] 必须是 1, 3, ...。

否则，APB 总线系统上的 IP 不能正确的传输数据。

在 AHB 总线系统上，JPEG 和安全子系统在 133MHz 时不能运行。AHB 时钟带有 DIVCLKJPEG 和 DIVCLKSECUR 独立地产生。因此，作为 APB 时钟它们有相同的限制。如表 3-2 所示，列出了建议时钟分频器的比例。

表 3-2 时钟分频器典型值的设置（SFR 设置值/输出频率）

APLL	MPLL	DIVARM	DIVHCLKX2	DIVHCLK	DIVPCLK	DIVCLKJPEG	DIVCLKSECUR
266MHz	266MHz	0 / 266MHz	0 / 266MHz	1 / 133MHz	3 / 66MHz	3 / 66MHz	3 / 66MHz
400MHz	266MHz	0 / 400MHz	0 / 266MHz	1 / 133MHz	3 / 66MHz	3 / 66MHz	3 / 66MHz
533MHz	266MHz	0 / 533MHz	0 / 266MHz	1 / 133MHz	3 / 66MHz	3 / 66MHz	3 / 66MHz
667MHz	266MHz	0 / 667MHz	0 / 266MHz	1 / 133MHz	3 / 66MHz	3 / 66MHz	3 / 66MHz

上述表格所描述的是，该分频器用于 ARM 独立地使用 APLL 输出时钟，并没有约束时钟分频器的值。

7. 时钟比例的改变

时钟分频器产生各种操作时钟，包括系统操作时钟，如 ARMCLK, HCLKX2, HCLK 和 PCLK。图 3-6 描述的是一个转换波形，时钟分频器用于系统操作时钟从 1~2 变化比例。从图中的波形可以看出，PLL 输出时钟缓慢地改变周期的比例。这个周期是不固定的，在典型的例子中大约是 10~20 时钟周期。

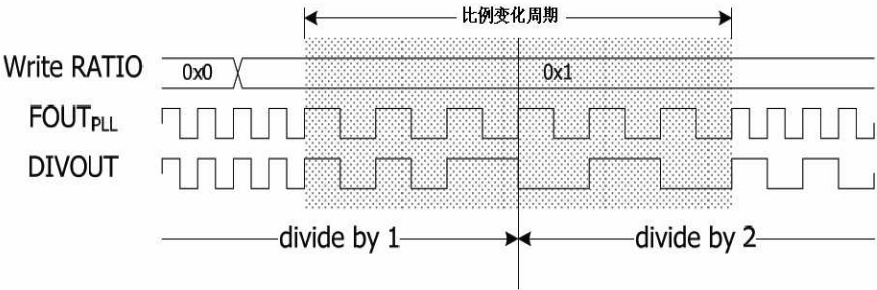


图 3-6 系统时钟比例变化的波形

因此，如果一些 IP 运行，必须特别注意比率改变的周期。否则，IP 操作将失败。

## 8. OneNAND 时钟发生器

OneNAND 接口控制器要求两个同步时钟。一个时钟的频率必须是其他时钟频率的一半。如图 3-7 所示，两个时钟的产生。

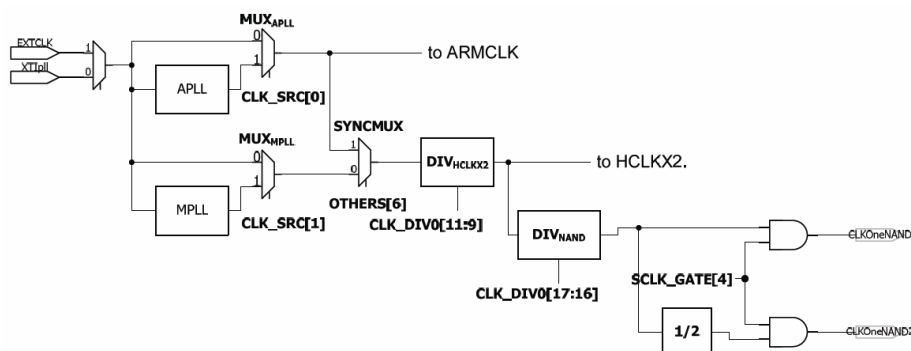


图 3-7 OneNAND 时钟发生器

## 9. MFC 时钟发生器

MFC 块在除了 HCLK 和 PCLK 外，还需要一个特殊的时钟。如图 3-8 所示，显示了这个特殊时钟的产生。

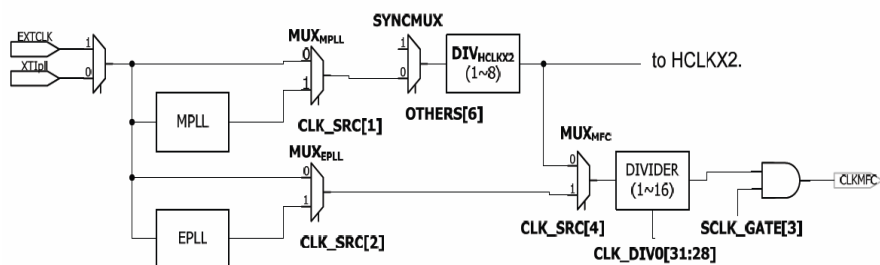


图 3-8 MFC 时钟发生器

时钟源在 HCLKX2 和 MOUTEPLL 之间进行选择。操作时钟使用 HCLKX2 进行分频。HCLKX2 的操作频率是固定的，默认为 266MHz。因此，CLK\_DIV0[31:28] 必须是 4' b0001 以产生 133MHz。当 MFC 不需要全性能时，有两种方法来减少操作频率。一种方法是当 CLK\_SRC[4] 设置为 1 时，使用 EPLL 输出时钟。通常，EPLL 是用于使音频时钟和输出时钟低于 MPPLL 的输出频率。另一种方法是调节时钟分频器 CLK\_DIV0[31:28] 的比例。使用此值，较低的频率可以应用到 MFC 块，使用 CLK\_SRC[4] 区域，以减少多余的功率耗散。因为 EPLL 的输出频率 HCLKX2 或 HCLK 是独立的。

## 10. Camera I/F 时钟发生器

如图 3-9 所示，显示用于 Camera 接口的时钟发生器。用于 Camera 接口的所有数据都是基于这个时钟来进行传输/接收的。最大操作时钟达到 133MHz。

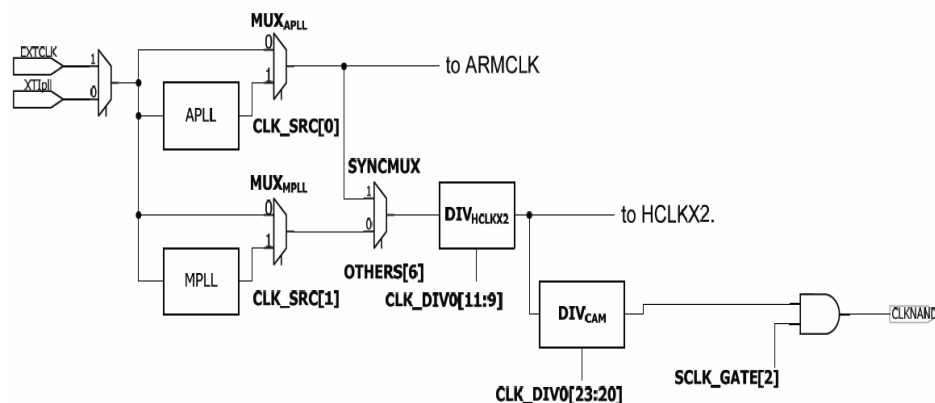


图 3-9 Camera I/F 时钟发生器

## 11. 显示时钟发生器（POST, LCD 和 scaler）

如图3-10所示，描述的是用于显示块的时钟发生器。通常LCD控制器需要的图像后处理器和定标器的逻辑。操作时钟可以独立地控制这个时钟发生器。CLKLCD 和 CLKPOST被连接到domain-F内的LCD 控制器和后处理器。CLKSCALER 是连接到domain-P内的定标器块。

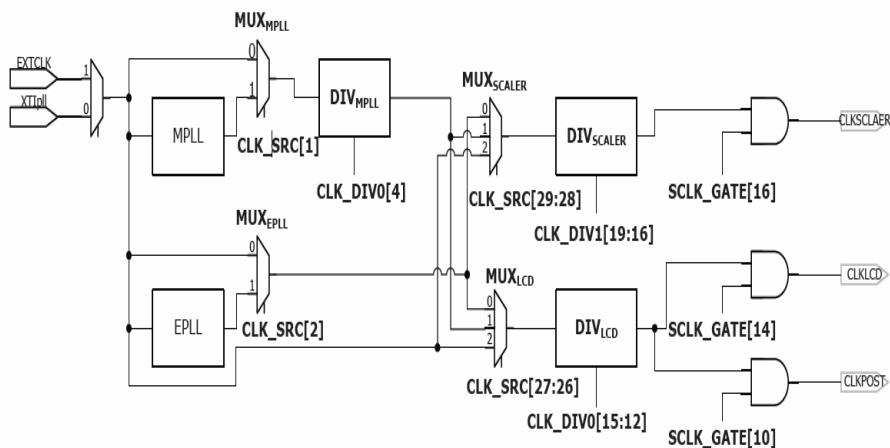


图 3-10 显示时钟发生器



## 12. 音频时钟发生器 (IIS 和 PCM)

如图 3-11 所示, 描述的是用于音频接口逻辑的特殊时钟发生器, 包括 IIS 和 PCM。S3C6410 有两个 IIS 通道和两个 PCM 通道。在任意时间, 它仅支持两个通道。一般来说, EPPLL 发生器是用于音频接口的一个特殊时钟。如果 S3C6410 要求两个独立的时钟频率, 如, 在两个音频接口之间不存在整数关系, 余下的时钟可以通过外部振荡器或使用 MPLL 来直接提供。

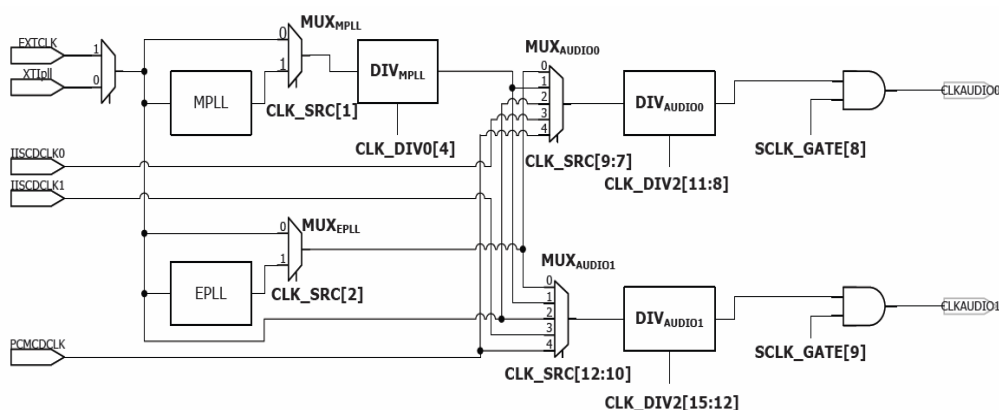


图 3-11 音频时钟发生器

## 13. 用于 UART, SPI 和 MMC 的时钟发生器

如图 3-12 所示, 描述的是用于 UART, SPI 和 MMC 的时钟发生器。有一个额外的时钟源 CLK27M, 给予了更多的灵活性。

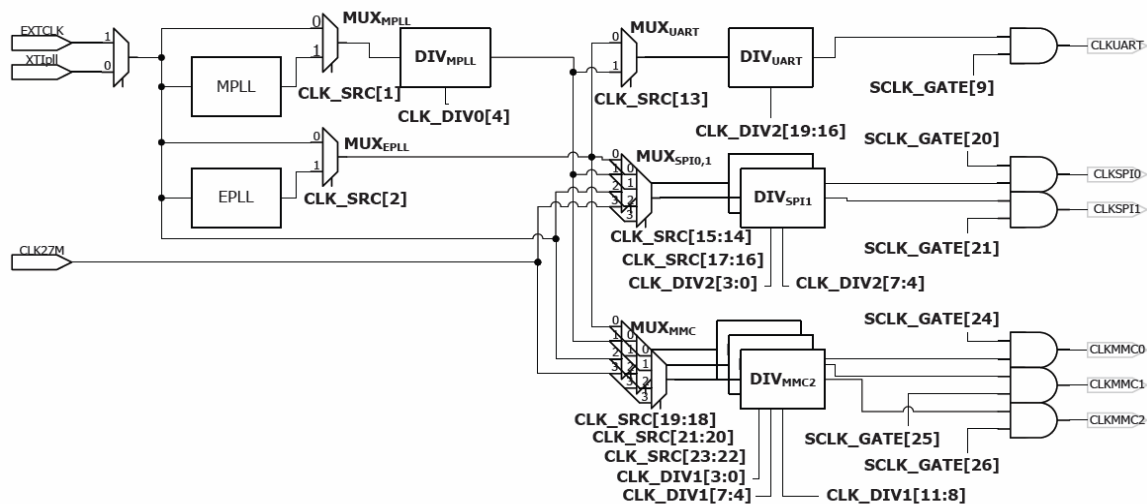


图 3-12 UART/SPI/MMC 时钟发生器

14. 用于 IrDA, USB host 时钟发生器

如图 3-13 所示，描述的是用于 IrDA 和 USB host 的时钟发生器。通常 USB 接口需要 48MHz 的操作时钟。

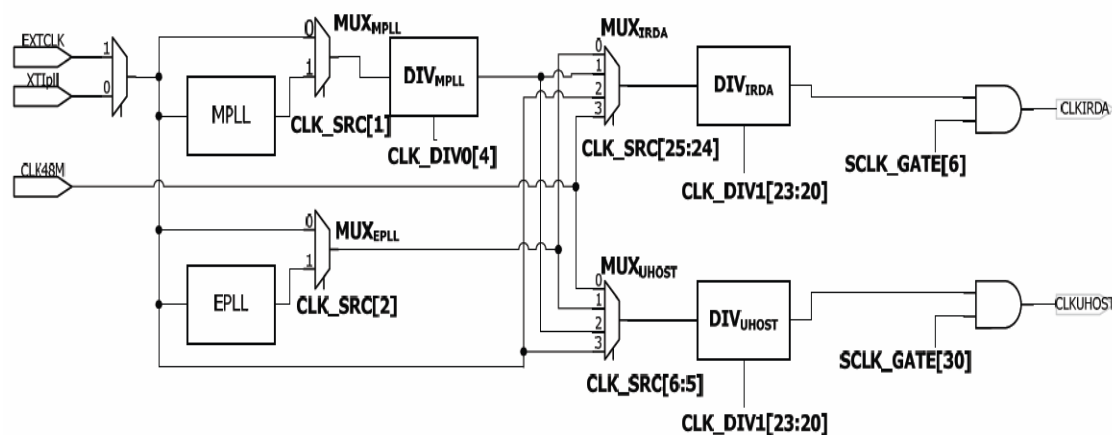


图 3-13 IrDA/USB host 时钟发生器

15. 时钟开/关控制

从以上的图中可以说明，HCLK\_GATE, PCLK\_GATE 和 SCLK\_GATE 控制时钟操作。如果一个位被设置，则通过每个时钟分频器相应的时钟将会被提供。否则，将被屏蔽。

HCLK\_GATE 控制 HCLK，用于每个 Ips。每个 IP 的 AHB 接口逻辑被独立地屏蔽，以减少动态电力消耗。PCLK\_GATE 控制 PCLK，用于每个 IP' s。某些 IP' s 需要特殊时钟正确的操作。通过 SCLK\_GATE 时钟被控制。

16. 时钟输出

S3C6410 时钟输出端口，产生内部时钟。这个时钟被用于正常的中断或调试用途。

3.3 低功率模式操作

S3C6410 通过低功率模式操作，支持低功率应用。如表 3-3 所示，总结了 S3C6410 的电源状态。它有四种电源状态，即正常状态、保持状态、电源选通状态和断电状态。所有的内部逻辑包括 F/Fs 和内存，其运行在正常状态下。保持状态在 STOP/DEEP-STOP 模式间减少不必要的电力消耗。然而，从 STOP/DEEP-STOP 模式中，保留预先的状态和支持快速唤醒时间。一些 DOMAIN-V, DOMAIN-I, DOMAIN-P, DOMAIN-F 和 DOMAIN-S

块没有状态保持特性。他们可以通过一个内部电源开关电路，利用电源选通以降低电力消耗。在睡眠模式下，外部调节器将会关闭，以减少电力消耗。S3C6410 最大限度地减少能量消耗和失去的所有信息，除了 ALIVE 和 RTC 块。

表 3-3 S3C6410 的四种电源状态

状态	外部调节器	内部 F/F	内部存储器
正常	ON	正常操作	正常操作
保持	ON	保留预先状态	保留预先状态
电源选通	ON	失去预先状态	失去预先状态
断电	OFF	失去预先状态	失去预先状态

1. S3C6410 中的电源域

S3C6410由几个电源域组成。子电源域，DOMAIN-V，DOMAIN-I，DOMAIN-P，DOMAIN-F和 DOMAIN-S是通过NORMAL\_CFG 和STOP\_CFG进行控制的。当S3C6410运行在正常或闲置模式时，由NORMAL\_CFG控制它们。如果控制位清除，相应的模块将改变电源门控模式和失去预先的状态。因此，用户软件在清除相应的位之前必须保存好内部状态。当S3C6410转换到STOP或DEEP-STOP模式时，子电源域自动的转换到电源门控模式。

STOP\_CFG仅控制ARM1176和top模块。如果用户软件要求快速相应时间，则ARM1176的内存和逻辑必须进行设置，同时在STOP模式间保留。在这种情况下，top模块的逻辑电源必须进行设置，同时top模块的内存电源也可以进行配置。另外，S3C6410可能不会返回到以前的状态了。当ARM1176电源关闭时（STOP\_CFG的位29和17为‘0’），ARM1176的泄漏电流可减至最低。这种配置就叫DEEP-STOP模式。进入DEEP-STOP模式之前，软件必须保留程序状态信息，包括内部寄存器，CPSR和SPSR等等。

2. 正常（NORMAL）/闲置（IDLE）模式

在正常模式下，ARM1176 内核，多媒体协控制器和所有外部设备都可以完全的运作。典型的系统总线操作频率可以达到 133MHz。每个多媒体协处理器和外设的时钟都可以进行选择性的停止，并通过软件去减少电源消耗。每个 IP 模块的个别时钟源，其时钟开/关门控的执行，主要是通过各自相应的时钟使能位来进行控制的。其使能位是通过 HCLK\_GATE，PCLK\_GATE 和 SCLK\_GATE 配置寄存器指定的。

在闲置模式下，ARM1176 的停止没有其他 IP’ s 的任何改变。通常情况下，ARM1176 等待一个唤醒事件，以回到正常模式下。

在正常/闲置模式下，所有的 IP’ s 都可运行在最大的操作频率上。当一些 IP’ s 没有要求运行时，

S3C6410 可以利用内部电源门控电路来切断电源的供应。如图 3-14 所示，五个电源域可以独立控制带有 NORMAL\_CFG 配置寄存器。当 IP’ s 的所有功能没有要求运行时，软件可以切断相应电源域的电源供应，如图 3-14 中灰色显示的部分。在相应的电源域关闭后，相应域的所有内部状态将消失。因此，用户软件必须保留所有要求恢复的内部状态信息。

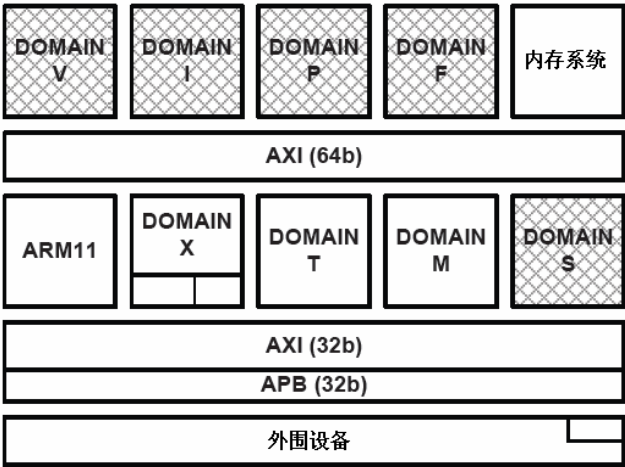


图 3-14 正常/闲置模式下的电源域

3. 停止（STOP）模式

在停止模式下，子电源域是作为黑色方框来表示的，用内部电源门控制电路关闭。如图 3-15 所示。其他模块是用灰色方框来表示的。ARM1176 保留以前的状态（保持状态）。因此，当外部唤醒事件发生时，内部状态在没有软件协助下也可以重新恢复。停止模式给予快速响应时间，但是需要一个小漏电流。

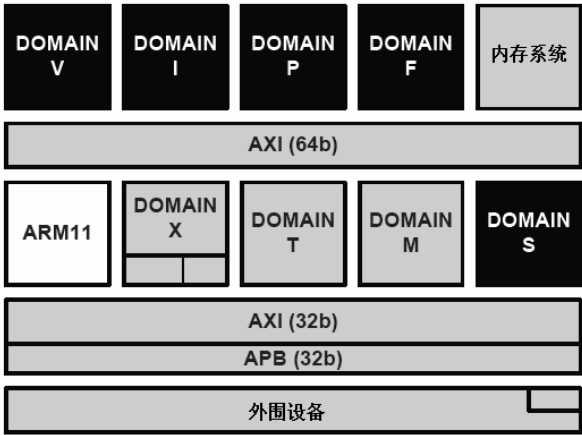


图 3-15 停止模式下的电源域

停止模式进入顺序如下：

- 1) 在停止模式下，用户软件设置 PWR\_CFG[6:5]。
- 2) 通过 MCR 指令 (MCR p15, 0, Rd, c7, c0, 4)，用户软件生成 STANDBYWFI 信号。
- 3) SYSCON 请求总线控制器，以完成当前 AHB 总线的事务处理。
- 4) 当前总线事务处理完成后，AHB 总线控制器发送到 SYSCON 进行确认。
- 5) SYSCON 请求 DOMAIN-V 以完成当前 AXI-总线的事务处理。
- 6) 当前总线事务处理完成后，AXI 总线控制器发送到 SYSCON 进行确认。
- 7) SYSCON 请求外部存储控制器进入自刷新模式，在停止模式期间外部内存的内容必须保存起来。
- 8) 自刷新模式时，存储控制器发送确认信息。
- 9) 如果 PLL 被使用，则 SYSCON 改变时钟源从 PLL 输出到外部振荡器。
- 10) SYSCON 禁用电源门控电路，以消除泄漏电流。
- 11) SYSCON 禁用 PLL 操作和晶体振荡器。

从停止模式到退出，除了正常中断以外，所有唤醒源都是可用的。停止模式下的唤醒顺序如下：

- 1) 在过渡期到正常模式间，SYSCON 声明 ARM1176 的复位信号。（仅应用于 DEEP-STOP 模式）。
- 2) SYSCON 使能晶体振荡器，等待振荡器稳定周期，它是通过 OSC\_STABLE 来配置的。
- 3) SYSCON 使能时钟门控电路，以提供操作电源和等待稳定时间，它是通过 MTC\_STABLE 来配置的。  
(仅应用于 DEEP-STOP 模式)
- 4) SYSCON 启动 PLL 逻辑和等待 PLL 锁周期，它是通过 A/M/EPLL\_LOCK 来配置的。
- 5) 如果 PLL 被使用，则 SYSCON 从外部振荡器到 PLL 输出改变时钟源。
- 6) SYSCON 释放自我刷新模式，请求到内存控制器。
- 7) 当准备就绪时，内存控制器发送确认信息。
- 8) SYSCON 释放对 AXI/AHB 总线的请求。
- 9) SYSCON 释放 ARM1176 的复位信号。（仅应用于 DEEP-STOP 模式）

#### 4. 深度停止 (DEEP-STOP) 模式

低功耗状态下，大多数移动应用需要比较长的待机周期和合理的响应时间。DEEP-STOP 模式集中用于需求。外部电源的开/关控制通常需要较长的转换时间 (~3ms)。当启动设备是 NAND 时，启动代码已经加载到 stepping-stone 中，在 DEEP-STOP 模式期间保留。启动代码的复制期可以忽略不计。

如图 3-16 所示，显示了 DEEP-STOP 模式下的状态。黑色方框表示电源门控模块，在 DEEP-STOP 模式期间消除漏电流。top 模块在 STOP 模式下保留以前的状态。

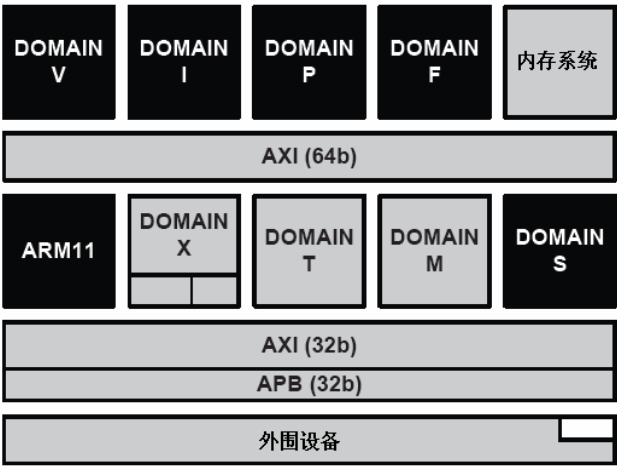


图 3-16 DEEP-STOP 模式下的电源域

进入和退出顺序类似于 STOP 模式，参考 STOP 模式顺序，用于 DEEP-STOP 模式的进入和退出顺序。

5. 睡眠（SLEEP）模式

在睡眠模式下，除了 ALIVE 和 RTC 模块之外，所有硬件逻辑都是利用外部电源调节器关闭电源的。睡眠模式支持的待机周期时间是最长的，用户软件必须保存所有内部状态到外部存储设备。ALIVE 模块等待一个外部唤醒事件，RTC 保存时间信息。用户软件可配置唤醒源，I/O 引脚的状态用 GPIO 来配置。

睡眠模式下的电源域，如图 3-17 所示。

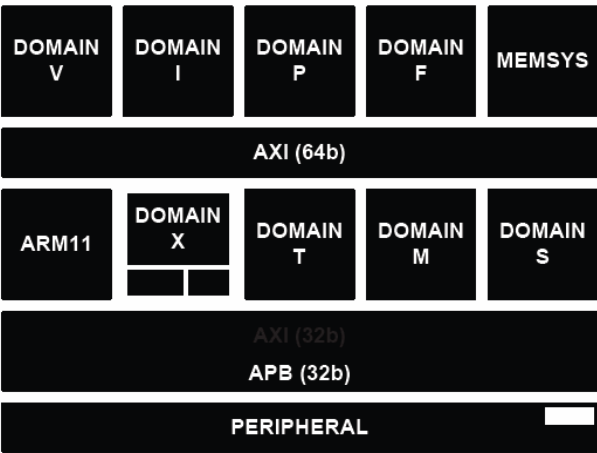


图 3-17 睡眠模式下的电源域

睡眠模式进入顺序如下：

- 1) 在 SLEEP 模式下，用户软件设置 PWR\_CFG[6:5]。
- 2) 通过 MCR 指令（MCR p15, 0, Rd, c7, c0, 4），用户软件生成 STANDBYWFI 信号。
- 3) SYSCON 请求总线控制器，以完成当前 AHB 总线的处理。
- 4) 当前总线的处理完成后，AHB 总线控制器发送确认信息到 SYSCON。
- 5) SYSCON 请求 DOMAIN-V 去完成当前 AXI 总线的处理。
- 6) 当前总线处理完成后，AXI 总线控制器发送确认信息到 SYSCON。
- 7) SYSCON 请求外部存储控制器进入到自刷新模式，在 SLEEP 模式期间外部内存的内容必须要保存起来。
- 8) 自刷新模式时，存储控制器发送确认信息。
- 9) 如果 PLL 被使用，则 SYSCON 改变时钟源从 PLL 输出到外部振荡器。
- 10) SYSCON 禁用 PLL 操作和晶体振荡器。
- 11) 最后，通过声明 XPWRRGTON 引脚到低电平状态，SYSCON 禁用外部时钟源用于内部逻辑。XPWRRGTON 信号控制外部调节器。

其退出顺序如下：

- 1) SYSCON 启动外部电源，通过声明 XPWRRGTON 引脚到高电平状态，通过 PWR\_STABLE 配置等待稳定时间。
- 2) SYSCON 生成系统时钟，包括 HCLK, PCLK 和 ARMCLK。
- 3) SYSCON 释放系统复位信号，包括 HRESETn 和 PRESETn。
- 4) 如果启动设备是 NAND，则 NFCON 从外部 NAND 设备到 stepping stone 复制启动代码。(XOM[4:3] == 2' b00)
- 5) SYSCON 释放 ARM 复位信号。

## 6. 唤醒

如表 3-4 所示，说明了从低功耗状态，IDLE，(DEEP)-STOP 和 SLEEP 模式中的各种唤醒源。根据低功耗状态，可用不同的唤醒源。

表 3-4 唤醒源的电源模式

电源模式	唤醒源
	所有中断源

闲置模式	停止模式	睡眠模式	MMC0, MM1, MMC2
			TS ADC
			外部中断源
			RTC 警告
			TICK
			键盘中断
			MSM (MODEM)
			电池故障
			HSI
			温复位

## 7. 复位

S3C6410 有五种类型的复位信号，SYSCON 可以把系统的五分之一进行复位。

- 硬件复位：它是通过声明 XnRESET 产生的。它可以完全初始化所有系统。
- 温复位：它是通过 XnWRESET 产生的。当需要初始化 S3C6410 和保存当前硬件状态时，XnWRESET 被使用。
- 看门狗复位：它是通过一个特殊的硬件模块产生的，也就是看门狗定时器。当系统发生一个不可预测的软件错误时，硬件模块监控内部硬件状态，同时产生复位信号来脱离该状态。
- 软件复位：它是通过设置 SW\_RESET 产生的。
- 唤醒复位：它是当 S3C6410 从睡眠模式唤醒时产生的。睡眠模式后，内部硬件状态在任何时候都不可用，必须对其进行初始化。

### 硬件复位

当 XnRESET 引脚被声明，系统内的所有单元（除了 RTC 之外）复位到预先定义好的状态时，硬件复位被调用。在这段期间，将发生下面的动作：

- 所有内部寄存器和 ARM1176 内核都到预先定义好的复位状态。
- 所有引脚都得到它们的复位状态。
- 当 XnRESET 被声明的同时，XnRSTOUT 引脚就被声明了。

XnRESET 是不被屏蔽的，始终保持使能状态。XnRESET 的声明，无论先前为何模式，S3C6410 都进入复位状态。XnRESET 必须持有足够长的时间允许内部稳定和传播。



S3C6410 的电源调节器必须预先稳定到 XnRESET. 的 deassertion 状态。否则，它可能会损害 S3C6410，发生不可预测的操作。

温复位

当在正常，闲置和停止模式下，为了超过 100ns，XnWRESET 引脚被声明时，温复位被调用。在睡眠模式下，它是作为一个唤醒事件被处理的。如果 XnBATFLT 保持低电平或系统处于唤醒时期，则 XnWRESET 被忽略。如图 3-18 所示，所有寄存器除了 SYSCON，RTC 和 GPIO 都被初始化。

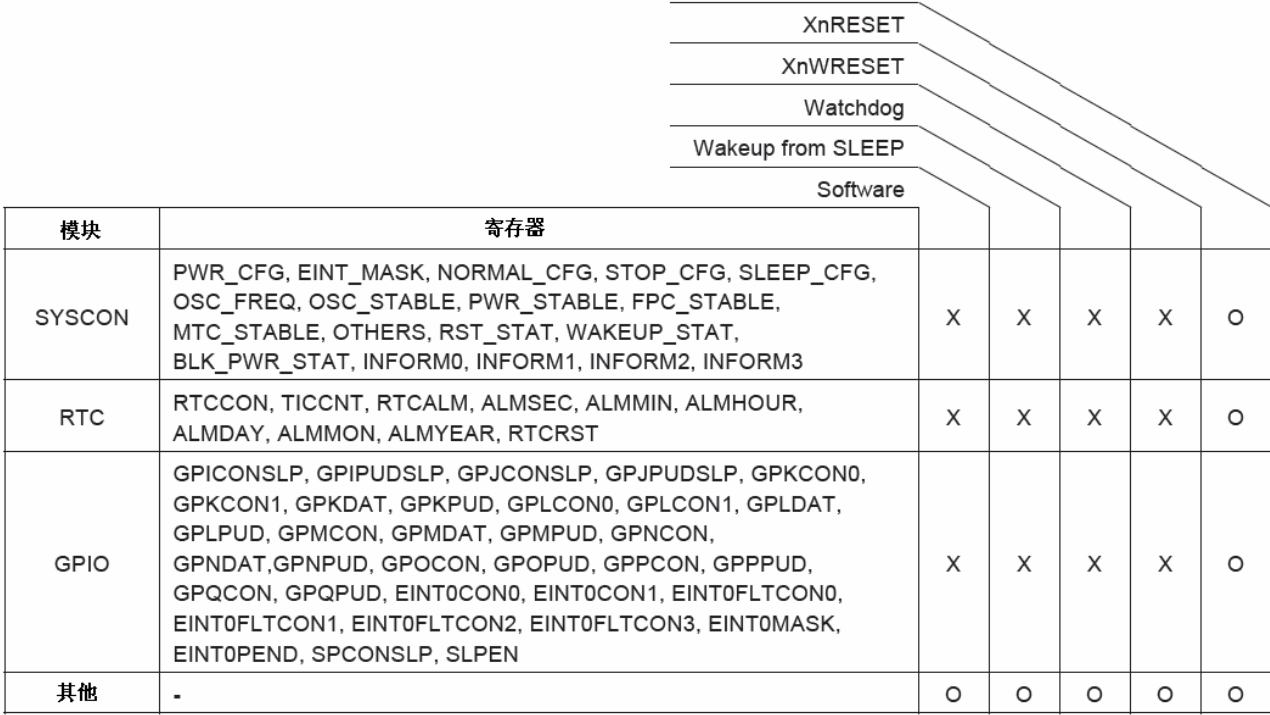


图 3-18 寄存器初始化的各种复位

在温复位期间，将发生以下的动作：

- 所有模块除了 ALIVE 和 RTC 模块之外，都到预先定义好的复位状态。
- 所有引脚进入复位状态。
- 在看门狗复位期间，nRSTOUT 引脚被声明。

当 XnWRESET 信号被声明为 ‘0’，下列依次发生：

- 1) SYSCON 请求 AHB 总线控制器，以完成当前 AHB 总线的处理。
- 2) 在当前总线处理完成之后，AHB 总线控制器发送确认信息到 SYSCON。
- 3) SYSCON 请求 DOMAIN-V，以完成当前 AXI 总线处理。

- 4) 在当前总线处理完成后，AXI 总线控制器发送确认信息到 SYSCON。
- 5) SYSCON 请求外部存储控制器进入到自刷新模式，当温复位被声明时，外部内存的内容必须被保存。
- 6) 当自刷新模式时，存储控制器发送确认信息。
- 7) SYSCON 声明内部复位信号和 XnRSTOUT. 。

温复位模块，在具体 ARM11 处理器中，代码实现如下：

```
void Test_WarmReset(void)
{
    u32 uRstId;
    u32 uInform0, uInform1;
    uRstId = SYSC_RdRSTSTAT(1);

    // Check Alive Reg.
    // Alive Register
    uInform0 = 0x01234567;
    uInform1 = 0x6400ABCD;

    // For Test
    //WDT_operate(1, 0, 0, 1, 100, 15625, 15625);
    if( ( uRstId == 1 ) && !(g_OnTest) )
    {
        printf("Warm Reset- Memory data check \n");
        CheckData_SDRAM(_DRAM_BaseAddress+0x1000000, 0x10000);

        //Check Information Register Value
        if( (uInform0 !=Inp32Inform(0) ) || (uInform1 != Inp32Inform(1)))
        {
            printf(" Information Register Value is wrong!!! \n");
        }
        else
```

```

{
    printf(" Information Register Value is correct!!! \n");
}

printf("Warm Reset test is done\n");
g_OnTest = 1;
SYSC_BLKPrONAll();
Delay(10);
SYSC_RdBLKPWR();
}
else
{
    printf("[WarmReset Test]\n");
    InitData_SDRAM(_DRAM_BaseAddress+0x1000000, 0x10000);

    // Alive Register Write
    Outp32Inform(0, uInform0);
    Outp32Inform(1, uInform1);

    //Added case : bus power down
    SYSC_BLKPrOffAll();
    printf("HCLKGATE: 0x%x\n", Inp32(0x7E00F030));
    printf("Now, Push Warm Reset Botton. \n");
    while(1)
    {
        // test case
        DMAC_InitCh(DMA0, DMA_ALL, &oDmac_0);
        DMAC_InitCh(DMA1, DMA_ALL, &oDmac_1);
        INTC_SetVectAddr(NUM_DMA0, Dma0Done_Test);
        INTC_SetVectAddr(NUM_DMA1, Dma1Done_Test);
    }
}

```

```

    INTC_Enable(NUM_DMA0);
    INTC_Enable(NUM_DMA1);

    g_DmaDone0=0;
    g_DmaDone1=0;
    printf("DMA Start \n");
    // 16MB
    DMACH_Setup(DMA_A, 0x0, 0x51f00000, 0, 0x51f01000, 0, WORD, 0x1000000, DEMAND, MEM,
                MEM, BURST4, &oDmac_0);
    DMACH_Setup(DMA_A, 0x0, 0x52000000, 0, 0x52001000, 0, WORD, 0x1000000, DEMAND, MEM,
                MEM, BURST4, &oDmac_1);

    // Enable DMA
    DMACH_Start(&oDmac_0);
    DMACH_Start(&oDmac_1);
    while((g_DmaDone0==0) || (g_DmaDone1==0))    // Int.
    {
        Copy(0x51000000, 0x51800000, 0x1000000);
    }
}
}
}

```

## 软件复位

当利用软件将 0x6410 写入到 SW\_RST 时，软件复位被调用。行为与温复位的情况相同。

软件复位在 ARM11 处理器中，代码实现如下：

```

void Test_SoftReset(void)
{
    u32 uRstId;

```

```

u32 uInform0, uInform1;

printf("rINFORM0: 0x%x\n", Inp32Inform(0));
printf("rINFORM1: 0x%x\n", Inp32Inform(1));
uInform0 = 0xABCD6400;
uInform1 = 0x6400ABCD;
uRstId = SYSC_RdRSTSTAT(1);
SYSC_RdBLKPWR();

if( ( uRstId == 5 ) && !(g_OnTest) )
{
    printf("Software Reset- Memory data check \n");
    CheckData_SDRAM(_DRAM_BaseAddress+0x1000000, 0x10000);

    //Check Information Register Value
    if( (uInform0 !=Inp32Inform(0) ) || (uInform1 != Inp32Inform(1)))
    {
        printf(" Information Register Value is wrong!!! \n");
    }
    else
    {
        printf(" Information Register Value is correct!!! \n");
    }
    printf("software reset test is done\n");
    g_OnTest = 1;
    SYSC_BLKPrONAll();
    Delay(10);
    SYSC_RdBLKPWR();
}

```

```

else
{
    printf("[SoftReset Test]\n");
    InitData_SDRAM(_DRAM_BaseAddress+0x1000000, 0x10000);

    //Added case : bus power down
    SYSC_BLKPwrOffAll();

    // Added case : Clock Off Case
    // Outp32SYSC(0x30, 0xFDDFFFFE); //IROM, MEM0, MFC
    // Outp32SYSC(0x30, 0xFFFFFFF0); // MFC, MFC Block OFF OK
    // Outp32SYSC(0x30, 0xFDDFFFFF); // IROM OK
    // Outp32SYSC(0x30, 0xFFDFFFFF); // MEM0
    printf("HCLKGATE: 0x%x\n", Inp32(0x7E00F030));
    // Alive Register Write
    Outp32Inform(0, uInform0);
    Outp32Inform(1, uInform1);
    //Outp32(0x7F008880, 0x1000);

    UART_TxEmpty();
    printf("Now, Soft Reset causes reset on 6410 except SDRAM. \n");
    SYSC_SWRST();
    while(!UART_GetKey());
}
}

```

## 看门狗复位

当软件挂起时，看门狗复位被调用。因此，在用于看门狗复位的 WDT 和 WDT 超时信号里，软件不能初始化寄存器。在看门狗复位期间，有以下动作发生：

- 除了 ALIVE 和 RTC 模块，所有模块进入预先定义好的复位状态。
- 所有引脚都进入复位状态。
- 在看门狗复位期间，nRSTOUT 引脚被声明。

在正常模式和闲置模式下，看门狗可被激活，并可产生超时信号。当看门狗定时器和复位使能时，其被调用。因此，下列依次发生：

- 1) WDT 产生超时信号。
- 2) SYSCON 调用复位信号，初始化内部 IP。
- 3) 包括 nRSTOUT 的复位被声明，直到复位计数器 RST\_STABLE 被终止。

看门狗复位在具体 ARM11 处理器中，代码实现如下：

```
void Test_WDTReset(void)
{
    printf("[WatchDog Timer Reset Test]\n");
    INTC_Enable(NUM_WDT);

    // 1. Clock division_factor 128
    printf("\nClock Division Factor: 1(dec), Prescaler: 100(dec)\n");
    // WDT reset enable
    printf("\nI will restart after 2 sec.\n");
    WDT_operate(1, 1, 0, 1, 100, 15625, 15625);

    //Test Case - add SUB Block Off
    SYSC_BLKPwrOffAll();

    //Added case : Clock Off Case
    Outp32(0x7E00F030, 0xFDDFFFE);
    // Outp32SYSC(0x30, 0xFFFFFEE);    // MFC, MFC Block OFF OK
    // Outp32SYSC(0x30, 0xFDFFFFFF);    // IROM OK
    // Outp32SYSC(0x30, 0xFFDFFFFF);    // MEMO
    printf("HCLKGATE: 0x%x\n", Inp32(0x7E00F030));
}
```

```

//while(!UART_GetKey());
// Test Case - add Bus operation
while(1)
{
    // test case
    DMAC_InitCh(DMA0, DMA_ALL, &oDmac_0);
    DMAC_InitCh(DMA1, DMA_ALL, &oDmac_1);
    INTC_SetVectAddr(NUM_DMA0, Dma0Done_Test);
    INTC_SetVectAddr(NUM_DMA1, Dma1Done_Test);
    INTC_Enable(NUM_DMA0);
    INTC_Enable(NUM_DMA1);

    g_DmaDone0=0;
    g_DmaDone1=0;

    printf("DMA Start \n");
    // 16MB
    DMACH_Setup(DMA_A, 0x0, 0x51f00000, 0, 0x51f01000, 0, WORD, 0x1000000, DEMAND, MEM,
                MEM, BURST4, &oDmac_0);
    DMACH_Setup(DMA_A, 0x0, 0x52000000, 0, 0x52001000, 0, WORD, 0x1000000, DEMAND, MEM,
                MEM, BURST4, &oDmac_1);

    // Enable DMA
    DMACH_Start(&oDmac_0);
    DMACH_Start(&oDmac_1);

    while((g_DmaDone0==0) || (g_DmaDone1==0))    // Int.
    {

```



```

Copy(0x51000000, 0x51800000, 0x1000000);

}

}

//INTC_Disable(NUM_WDT);

}

```

### 唤醒复位

当 S3C6410 通过一个唤醒事件，从睡眠模式唤醒时，唤醒复位被调用。

## 3.4 寄存器描述

系统控制器控制 PLL，时钟发生器，电源管理部分和其他系统部分。本节描述了在系统控制器内，如何使用的 SFR（特殊功能寄存器）来控制这些部分。

### 存储器映射

以下突出系统控制器内的 34 个寄存器。

寄存器	地址	读/写	描述	复位值
APLL_LOCK	0x7E00_F000	读/写	控制 PLL 锁定期 APLL。	0x0000_FFFF
MPLL_LOCK	0x7E00_F004	读/写	控制 PLL 锁定期 MPLL。	0x0000_FFFF
EPLL_LOCK	0x7E00_F008	读/写	控制 PLL 锁定期 EPLL。	0x0000_FFFF
APLL_CON	0x7E00_F00C	读/写	控制 PLL 输出频率 APLL。	0x0190_0302
MPLL_CON	0x7E00_F010	读/写	控制 PLL 输出频率 MPLL。	0x0214_0603
EPLL_CON0	0x7E00_F014	读/写	控制 PLL 输出频率 EPLL。	0x0020_0102
EPLL_CON1	0x7E00_F018	读/写	控制 PLL 输出频率 EPLL。	0x0000_9111
CLK_SRC	0x7E00_F01C	读/写	选择时钟源。	0x0000_0000
CLK_DIV0	0x7E00_F020	读/写	设置时钟分频器的比例。	0x0105_1000
CLK_DIV1	0x7E00_F024	读/写	设置时钟分频器的比例。	0x0000_0000
CLK_DIV2	0x7E00_F028	读/写	设置时钟分频器的比例。	0x0000_0000
CLK_OUT	0x7E00_F02C	读/写	选择时钟输出。	0x0000_0000

HCLK_GATE	0x7E00_F030	读/写	控制 HCLK 时钟选通。	0xFFFF_FFFF
PCLK_GATE	0x7E00_F034	读/写	控制 PCLK 时钟选通。	0xFFFF_FFFF
SCLK_GATE	0x7E00_F038	读/写	控制 SCLK 时钟选通。	0xFFFF_FFFF
RESERVED	0x7E00_F03C~ 0x7E00_F0FC	—	保留。	—
AHB_CON0	0x7E00_F100	读/写	配置 AHB I/P/X/F 总线。	0x0400_0000
AHB_CON1	0x7E00_F104	读/写	配置 AHB M1/M0/T1/T0 总线。	0x0000_0000
AHB_CON2	0x7E00_F108	读/写	配置 AHB R/S1/S0 总线。	0x0000_0000
RESERVED	0x7E00_F10C	—	保留。	—
SDMA_SEL	0x7E00_F110	读/写	选择安全 DMA 输入。	0x0000_0000
SW_RST	0x7E00_F114	读/写	产生软件复位。	0x0000_0000
SYS_ID	0x7E00_F118	读	系统 ID 版本和审查通过。	0x0000_0000
RESERVED	0x7E00_F11C	—	保留。	—
MEM_SYS_CFG	0x7E00_F120	读/写	配置存储器子系统。	0x0000_0080
QOS_OVERRIDE0	0x7E00_F124	读/写	取代 DMC0 QOS。	0x0000_0000
QOS_OVERRIDE1	0x7E00_F128	读/写	取代 DMC1 QOS。	0x0000_0000
MEM_CFG_STAT	0x7E00_F12C	读	存储器子系统建立状态。	0x0000_0000
RESERVED	0x7E00_F200~ 0x7E00_F800	—	保留。	—
PWR_CFG	0x7E00_F804	读/写	配置电源管理。	0x0000_0001
EINT_MASK	0x7E00_F808	读/写	配置 EINT(外部中断)屏蔽。	0x0000_0000
RESERVED	0x7E00_F80C	—	保留。	—
NORMAL_CFG	0x7E00_F810	读/写	在正常模式下，配置电源管理。	0xFFFF_FF00
STOP_CFG	0x7E00_F814	读/写	在停止模式下，配置电源管理。	0x2012_0100
SLEEP_CFG	0x7E00_F818	读/写	在睡眠模式下，配置电源管理。	0x0000_0000

RESERVED	0x7E00_F81C	–	保留。	–
OSC_FREQ	0x7E00_F820	读/写	振荡器频率刻度计数器。	0x0000_000F
PWR_STABLE	0x7E00_F828	读/写	电源稳定计数器。	0x0000_0001
RESERVED	0x7E00_F82C	–	保留。	–
MTC_STABLE	0x7E00_F830	读/写	MTC 稳定计数器。	0xFFFF_FFFF
RESERVED	0x7E00_F834~ 0x7E00_F8FC	–	保留。	–
OTHERS	0x7E00_F900	读/写	其他控制寄存器。	0x0000_801E
RST_STAT	0x7E00_F904	读	复位状态寄存器。	0x0000_0001
WAKEUP_STAT	0x7E00_F908	读/写	唤醒状态寄存器。	0x0000_0000
BLK_PWR_STAT	0x7E00_F90C	读	块电源状态寄存器。	0x0000_007F
INFORM0	0x7E00_FA00	读/写	信息寄存器 0。	0x0000_0000
INFORM1	0x7E00_FA04	读/写	信息寄存器 1。	0x0000_0000
INFORM2	0x7E00_FA08	读/写	信息寄存器 2。	0x0000_0000
INFORM3	0x7E00_FA0C	读/写	信息寄存器 3。	0x0000_0000

SFR 由五部分组成。SFR 的地址为 0x7E00\_F0XX，控制 PLL 和时钟发生器。控制三个 PLL 的输出频率，时钟源选择和时钟分频器的比例。SFRs 的地址为 0x7E00\_F1XX，控制总线系统，内存系统和软件复位。SFRs 的地址为 0x7E00\_F8XX，控制电源管理模块。SFRs 的地址为 0x7E00\_F9XX，显示内部状态。消息寄存器的地址为 0x7E00\_FA0X，保留用户信息，直到硬件复位信号 (XnRESET) 被声明。

下面主要针对个别的寄存器进行描述。

### 3.4.1. PLL 控制寄存器

S3C6410 有三个内部 PLL，分别是 APLL，MPLL 和 EPLL。它们通过以下所示七个特殊寄存器进行控制。

寄存器	地址	读/写	描述	复位值
APLL_LOCK	0x7E00_F000	读/写	控制 PLL 锁定期 APLL。	0x0000_FFFF
MPLL_LOCK	0x7E00_F004	读/写	控制 PLL 锁定期 MPLL。	0x0000_FFFF

EPLL_LOCK	0x7E00_F008	读/写	控制 PLL 锁定期 EPLL。	0x0000_FFFF
APLL_CON	0x7E00_F00C	读/写	控制 PLL 输出频率 APLL。	0x0190_0302
MPLL_CON	0x7E00_F00C	读/写	控制 PLL 输出频率 MPLL。	0x0214_0603
EPLL_CON0	0x7E00_F00C	读/写	控制 PLL 输出频率 EPLL。	0x0020_0102
EPLL_CON1	0x7E00_F00C	读/写	控制 PLL 输出频率 EPLL。	0x0000_9111

当输入频率被改变或是分频值被改变时，PLL 要求锁周期。PLL\_LOCK 寄存器指定的这个锁周期是基于 PLL 的时钟源。在这个周期，输出将被屏蔽为 ‘0’ 。

APLL_LOCK / MPLL_LOCK / EPLL_LOCK	位	描述	初始状态
RESERVED	[31:16]	保留。	0x0000
PLL_LOCKTIME	[15:0]	在规定期间内产生一个稳定的时钟输出。	0xFFFF

PLL\_CON 寄存器控制每个 PLL 的操作。如果 ENABLE 位被设置，相应的 PLL 发生输出后 PLL 锁定周期。PLL 的输出频率是通过 MDIV，PDIV，SDIV 和 KDIV 的值进行控制的。

APLL_CON / MPLL_CON	位	描述	初始状态
ENABLE	[31]	PLL 使能控制（0：禁用，1：使能）。	0
RESERVED	[30:26]	保留。	0x00
MDIV	[25:16]	PLL 的 M 分频值。	0x190 / 0x214
RESERVED	[15:14]	保留。	0x0
PDIV	[13:8]	PLL 的 P 分频值。	0x3 / 0x6
RESERVED	[7:3]	保留。	0x00
SDIV	[2:0]	PLL 的 S 分频值。	0x2 / 0x3

如果输入时钟频率是 12MHz，则 APLL\_CON / MPLL\_CON 的复位值分别产生 400MHz 和 133MHz 的输出时钟。

**注：**

使用以下公式进行输出频率的计算：

$$F_{OUT} = M_{DIV} \times F_{IN} / (P_{DIV} \times 2^{SDIV})$$

这里，用于 APLL 和 MPLL 的 M<sub>DIV</sub>，P<sub>DIV</sub>，S<sub>DIV</sub> 必须符合以下条件：

$$M_{DIV}: 56 \leq M_{DIV} \leq 1023$$

$$P_{DIV}: 1 \leq P_{DIV} \leq 63$$

$$S_{DIV}: 0 \leq S_{DIV} \leq 5$$

$$F_{VCO} (=M_{DIV} \times F_{IN} / P_{DIV}): 1000MHz \leq F_{VCO} \leq 1600MHz$$

$$F_{OUT}: 31.25MHz \leq F_{VCO} \leq 1600MHz$$

EPLL_CON0	位	描述	初始状态
ENABLE	[31]	PLL 使能控制（0：禁用，1：使能）。	0
RESERVED	[30:24]	保留。	0x00
M <sub>DIV</sub>	[23:16]	PLL 的 M 分频值。	0x20
RESERVED	[15:14]	保留。	0x0
P <sub>DIV</sub>	[13:8]	PLL 的 P 分频值。	0x1
RESERVED	[7:3]	保留。	0x00
S <sub>DIV</sub>	[2:0]	PLL 的 S 分频值。	0x2

EPLL_CON1	位	描述	初始状态
RESERVED	[31:16]	保留。	0x0000
K <sub>DIV</sub>	[15:0]	PLL 的 K 分频值。	0x9111

如果输入时钟频率是 12MHz，EPLL\_CON0 / EPLL\_CON1 的复位值分别产生 97.70MHz 的输出时钟。

**注：**

使用以下公式进行输出频率的计算：

$$F_{OUT} = (M_{DIV} + K_{DIV} / 2^{16}) \times F_{IN} / (P_{DIV} \times 2^{SDIV})$$

这里，用于 APLL 和 MPLL 的 M<sub>DIV</sub>，P<sub>DIV</sub>，S<sub>DIV</sub> 必须符合以下条件：

$$M_{DIV}: 13 \leq M_{DIV} \leq 255$$

$$P_{DIV}: 1 \leq P_{DIV} \leq 63$$

$KDIV: 0 \leq KDIV \leq 65535$

$SDIV: 0 \leq SDIV \leq 5$

$FVCO (= (MDIV + KDIV / 2^{16}) \times FIN / PDIV) : 250MHz \leq FVCO \leq 600MHz$

$FOUT : 16MHz \leq FOUT \leq 600MHz$

3.4.2. 时钟源控制寄存器

S3C6410 有很多时钟源，从 GPIO 配置中，包括三个 PLL 输出，外部振荡器，外部时钟和其他时钟源。CLK\_SRC 寄存器控制每个时钟分频器的时钟源。如图 3-19 所示。

寄存器	地址	读/写	描述	复位值
CLK_SRC	0x7E00_F01C	读/写	选择时钟源。	0x0000_0000

CLK_SRC	位	描述	初始状态
TV27_SEL	[31]	控制MUXTV27，它是TV27MHz的时钟源。 (0: 27MHz, 1: FINEPLL)	0
DAC27_SEL	[30]	控制MUXDAC27，它是DAC27MHz的时钟源。 (0: 27MHz, 1: FINEPLL)	0
SCALER_SEL	[29:28]	控制MUXSCALER，它是TVSCALER的时钟源。 (00: MOUTEPLL, 01: DOUTMPLL, 10: FINEPLL)	0x0
LCD_SEL	[27:26]	控制MUXLCD，它是LCD的时钟源。 (00: MOUTEPLL, 01: DOUTMPLL, 10: FINEPLL)	0x0
IRDA_SEL	[25:24]	控制MUXIRDA，它是IRDA的时钟源。 (00: MOUTEPLL, 01: DOUTMPLL, 10: FINEPLL, 11: 48MHz)	0x0
MMC2_SEL	[23:22]	控制MUXMMC2，它是MMC2的时钟源。 (00: MOUTEPLL, 01: DOUTMPLL, 10: FINEPLL, 11: 27MHz)	0x0
MMC1_SEL	[21:20]	控制MUXMMC1，它是MMC1的时钟源。 (00: MOUTEPLL, 01: DOUTMPLL, 10: FINEPLL, 11: 27MHz)	0x0

		27MHz)	
MMCO_SEL	[19:18]	控制MUXMMCO，它是MMCO的时钟源。 (00: MOUTEPLL, 01: DOUTMPLL, 10: FINEPLL, 11: 27MHz)	0x0
SPI1_SEL	[17:16]	控制MUXSPI1，它是SPI1的时钟源。 (00: MOUTEPLL, 01: DOUTMPLL, 10: FINEPLL, 11: 27MHz)	0x0
SPI0_SEL	[15:14]	控制MUXSPI0，它是SPI0的时钟源。 (00: MOUTEPLL, 01: DOUTMPLL, 10: FINEPLL, 11: 27MHz)	0x0
UART_SEL	[13]	控制MUXUART0，它是UART的时钟源。 (0: MOUTEPLL, 1: DOUTMPLL)	0
AUDIOI1_SEL	[12:10]	控制MUXAUDIOI1，它是IIS1，PCM1和AC97 1的时钟源。 (000: MOUTEPLL, 0 01: DOUTMPLL, 010: FINEPLL, 011: IISCDCLK1, 100: PCMCDCLK)	0x0
AUDIOO0_SEL	[9:7]	控制MUXAUDIOO0，它是IIS0，PCM0和AC97 0的时钟源。 (000: MOUTEPLL, 001: DOUTMPLL, 010: FINEPLL, 011: IISCDCLK0, 10x: PCMCDCLK)	0x0
UHOST_SEL	[6:5]	控制MUXUHOST，它是USB Host的时钟源。 (00: 48MHz, 01: MOUTEPLL, 10: DOUTMPLL, 11: FINEPLL)	0x0
MFCCLK_SEL	[4]	控制MUXMFC，它是MFC的时钟源。	0
RESERVED	[3]	保留。	0
EPLL_SEL	[2]	控制 MUXEPLL (0: FINEPLL, 1: FOUTEPLL)。	0
MPLL_SEL	[1]	控制 MUXMPLL (0: FINMPLL, 1: FOUTMPLL)。	0

APLL_SEL	[0]	控制 MUXAPLL (0: FINAPLL, 1: FOUTAPLL)。	0
----------	-----	---------------------------------------	---

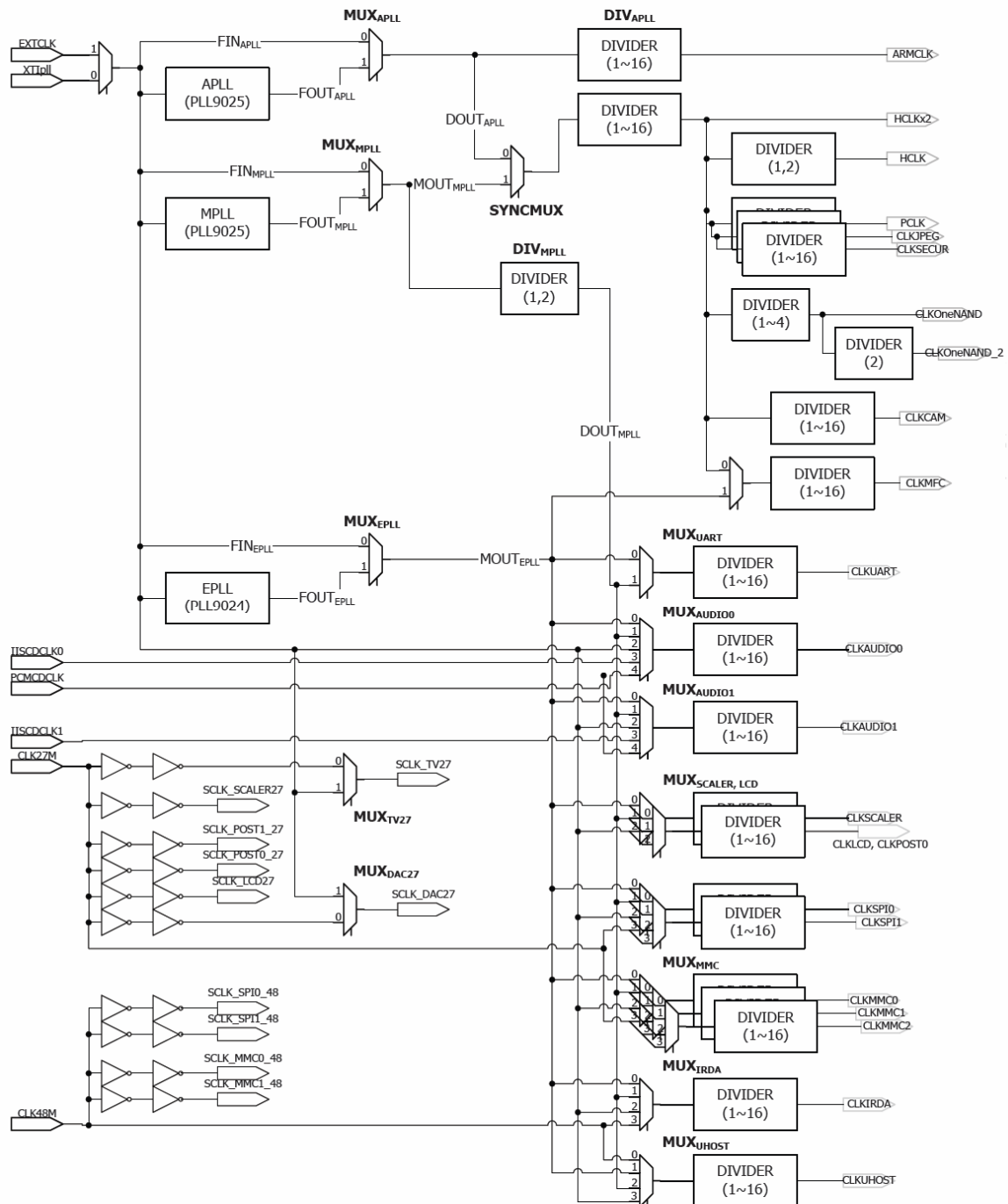


图 3-19 时钟源控制寄存器



### 3.4.3. 时钟分频控制寄存器

S3C6410 有几个时钟分频器，以支持各种操作的时钟频率。时钟分频器的比例可以通过 CLK\_DIV0，CLK\_DIV1 和 CLK\_DIV2 进行控制。

寄存器	地址	读/写	描述	复位值
CLK_DIV0	0x7E00_F020	读/写	选择时钟分频器的比例。	0x0105_1000
CLK_DIV1	0x7E00_F024	读/写	选择时钟分频器的比例。	0x0000_0000
CLK_DIV2	0x7E00_F028	读/写	选择时钟分频器的比例。	0x0000_0000

CLK\_DIV0 主要控制系统时钟和多媒体 IP 的特殊时钟。APLL 和 MPLL 的输出频率是通过 ARM\_RATIO 和 MPLL\_RATIO 进行分频的。HCLKX2 时钟是其他操作系统时钟的基础时钟，通过 HCLKX2\_RATIO 进行分频。有操作频率的局限性。HCLKX2，HCLK 和 PCLK 的最大操作频率分别为 266MHz，133MHz 和 66MHz。NAND，SECUR 和 JPEG 的时钟操作不能超过 66MHz。MFC 和 CAM 时钟操作不能操过 133MHz。此时钟操作的条件必须满足 CLK\_DIV0 的配置。

用户软件必须通过 CLK\_DIV0 对时钟分频器的控制负责。

CLK_DIV0	位	描述	初始状态
MFC_RATIO	[31:28]	MFC时钟分频器的比例。 $CLKMFC = CLKMFCIN / (MFC\_RATIO + 1)$	0x0
JPEG_RATIO	[27:24]	JPEG时钟分频器的比例，必须是奇数值。换句话说，S3C6410仅支持偶数分频比例。 $CLKJPEG = HCLKX2 / (JPEG\_RATIO + 1)$	0x1
CAM_RATIO	[23:20]	CAM时钟分频器的比例。 $CLKCAM = HCLKX2 / (CAM\_RATIO + 1)$	0x0
SECUR_RATIO	[19:18]	安全时钟分频器的比例，必须是0x1或0x3。 $CLKSECUR = HCLKX2 / (SECUR\_RATIO + 1)$	0x1
ONENAND_RATIO	[17:16]	OneNAND时钟分频器的比例。 $CLKONENAND = HCLKX2 / (ONENAND\_RATIO + 1)$	0x1
PCLK_RATIO	[15:12]	PCLK 时钟分频器的比例，它必须是奇数值。换句话说，	0x1

		S3C6410 仅支持偶数分频比例。PCLK = HCLKX2 / (PCLK_RATIO + 1)	
HCLKX2_RATIO	[11:9]	HCLKX2时钟分频器的比例。 $HCLKX2 = HCLKX2IN / (HCLKX2\_RATIO + 1)$	0x0
HCLK_RATIO	[8]	HCLK时钟分频器的比例。 $HCLK = HCLKX2 / (HCLK\_RATIO + 1)$	0
RESERVED	[7:5]	保留。	0x0
MPLL_RATIO	[4]	DIVMPLL 时钟分频器的比例。 $DOUTMPLL = MOUTMPLL / (MPLL\_RATIO + 1)$	0
RESERVED	[3]	保留。	0
ARM_RATIO	[2:0]	DIVARM 时钟分频器的比例。 $ARMCLK = DOUTAPLL / (ARM\_RATIO + 1)$	0x0

CLK\_DIV1 控制MMC，LCD，TV 定标器和UHOST 时钟。

CLK_DIV1	位	描述	初始状态
RESERVED	[31:24]	保留。	0x0
UHOST_RATIO	[23:20]	USB 主机时钟分频器的比例。 $CLKUHOST = CLKUHOSTIN / (UHOST\_RATIO + 1)$	0x0
SCALER_RATIO	[19:16]	TV 定标器时钟分频器的比例。 $CLKSCALER = CLKSCALERIN / (SCALER\_RATIO + 1)$	0x0
LCD_RATIO	[15:12]	LCD时钟分频器的比例。 $CLKLCD = CLKLCDIN / (LCD\_RATIO + 1)$	0x0
MMC2_RATIO	[11:8]	MMC2 时钟分频器的比例。 $CLKMMC2 = CLKMMC2IN / (MMC2\_RATIO + 1)$	0x0
MMC1_RATIO	[7:4]	MMC1时钟分频器的比例。 $CLKMMC1 = CLKMMC1IN / (MMC1\_RATIO + 1)$	0x0
MMCO_RATIO	[3:0]	MMCO时钟分频器的比例。 $CLKMMCO = CLKMMCOIN / (MMCO\_RATIO + 1)$	0x0

CLK\_DIV2 控制SPI, AUDIO, UART和IrDA 时钟。

CLK_DIV2	位	描述	初始状态
RESERVED	[31:24]	保留。	0x0
IRDA_RATIO	[23:20]	IRDA 时钟分频器的比例。 $CLKIRDA = CLKIRDAIN / (IRDA\_RATIO + 1)$	0x0
UART_RATIO	[19:16]	UART时钟分频器的比例。 $CLKUART = CLKUARTIN / (UART\_RATIO + 1)$	0x0
AUDIO1_RATIO	[15:12]	AUDIO1时钟分频器的比例。 $CLKAUDIO1 = CLKAUDIO1IN / (AUDIO1\_RATIO + 1)$	0x0
AUDIO0_RATIO	[11:8]	AUDIO0时钟分频器的比例。 $CLKAUDIO0 = CLKAUDIO0IN / (AUDIO0\_RATIO + 1)$	0x0
SPI1_RATIO	[7:4]	SPI1时钟分频器的比例。 $CLKSPI1 = CLKSPI1IN / (SPI1\_RATIO + 1)$	0x0
SPI0_RATIO	[3:0]	SPI0时钟分频器的比例。。 $CLKSPI0 = CLKSPI0IN / (SPI0\_RATIO + 1)$	0x0

### 3.4.4. 时钟输出配置寄存器

内部时钟可以通过 GPIO 端口进行监控，其端口是 GPIO port-F。CLK\_OUT 寄存器在 PLL 输出，HCLK，48MHz，27MHz，RTC 和 TICK 间选择一个内部时钟。同时，它也分频选定的时钟。

寄存器	地址	读/写	描述	复位值
CLK_OUT	0x7E00_F02C	读/写	选择时钟输出。	0x0000_0000

CLK_OUT	位	描述	初始状态
RESERVED	[31:20]	保留。	0x000
DIVVAL	[19:16]	分频比例 (Divide ratio = DIVVAL + 1)。如果该区域不是0，则DCLKCMP没有意义。因此，当DIVVAL > 0	0x0

		时，DOUT总是有50%的占空比。	
RESERVED	[15]	保留。	0
CLKSEL	[14:12]	000 = FOUTAPLL/2 001 = FOUTEPLL 010 =HCLK 011 =CLK48M 100 =CLK27M 101 =RTC 110 =TICK 111 =DOUT	0x0
DCLKCMP	[11:8]	该区域DCLK时钟占空比的变化。因此，必须小于 DCLKDIV。只有当CLKSEL为 3' b111时，它是有效的。 如果 DCLKCMP 是 n，低水平的持续时间为 (n+1)。 高水平的持续时间为 ((DCLKDIV + 1) - (n+1))	0x0
DCLKDIV	[7:4]	DCLK分频值。 $DCLK \text{ 频率} = \text{时钟源} / (DCLKDIV + 1)$	0x0
RESERVED	[3:2]	保留。	0x0
DCLKSEL	[1]	选择 DCLK 时钟源(0: PCLK, 1: 48MHz)。	0
DCLKEN	[0]	使能 DCLK (0: 禁用, 1: 使能)。	0

### 3.4.5. 时钟选通控制寄存器

当没有要求运行时，S3C6410 可禁用每个 IP 的时钟操作。以下三个寄存器控制时钟禁用/使能操作。

寄存器	地址	读/写	描述	复位值
HCLK_GATE	0x7E00_F030	读/写	HCLK 时钟选通控制。	0xFFFF_FFFF
PCLK_GATE	0x7E00_F034	读/写	PCLK 时钟选通控制。	0xFFFF_FFFF
SCLK_GATE	0x7E00_F038	读/写	特殊时钟选通控制。	0xFFFF_FFFF

HCLK\_GATE控制所有Ips的HCLK，如果区域为‘1’，则HCLK被提供，否则，HCLK被屏蔽。当S3C6410转换成掉电模式时，系统控制器检查一些模块（IROM，MEM0，MEM1和MFC模块）的状态。因此，位25，22，21，0必须为‘1’，以符合掉电的要求。

HCLK_GATE	位	描述	初始状态
RESERVED	[31:30]	保留。	0x3
HCLK_UHOST	[29]	为 UHOST 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_SECUR	[28]	为安全子系统选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_SDMA1	[27]	为 SDMA1 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_SDMA0	[26]	为 SDMA0 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_IROM	[25]	为 IROM 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_DDR1	[24]	为 DDR1 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_DDR0	[23]	为 DDR0 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_MEM1	[22]	为 DMC1 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_MEM0	[21]	为 DMC0, SR0M, OneNAND, NFCON 和 CFCON 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_USB	[20]	为 USB OTG 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_HSMMC2	[19]	为 HSMMC2 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_HSMMC1	[18]	为 HSMMC1 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_HSMMC0	[17]	为 HSMMC0 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_MDP	[16]	为 MDP 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_DHOST	[15]	为直接 HOST 接口选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_IHOST	[14]	为间接 HOST 接口选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_DMA1	[13]	为 DMA1 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_DMA0	[12]	为 DMA0 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_JPEG	[11]	为 JPEG 选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_CAMIF	[10]	为相机接口选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_SCALER	[9]	为定标器选通 HCLK (0: 屏蔽, 1: 通过)。	1
HCLK_2D	[8]	为 2D 选通 HCLK (0: 屏蔽, 1: 通过)。	1

HCLK_TV	[7]	为 TV 译码器选通 HCLK(0: 屏蔽, 1: 通过)。	1
RESERVED	[6]	保留。	1
HCLK_POST0	[5]	为 POST0 选通 HCLK(0: 屏蔽, 1: 通过)。	1
HCLK_ROT	[4]	为旋转器选通 HCLK(0: 屏蔽, 1: 通过)。	1
HCLK_LCD	[3]	为 LCD 控制器选通 HCLK(0: 屏蔽, 1: 通过)。	1
HCLK_TZIC	[2]	为中断控制器选通 HCLK(0: 屏蔽, 1: 通过)。	1
HCLK_INTC	[1]	为向量中断控制器选通 HCLK(0: 屏蔽, 1: 通过)。	1
HCLK_MFC	[0]	为 MFC 选通 HCLK(0: 屏蔽, 1: 通过)。	1

PCLK\_GATE 控制所有Ips的PCLK。

PCLK_GATE	位	描述	初始状态
RESERVED	[31:25]	保留。	0x7F
PCLK_SKEY	[24]	为安全键选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_CHIPID	[23]	为片上 ID 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_SPI1	[22]	为 SPI1 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_SPI0	[21]	为 SPI0 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_HSIRX	[20]	为 HSI 接收器选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_HSITX	[19]	为 HIS 发送器选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_GPIO	[18]	为 GPIO 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_IIC	[17]	为 IIC 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_IIS1	[16]	为 IIS1 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_IIS0	[15]	为 IIS0 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_AC97	[14]	为 AC97 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_TZPC	[13]	为 TZPC 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_TSADC	[12]	为触摸屏 ADC 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_KEYPAD	[11]	为 Key PAD 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_IRDA	[10]	为 IRDA 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_PCM1	[9]	为 PCM1 选通 PCLK(0: 屏蔽, 1: 通过)。	1

PCLK_PCM0	[8]	为 PCM0 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_PWM	[7]	为 PWM 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_RTC	[6]	为 RTC 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_WDT	[5]	为看门狗定时器选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_UART3	[4]	为 UART3 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_UART2	[3]	为 UART2 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_UART1	[2]	为 UART1 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_UART0	[1]	为 UART0 选通 PCLK(0: 屏蔽, 1: 通过)。	1
PCLK_MFC	[0]	为 MFC 选通 PCLK(0: 屏蔽, 1: 通过)。	1

SCLK\_GATE控制IP的特殊时钟。

PCLK_GATE	位	描述	初始状态
RESERVED	[31]	保留。	1
SCLK_UHOST	[30]	为 USB-HOST 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_MMC2_48	[29]	为 MMC2 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_MMC1_48	[28]	为 MMC1 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_MMC0_48	[27]	为 MMC0 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_MMC2	[26]	为 MMC2 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_MMC1	[25]	为 MMC1 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_MMC0	[24]	为 MMC0 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_SPI1_48	[23]	为 SPI 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_SPI0_48	[22]	为 SPI 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_SPI1	[21]	为 SPI 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_SPI0	[20]	为 SPI 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_DAC27	[19]	为 DAC 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_TV27	[18]	为 TV 译码器选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_SCALER27	[17]	为 scaler27 选通特殊时钟 (0: 屏蔽, 1: 通过)。	1
SCLK_SCALER	[16]	为定标器选通特殊时钟 (0: 屏蔽, 1: 通过)。	1

SCLK_LCD27	[15]	为 LCD 控制器选通特殊时钟（0：屏蔽，1：通过）。	1
SCLK_LCD	[14]	为 LCD 控制器选通特殊时钟（0：屏蔽，1：通过）。	1
RESERVED	[13]	保留。	1
SCLK_POST0_27	[12]	为 POST0 选通特殊时钟（0：屏蔽，1：通过）。	1
RESERVED	[11]	保留。	1
SCLK_POST0	[10]	为 POST0 选通特殊时钟（0：屏蔽，1：通过）。	1
SCLK_AUDI01	[9]	为 PCM1, IIS1 和 AC97 1 选通特殊时钟（0：屏蔽，1：通过）。	1
SCLK_AUDI00	[8]	为 PCM0, IIS0 和 AC97 0 选通特殊时钟（0：屏蔽，1：通过）。	1
SCLK_SECUR	[7]	为安全模块选通特殊时钟（0：屏蔽，1：通过）。	1
SCLK_IRDA	[6]	为 IRDA 选通特殊时钟（0：屏蔽，1：通过）。	1
SCLK_UART	[5]	为 UART0~3 选通特殊时钟（0：屏蔽，1：通过）。	1
SCLK_OneNAND	[4]	为 OneNAND 选通特殊时钟（0：屏蔽，1：通过）。	1
SCLK_MFC	[3]	为 MFC 选通特殊时钟（0：屏蔽，1：通过）。	1
SCLK_CAM	[2]	为相机接口选通特殊时钟（0：屏蔽，1：通过）。	1
SCLK_JPEG	[1]	为 JPEG 选通特殊时钟（0：屏蔽，1：通过）。	1
RESERVED	[0]	保留。	1

### 3.4.6. AHB 总线控制寄存器

大多数多媒体 Ips 都是通过 AHB2.0 总线系统来连接的。总线优先级可以使用以下三个寄存器来控制。

寄存器	地址	读/写	描述	复位值
AHB_CON0	0x7E00_F100	读/写	配置 AHB I/P/X/F 总线。	0x0400_0000
AHB_CON1	0x7E00_F104	读/写	配置 AHB M1/M0/T1/T0 总线。	0x0000_0000
AHB_CON2	0x7E00_F108	读/写	配置 AHB R/S1/S0 总线。	0x0000_0000

可为每个 AHB 总线配置仲裁方法，利用 “PRIOR\_TYPE\_name” 域如下：



- 00：固定优先类型。其首要任务是定义使用“FIX\_PRIOR\_name”域。
- 01：最后允许的最低信号。最后假定主机有最低的仲裁值。
- 10：校准。

当优先权类型域选择固定优先级（00）时，优先顺序的定义如表 3-5 所示，显示如下：

表 3-5 优先顺序的定义

FIX_PRIOR_name	Highest
000	0 - 1 - 2 - 3 - 4 - 5 - 6 - 7
001	1 - 2 - 3 - 4 - 5 - 6 - 0 - 7
010	2 - 3 - 4 - 5 - 6 - 0 - 1 - 7
011	3 - 4 - 5 - 6 - 0 - 1 - 2 - 7
100	4 - 5 - 6 - 0 - 1 - 2 - 3 - 7
101	5 - 6 - 0 - 1 - 2 - 3 - 4 - 7
110	6 - 0 - 1 - 2 - 3 - 4 - 5 - 7

DISABLE\_HLOCK 域控制 AHB 总线系统的锁操作。如果主机锁操作请求时，域为‘0’总线仲裁者获得允许信号。如果域为‘1’，主机不能获得锁存允许。

AHB\_CON0 控制 AHB-F，AHB-X，AHB-P 和 AHB-I 总线系统。

AHB_CON0	位	描述	初始状态
DISABLE_HLOCK_I	[31]	为 I-block 控制 HLOCK（0：禁用。1：使能）。	0
RESERVED	[30]	保留。	0
PRIOR_TYPE_I	[29:28]	为 AHB-I 仲裁类型。	0
RESERVED	[27]	保留。	0
FIX_PRIOR_I	[26:24]	为 AHB-I 固定优先顺序。	0x4
DISABLE_HLOCK_P	[23]	为 P-block 控制 HLOCK（0：禁用。1：使能）。	0
RESERVED	[22]	保留。	0
PRIOR_TYPE_P	[21:20]	为 AHB-P 仲裁类型。	0
RESERVED	[19]	保留。	0
FIX_PRIOR_P	[18:16]	为 AHB-P 固定优先顺序。	0x0
DISABLE_HLOCK_X	[15]	为 X-block 控制 HLOCK（0：禁用。1：使能）。	0

RESERVED	[14]	保留。	0
PRIOR_TYPE_X	[13:12]	为 AHB-X 仲裁类型。	0
RESERVED	[11]	保留。	0
FIX_PRIOR_X	[10:8]	为 AHB-X 固定优先顺序。	0x0
DISABLE_HLOCK_F	[7]	为 F-block 控制 HLOCK（0：禁用。1：使能）。	0
RESERVED	[6]	保留。	0
PRIOR_TYPE_F	[5:4]	为 AHB-F 仲裁类型。	0
RESERVED	[3]	保留。	0
FIX_PRIOR_F	[2:0]	为 AHB-F 固定优先顺序。	0x0

以上总线系统的仲裁数量显示，如表 3-6 所示：

表 3-6 总线系统的仲裁数量

仲裁数量	AHB-F	AHB-X	AHB-P	AHB-I
0	显示控制器 0	USB OTG	2D 图表	Camera I/F
1	显示控制器 1	HS-MMC0	TV 定标器	Camera I/F
2	显示控制器 2	HS-MMC1	–	Camera I/F
3	显示控制器 3	HS-MMC2	–	Camera I/F
4	显示控制器 4	–	–	JPEG
5	POST	–	–	–
6	旋转器	–	–	–
7	默认主机总线			

AHB\_CON1控制AHB-T0，AHB-T1，AHB-M0和AHB-M1总线系统。

AHB_CON1	位	描述	初始状态
DISABLE_HLOCK_M	[31]	为 M-block 控制 HLOCK（0：禁用。1：使能）。	0
RESERVED	[30]	保留。	0
PRIOR_TYPE_M1	[29:28]	为 AHB-M1 仲裁类型。	0
RESERVED	[27]	保留。	0

FIX_PRIOR_M1	[26:24]	为 AHB-M1 固定优先顺序。	0x0
RESERVED	[23:22]	保留。	0x0
PRIOR_TYPE_M0	[21:20]	为 AHB-M0 仲裁类型。	0
RESERVED	[19]	保留。	0
FIX_PRIOR_M0	[18:16]	为 AHB-M0 固定优先顺序。	0x0
DISABLE_HLOCK_T	[15]	为 T-block 控制 HLOCK（0：禁用。1：使能）。	0
RESERVED	[14]	保留。	0
PRIOR_TYPE_T1	[13:12]	为 AHB-T1 仲裁类型。	0
RESERVED	[11]	保留。	0
FIX_PRIOR_T1	[10:8]	为 AHB-T1 固定优先顺序。	0x0
RESERVED	[7:6]	保留。	0x0
PRIOR_TYPE_T0	[5:4]	为 AHB-T0 仲裁类型。	0
RESERVED	[3]	保留。	0
FIX_PRIOR_T0	[2:0]	为 AHB-T0 固定优先顺序。	0x0

以上总线系统的仲裁数量显示，如表 3-7 所示：

表 3-7 总线系统的仲裁数量

仲裁数量	AHB-T0	AHB-T1	AHB-M0	AHB-M1
0	HOST I/F	HOST I/F	DMA0	DMA0
1	USB Host	—	DMA1	DMA1
2	MDP I/F	—	—	—
3	—	—	—	—
4	—	—	—	—
5	—	—	—	—
6	—	—	—	—
7	默认主机总线			

AHB\_CON2控制AHB-S0，AHB-S1和AHB-R总线系统。

AHB_CON1	位	描述	初始状态
RESERVED	[31:24]	保留。	0x00
DISABLE_HLOCK_R	[23]	为 R-block 控制 HLOCK（0：禁用。1：使能）。	0
RESERVED	[22:16]	保留。	0
DISABLE_HLOCK_S	[15]	为 S-block 控制 HLOCK（0：禁用。1：使能）。	0
RESERVED	[14]	保留。	0
PRIOR_TYPE_S1	[13:12]	为 AHB-S1 仲裁类型。	0
RESERVED	[11]	保留。	0
FIX_PRIOR_S1	[10:8]	为 AHB-S1 固定优先顺序。	0x0
RESERVED	[7:6]	保留。	0x0
PRIOR_TYPE_S0	[5:4]	为 AHB-S0 仲裁类型。	0
RESERVED	[3]	保留。	0
FIX_PRIOR_S0	[2:0]	AHB-S0 固定优先顺序。	0x0

以上总线系统的仲裁数量显示，如表 3-8 所示：

表 3-8 总线系统的仲裁数量

仲裁数量	AHB-S0	AHB-S1	AHB-R
0	安全 DMA0	安全 DMA0	CFCON
1	安全 DMA1	安全 DMA1	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	默认主机总线		

3.4.7. 安全 DMA 控制寄存器

寄存器	地址	读/写	描述	复位值
SDMA_SEL	0x7E00_F110	读/写	安全 DMA 输入选择。	0x0000_0000

SDMA_SEL	位	描述	初始状态
SECURITY_TX	[31]	安全 Tx 的 DMA 选择（总是选择 SDMA1，不管 SECURITY_TX 域）。	0
SECURITY_RX	[30]	安全 Rx 的 DMA 选择（总是选择 SDMA1，不管 SECURITY_TX 域）。	0
RESERVED	[29:28]	保留。	0x0
EXTERNAL	[27]	外部的 DMA 选择(0: SDMA1, 1: DMA1)。	0
IRDA	[26]	IrDA 的 DMA 选择(0: SDMA1, 1: DMA1)。	0
PWM	[25]	IrDA 的 DMA 选择(0: SDMA1, 1: DMA1)。	0
AC_MICIN	[24]	IrDA 的 DMA 选择(0: SDMA1, 1: DMA1)。	0
AC_PCMIN	[23]	PCM 输入的 DMA 选择(0: SDMA1, 1: DMA1)。	0
AC_PCMOUT	[22]	PCM 输出的 DMA 选择(0: SDMA1, 1: DMA1)。	0
SPI1_RX	[21]	SPI1 Rx 的 DMA 选择(0: SDMA1, 1: DMA1)。	0
SPI1_TX	[20]	SPI1 Tx 的 DMA 选择(0: SDMA1, 1: DMA1)。	0
I2S1_RX	[19]	I2S1 Rx 的 DMA 选择(0: SDMA1, 1: DMA1)。	0
I2S1_TX	[18]	I2S1 Tx 的 DMA 选择(0: SDMA1, 1: DMA1)。	0
PCM1_RX	[17]	PCM1 Rx 的 DMA 选择(0: SDMA1, 1: DMA1)。	0
PCM1_TX	[16]	PCM1 Tx 的 DMA 选择(0: SDMA1, 1: DMA1)。	0
HSI_RX	[15]	HSI Rx 的 DMA 选择(0: SDMA0, 1: DMA0)。	0
HSI_TX	[14]	HSI Tx 的 DMA 选择(0: SDMA0, 1: DMA0)。	0
SPI0_RX	[13]	SPI0 Rx 的 DMA 选择(0: SDMA0, 1: DMA0)。	0
SPI0_TX	[12]	SPI0 Tx 的 DMA 选择(0: SDMA0, 1: DMA0)。	0
I2S0_RX	[11]	I2S0 Rx 的 DMA 选择(0: SDMA0, 1: DMA0)。	0

I2S0_TX	[10]	I2S0 Tx 的 DMA 选择 (0: SDMA0, 1: DMA0)。	0
PCM0_RX	[9]	PCM0 Rx 的 DMA 选择 (0: SDMA0, 1: DMA0)。	0
PCM0_TX	[8]	PCM0 Tx 的 DMA 选择 (0: SDMA0, 1: DMA0)。	0
UART3[1]	[7]	UART3 的 DMA 选择 (0: SDMA0, 1: DMA0)。	0
UART3[0]	[6]	UART3 的 DMA 选择 (0: SDMA0, 1: DMA0)。	0
UART2[1]	[5]	UART2 的 DMA 选择 (0: SDMA0, 1: DMA0)。	0
UART2[0]	[4]	UART2 的 DMA 选择 (0: SDMA0, 1: DMA0)。	0
UART1[1]	[3]	UART1 的 DMA 选择 (0: SDMA0, 1: DMA0)。	0
UART1[0]	[2]	UART1 的 DMA 选择 (0: SDMA0, 1: DMA0)。	0
UART0[1]	[1]	UART0 的 DMA 选择 (0: SDMA0, 1: DMA0)。	0
UART0[0]	[0]	UART0 的 DMA 选择 (0: SDMA0, 1: DMA0)。	0

### 3.4.8. 软件复位控制寄存器

寄存器	地址	读/写	描述	复位值
SW_RST	0x7E00_F114	读/写	产生软件复位。	0x0000_0000

SW_RST	位	描述	初始状态
RESERVED	[31:16]	保留。	0x0000
SWRESET	[15:0]	当值为 0x6410 时，产生软件复位。	0x0000

### 3.4.9. 系统 ID 寄存器

寄存器	地址	读/写	描述	复位值
SYS_ID	0x7E00_F118	读	系统 ID，用于修改并通过。	0x0000_0000

SYS_ID	位	描述	初始状态
RESERVED	[31:8]	保留。	0x0000_00

Revision	[7:4]	规格修改。	0x0
Pass	[3:0]	布局修改。	0x0

### 3.4.10. 内存控制状态寄存器

内存控制状态寄存器必须通过软件进行初始化，除了MEM\_CFG\_STAT之外。

寄存器	地址	读/写	描述	复位值
MEM_SYS_CFG	0x7E00_F120	读/写	内存子系统配置寄存器。	0x0000_0080
QOS_OVERRIDE0	0x7E00_F124	读/写	DMC0 QOS 覆写寄存器。	0x0000_0000
QOS_OVERRIDE1	0x7E00_F128	读/写	DMC1 QOS 覆写寄存器。	0x0000_0000
MEM_CFG_STAT	0x7E00_F12C	读	内存子系统建立状态寄存器。	0x0000_0000

MEM_SYS_CFG	位	描述	初始状态
RESERVED	[31:15]	保留。	0x0000_0
INDEP_CF	[14]	使用独立的 CF 接口。  0=使用内存端口 0 共享 EBI  1=使用独立的 CF 接口	0
RESERVED	[13]	保留。	0
BUS_WIDTH	[12]	SR0MC CS0 内存总线宽度的初始状态选择。  0=8 位数据宽度  1=16 位数据宽度  如果选择 NOR 引导（OM[4:1] = 0101），则这个设置被忽略，选择 16 位数据宽度。  即使该位被设置为 0 或 1，在 SR0MC 中，该选择只有 SR0M_BW SFR 的 DataWidth0 复位值。SR0MC 的 CS0 总线宽度遵循 DataWidth0 的设置。	0
EBI_PRI	[11]	设置 EBI 优先权设计。	0

		0=固定优先权设计 1=循环优先权设计																						
EBI_FIX_PRIORITY	[10:8]	EBI 固定优先权设置。 <table><tr><td></td><td>高 &lt;-&gt; 低</td><td></td></tr><tr><td>0, 6, 7</td><td>DMC0 - SROMC - OneNANDC CS0 - OneNANDC CS1 - NFCON - CFCON</td><td></td></tr><tr><td>1</td><td>DMC0 - OneNANDC CS0 - OneNANDC CS1 - SROMC - NFCON - CFCON</td><td></td></tr><tr><td>2</td><td>DMC0 - OneNANDC CS1 - NFCON - SROMC - OneNANDC CS0 - CFCON</td><td></td></tr><tr><td>3</td><td>DMC0 - NFCON - SROMC - OneNANDC CS0 - OneNANDC CS1 - CFCON</td><td></td></tr><tr><td>4</td><td>DMC0 - CFCON - SROMC - OneNANDC CS0 - OneNANDC CS1 - NFCON</td><td></td></tr><tr><td>5</td><td>SROMC - DMC0 - OneNANDC CS0 - OneNANDC CS1 - NFCON - CFCON</td><td></td></tr></table>		高 <-> 低		0, 6, 7	DMC0 - SROMC - OneNANDC CS0 - OneNANDC CS1 - NFCON - CFCON		1	DMC0 - OneNANDC CS0 - OneNANDC CS1 - SROMC - NFCON - CFCON		2	DMC0 - OneNANDC CS1 - NFCON - SROMC - OneNANDC CS0 - CFCON		3	DMC0 - NFCON - SROMC - OneNANDC CS0 - OneNANDC CS1 - CFCON		4	DMC0 - CFCON - SROMC - OneNANDC CS0 - OneNANDC CS1 - NFCON		5	SROMC - DMC0 - OneNANDC CS0 - OneNANDC CS1 - NFCON - CFCON		000
	高 <-> 低																							
0, 6, 7	DMC0 - SROMC - OneNANDC CS0 - OneNANDC CS1 - NFCON - CFCON																							
1	DMC0 - OneNANDC CS0 - OneNANDC CS1 - SROMC - NFCON - CFCON																							
2	DMC0 - OneNANDC CS1 - NFCON - SROMC - OneNANDC CS0 - CFCON																							
3	DMC0 - NFCON - SROMC - OneNANDC CS0 - OneNANDC CS1 - CFCON																							
4	DMC0 - CFCON - SROMC - OneNANDC CS0 - OneNANDC CS1 - NFCON																							
5	SROMC - DMC0 - OneNANDC CS0 - OneNANDC CS1 - NFCON - CFCON																							
ADDR_EXPAND	[7]	Xm1DATA[31:16]引脚使用的设置。 0= Xm1DATA[31:16]引脚用于 DMC1 上的半字数据域，data[31:16]的使用。 1= Xm1DATA[31:16]引脚用于 SROMC 上的 10 位地址域，address[25:16]的使用。	1																					
ENDIAN	[6]	用于 SROMC，NFCON 和内部 ROM 的端控制设置。 0=小端内存系统 1=大端字节不变内存系统	0																					
MP0_CS_SELECT	[5:0]	设置静态存储器片选，内存端口 0 的多路复用。 MP0_CS_SEL[0]和 MP0_CS_SEL[2]的设置被忽略。区分 OneNANDC 和 NFCON 是由 XSELNAND 引脚的值决定的，而不是 MP0_CS_SEL[0]和 MP0_CS_SEL[2]。当 XSELNAND 为 0 时，OneNANDC 被选择。当 XSELNAND 为 1 时，NFCON 被选择。	0x00																					



		<p>当选择NAND引导（OM[4:3] = 00）时，则MP0_CS_SEL[1]和MP0_CS_SEL[3]值的设置以及XSELNAND设置被忽略。</p> <p>Xm0CSn[2]和Xm0CSn[3]作为NFC CON CS0和NFC CON CS1被使用。</p> <p>在这种情况下，XSELNAND应被设置为1。</p> <p>当选择OneNAND引导（OM[4:1] = 0110）时，MP0_CS_SEL[1]和MP0_CS_SEL[3]值的设置被忽略。Xm0CSn[2]和Xm0CSn[3]作为OneNAND CS0和OneNAND CS1被使用。在这种情况下，XSELNAND应被设置为0。</p>																																																																																																																
		<table><tr><th rowspan="2"></th><th colspan="6">MP0_CS_SEL</th><th rowspan="2"></th></tr><tr><th>[5]</th><th>[4]</th><th>[3]</th><th>[2]</th><th>[1]</th><th>[0]</th></tr><tr><td></td><td>]</td><td>]</td><td>]</td><td>]</td><td>]</td><td>]</td><td></td></tr><tr><td>Xm0CSn[0]</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>SROMC CS0</td></tr><tr><td>Xm0CSn[0]</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>SROMC CS1</td></tr><tr><td rowspan="3">Xm0CSn[2]</td><td>-</td><td>-</td><td>-</td><td>-</td><td>1</td><td>-</td><td>SROMC CS2</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>0</td><td>-</td><td>OneNAND CS0</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>0</td><td>-</td><td>NFC CON CS0</td></tr><tr><td rowspan="3">Xm0CSn[3]</td><td>-</td><td>-</td><td>1</td><td>-</td><td>-</td><td>-</td><td>SROMC CS3</td></tr><tr><td>-</td><td>-</td><td>0</td><td>-</td><td>-</td><td>-</td><td>OneNAND CS1</td></tr><tr><td>-</td><td>-</td><td>0</td><td>-</td><td>-</td><td>-</td><td>NFC CON CS1</td></tr><tr><td rowspan="2">Xm0CSn[4]</td><td>-</td><td>0</td><td>-</td><td>-</td><td>-</td><td>-</td><td>SROMC CS4</td></tr><tr><td>-</td><td>1</td><td>-</td><td>-</td><td>-</td><td>-</td><td>CFC CON CS0</td></tr><tr><td rowspan="2">Xm0CSn[5]</td><td>0</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>SROMC CS5</td></tr><tr><td>1</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>CFC CON CS1</td></tr></table>		MP0_CS_SEL							[5]	[4]	[3]	[2]	[1]	[0]		]	]	]	]	]	]		Xm0CSn[0]	-	-	-	-	-	-	SROMC CS0	Xm0CSn[0]	-	-	-	-	-	-	SROMC CS1	Xm0CSn[2]	-	-	-	-	1	-	SROMC CS2	-	-	-	-	0	-	OneNAND CS0	-	-	-	-	0	-	NFC CON CS0	Xm0CSn[3]	-	-	1	-	-	-	SROMC CS3	-	-	0	-	-	-	OneNAND CS1	-	-	0	-	-	-	NFC CON CS1	Xm0CSn[4]	-	0	-	-	-	-	SROMC CS4	-	1	-	-	-	-	CFC CON CS0	Xm0CSn[5]	0	-	-	-	-	-	SROMC CS5	1	-	-	-	-	-	CFC CON CS1
	MP0_CS_SEL																																																																																																																	
	[5]	[4]	[3]	[2]	[1]	[0]																																																																																																												
	]	]	]	]	]	]																																																																																																												
Xm0CSn[0]	-	-	-	-	-	-	SROMC CS0																																																																																																											
Xm0CSn[0]	-	-	-	-	-	-	SROMC CS1																																																																																																											
Xm0CSn[2]	-	-	-	-	1	-	SROMC CS2																																																																																																											
	-	-	-	-	0	-	OneNAND CS0																																																																																																											
	-	-	-	-	0	-	NFC CON CS0																																																																																																											
Xm0CSn[3]	-	-	1	-	-	-	SROMC CS3																																																																																																											
	-	-	0	-	-	-	OneNAND CS1																																																																																																											
	-	-	0	-	-	-	NFC CON CS1																																																																																																											
Xm0CSn[4]	-	0	-	-	-	-	SROMC CS4																																																																																																											
	-	1	-	-	-	-	CFC CON CS0																																																																																																											
Xm0CSn[5]	0	-	-	-	-	-	SROMC CS5																																																																																																											
	1	-	-	-	-	-	CFC CON CS1																																																																																																											

QOS_OVERRIDE0	位	描述	初始状态
---------------	---	----	------

/QOS_OVERRID1																																										
RESERVED	[31:16]	保留。	0x0000_0000																																							
QOS_OV_ID	[15:0]	<p>当有一位或多位为高电平，DMC0/DMC1 中的匹配位相当于 QOS_OV_ID 位时，那么读地址的服务质量被迫到最低延迟。</p> <p>AXI 主机 ID 如下：</p> <table><tr><td>AXI ID 分配</td><td>主机名</td><td>相关的 IP</td></tr><tr><td>0x00</td><td>I_BLOCK</td><td>Camera 和 JPEG</td></tr><tr><td>0x01</td><td>F_BLOCK</td><td>显示</td></tr><tr><td>0x02</td><td>P_BLOCK</td><td>POST</td></tr><tr><td>0x03</td><td>V_BLOCK</td><td>MFC</td></tr><tr><td>0x04</td><td>X_BLOCK</td><td>HSMMC 和 OTG</td></tr><tr><td>0x05</td><td>T_BLOCK</td><td>Host I/F</td></tr><tr><td>0x06</td><td>M_BLOCK</td><td>DMA</td></tr><tr><td>0x07</td><td>S_BLOCK</td><td>Security</td></tr><tr><td>0x08</td><td>ARMI</td><td>ARM 内核指令</td></tr><tr><td>0x09</td><td>ARMRW</td><td>ARM 内核数据</td></tr><tr><td>0x0A</td><td>ARMD</td><td>ARM 内核 DMA</td></tr><tr><td>0x0B</td><td>CF</td><td>CFCON</td></tr></table>	AXI ID 分配	主机名	相关的 IP	0x00	I_BLOCK	Camera 和 JPEG	0x01	F_BLOCK	显示	0x02	P_BLOCK	POST	0x03	V_BLOCK	MFC	0x04	X_BLOCK	HSMMC 和 OTG	0x05	T_BLOCK	Host I/F	0x06	M_BLOCK	DMA	0x07	S_BLOCK	Security	0x08	ARMI	ARM 内核指令	0x09	ARMRW	ARM 内核数据	0x0A	ARMD	ARM 内核 DMA	0x0B	CF	CFCON	0000
AXI ID 分配	主机名	相关的 IP																																								
0x00	I_BLOCK	Camera 和 JPEG																																								
0x01	F_BLOCK	显示																																								
0x02	P_BLOCK	POST																																								
0x03	V_BLOCK	MFC																																								
0x04	X_BLOCK	HSMMC 和 OTG																																								
0x05	T_BLOCK	Host I/F																																								
0x06	M_BLOCK	DMA																																								
0x07	S_BLOCK	Security																																								
0x08	ARMI	ARM 内核指令																																								
0x09	ARMRW	ARM 内核数据																																								
0x0A	ARMD	ARM 内核 DMA																																								
0x0B	CF	CFCON																																								

MEM_CFG_STAT	位	描述	初始状态
RESERVED	[31:16]	保留。	0x0000_00
CFG_PRI_TYPE	[15]	当前 EBI 优先权设计。注意 MEM_SYS_CFG 寄存器的 EBI_PRI 域。	0
CFG_FIX_PRI_TYPE	[14:12]	当前 EBI 固定优先权的设置。注意 MEM_SYS_CFG 寄存器的 EBI_FIX_PRI 域。	0
RESERVED	[11]	保留。	0
CFG_INDEP_CF	[10]	显示独立的 CF 接口。	0

CFG_MUX_FLASH	[9]	显示 NAND Flash 类型设置。 0=使用 OneNAND 控制器 1=使用 NAND Flash 控制器	0
CFG_BUS_WIDTH	[8]	显示 SROMC CS0/NAND Flash 存储器总线宽度。	0
CFG_NOR_BOOT	[8]	0=16 位宽度, NOR 引导没被选择。 1=16 位宽度, NOR 引导被选择。	XOM dependent
CFG_NFCON_BOOT	[7]	0=NAND Flash 没被用于引导使用。 1=NAND Flash 引导被设置。	XOM dependent
CFG_ADDR_EXPAND	[4]	显示是否有 Xm1DATA[31:16]引脚被 SROMC 地址域使用。 0=Xm1DATA[31:16]引脚被用于 DMC1 的高半字数 据域, data[31:16]。 1=Xm1DATA[31:16]引脚被用于 SROMC 的高 10 位地 址域, address[25:16]。	0
CFG_ADV_FLASH	[3]	显示是否有 NAND Flash 的初始设置为高级 Flash。 0=正常 NAND Flash 1=高级 NAND Flash	0
CFG_ADDR_CYCLE	[2]	显示 NAND flash 的地址周期初始设置。	0
CFG_BUS_WIDTH	[1]	显示 SROMC CS0 内存总线宽度初始设置。 0=8 位 1=16 位	0
CFG_PAGE_SIZE	[0]	显示 NAND flash 的页大小初始设置。	0

### 3.4.11. 电源模式控制寄存器

寄存器	地址	读/写	描述	复位值
PWR_CFG	0x7E00_F804	读/写	电源管理配置。	0x0000_0001
EINT_MASK	0x7E00_F808	读/写	EINT 屏蔽设置。	0x0000_0000

NORMAL_CFG	0x7E00_F810	读/写	正常模式下的电源管理配置。	0xFFFF_FF00
STOP_CFG	0x7E00_F814	读/写	停止模式下的电源管理配置。	0x2012_0100
SLEEP_CFG	0x7E00_F818	读/写	睡眠模式下的电源管理配置。	0x0000_0000

PWR_CFG	位	描述	初始状态
RESERVED	[31:17]	保留。	0x0000
MMC2_WAKEUP_MASK	[16]	MMC2唤醒源(0: 作为一个唤醒源使用, 1: 禁用)。	0
MMC1_WAKEUP_MASK	[15]	MMC1唤醒源(0: 作为一个唤醒源使用, 1: 禁用)。	0
MMCO_WAKEUP_MASK	[14]	MMCO唤醒源(0: 作为一个唤醒源使用, 1: 禁用)。	0
HSI_WAKEUP_MASK	[13]	HSI唤醒源(0: 作为一个唤醒源使用, 1: 禁用)。	0
TS_WAKEUP_MASK	[12]	触摸屏唤醒源(0: 作为一个唤醒源使用, 1: 禁用)。	0
TICK_WAKEUP_MASK	[11]	RTC TICK唤醒源(0: 作为一个唤醒源使用, 1: 禁用)。	0
ALARM_WAKEUP_MASK	[10]	RTC警告唤醒源(0: 作为一个唤醒源使用, 1: 禁用)。	0
MSM_WAKEUP_MASK	[9]	MSM modem唤醒源(0: 作为一个唤醒源使用, 1: 禁用)。	0
KEY_WAKEUP_MASK	[8]	键盘唤醒源(0: 作为一个唤醒源使用, 1: 禁用)。	0
BATF_WAKEUP_MASK	[7]	BATF唤醒源(0: 禁用, 1: 使用CFG_BATFLT域为01时, 作为一个唤醒源使用)。	0
CFG_STANDBYWFI	[6:5]	ARM1176 STADNBYWFI 的配置。 00: 忽略 01: 进入闲置模式 10: 进入停止模式 11: 进入睡眠模式	0x0
CFG_BATFLT	[4:3]	nBAT_FLT 的配置。 00: 忽略 01: 产生中断(仅当 BATF_WAKEUP_MASK 域为 1 时被使用) 11: 进入 SLEEP 模式或 E-SLEEP 模式	0x0
CFG_BATF_WKUP	[2]	XnBATF被清除后, 配置唤醒源。 0: 仅使用nWRESET 1: 使用所有睡眠模式的唤醒源	0

RESERVED	[1]	保留。	0
OSC27_EN	[0]	控制 27MHz X-tal 振荡器 pad。 (0: 禁用, 1: 使能)	1

EINT_MASK	位	描述	初始状态
RESERVED	[31:28]	保留。	0x0
EINT_WAKEUP_MASK	[27:0]	外部中断唤醒, 屏蔽EINT[27:0] (0: 作为一个唤醒源使用, 1: 禁用), 该区域反应在正常模式上。因此, 当EINT被作为一个正常的外部中断源使用时, 该区域必须清除。	0x00_0000

NORMAL_CFG	位	描述	初始状态
RESERVED	[31]	不改变。	1
IROM	[30]	0: LP模式(OFF), 1: 激活模式(ON)。	1
RESERVED	[29:17]	不改变。	0x1FFF
DOMAIN_ETM	[16]	0: LP模式(OFF), 1: 激活模式(ON)。	1
DOMAIN_S	[15]	0: LP模式(OFF), 1: 激活模式(ON)。	1
DOMAIN_F	[14]	0: LP模式(OFF), 1: 激活模式(ON)。	1
DOMAIN_P	[13]	0: LP模式(OFF), 1: 激活模式(ON)。	1
DOMAIN_I	[12]	0: LP模式(OFF), 1: 激活模式(ON)。	1
RESERVED	[11:10]	不改变。	0x3
DOMAIN_V	[9]	0: LP模式(OFF), 1: 激活模式(ON)。	1
RESERVED	[8:0]	不改变。	0x100

STOP_CFG	位	描述	初始状态
RESERVED	[31:30]	不改变。	0
MEMORY_ARM	[29]	0: LP模式(OFF), 1: 激活模式(ON)。	1
RESERVED	[28:21]	不改变。	0x00
TOP_MEMORY	[20]	0: LP模式(OFF), 1: 激活模式(Retention)。	1

RESERVED	[19:18]	不改变。	0x0
ARM_LOGIC	[17]	0: LP模式 (OFF), 1: 激活模式 (ON)。	1
RESERVED	[16:9]	不改变。	0x00
TOP_LOGIC	[8]	0: LP模式 (Retention), 1: 激活模式 (ON), 在进入停止模式之前, 该区域必须为 ‘0’ 。	1
RESERVED	[7:2]	不改变。	0x00
PLL_EN	[1]	在停止模式下, 控制PLL操作。 (0: 禁用, 1: 使能)	0
OSC_EN	[0]	在停止模式下, 控制X-tal振荡器pad。 (0: 禁用, 1: 使能)	0

SLEEP_CFG	位	描述	初始状态
RESERVED	[31:1]	保留。	0x0000_0000
OSC_EN	[0]	在睡眠模式下, 控制X-tal振荡器pad。 (0: 禁用, 1: 使能)	0

### 3.4.12. 系统稳定计数器

寄存器	地址	读/写	描述	复位值
OSC_FREQ	0x7E00_F820	读/写	振荡器频率刻度计数器。	0x0000_000F
OSC_STABLE	0x7E00_F824	读/写	振荡器 pad 稳定计数器。	0x0000_0001
PWR_STABLE	0x7E00_F828	读/写	电源稳定计数器。	0x0000_0001
MTC_STABLE	0x7E00_F830	读/写	MTC 稳定计数器。	0xFFFF_FFFF

OSC_FREQ	位	描述	初始状态
RESERVED	[31:4]	保留。	0x0000
OSC_FREQ_VALUE	[3:0]	振荡器频率刻度计数器。 (OSC_FREQ_VALUE / <i>oscillator_frequency</i>	0xF

		> 200ns)	
--	--	----------	--

OSC_STABLE	位	描述	初始状态																								
RESERVED	[31:4]	保留。	0x0000_000																								
OSC_CNT_VALUE	[3:0]	振荡器pad稳定计数器的值。 <table> <tr> <td>值</td><td>周期</td><td>值</td><td>周期</td></tr> <tr> <td>0x0</td><td>2<sup>4</sup></td><td>0x1</td><td>2<sup>9</sup></td></tr> <tr> <td>0x2</td><td>2<sup>10</sup></td><td>0x3</td><td>2<sup>11</sup></td></tr> <tr> <td>0x4</td><td>2<sup>12</sup></td><td>0x5</td><td>2<sup>13</sup></td></tr> <tr> <td>0x6</td><td>2<sup>14</sup></td><td>0x7</td><td>2<sup>15</sup></td></tr> <tr> <td>0x8</td><td>2<sup>16</sup></td><td>其它</td><td>2<sup>4</sup></td></tr> </table>	值	周期	值	周期	0x0	2 <sup>4</sup>	0x1	2 <sup>9</sup>	0x2	2 <sup>10</sup>	0x3	2 <sup>11</sup>	0x4	2 <sup>12</sup>	0x5	2 <sup>13</sup>	0x6	2 <sup>14</sup>	0x7	2 <sup>15</sup>	0x8	2 <sup>16</sup>	其它	2 <sup>4</sup>	0x1
值	周期	值	周期																								
0x0	2 <sup>4</sup>	0x1	2 <sup>9</sup>																								
0x2	2 <sup>10</sup>	0x3	2 <sup>11</sup>																								
0x4	2 <sup>12</sup>	0x5	2 <sup>13</sup>																								
0x6	2 <sup>14</sup>	0x7	2 <sup>15</sup>																								
0x8	2 <sup>16</sup>	其它	2 <sup>4</sup>																								

PWM_STABLE	位	描述	初始状态																								
RESERVED	[31:4]	保留。	0x0000_000																								
PWR_CNT_VALUE	[3:0]	电源稳定计数器的值。 <table> <tr> <td>值</td><td>周期</td><td>值</td><td>周期</td></tr> <tr> <td>0x0</td><td>2<sup>4</sup></td><td>0x1</td><td>2<sup>12</sup></td></tr> <tr> <td>0x2</td><td>2<sup>13</sup></td><td>0x3</td><td>2<sup>14</sup></td></tr> <tr> <td>0x4</td><td>2<sup>15</sup></td><td>0x5</td><td>2<sup>16</sup></td></tr> <tr> <td>0x6</td><td>2<sup>17</sup></td><td>0x7</td><td>2<sup>18</sup></td></tr> <tr> <td>0x8</td><td>2<sup>19</sup></td><td>其它</td><td>2<sup>4</sup></td></tr> </table>	值	周期	值	周期	0x0	2 <sup>4</sup>	0x1	2 <sup>12</sup>	0x2	2 <sup>13</sup>	0x3	2 <sup>14</sup>	0x4	2 <sup>15</sup>	0x5	2 <sup>16</sup>	0x6	2 <sup>17</sup>	0x7	2 <sup>18</sup>	0x8	2 <sup>19</sup>	其它	2 <sup>4</sup>	0x1
值	周期	值	周期																								
0x0	2 <sup>4</sup>	0x1	2 <sup>12</sup>																								
0x2	2 <sup>13</sup>	0x3	2 <sup>14</sup>																								
0x4	2 <sup>15</sup>	0x5	2 <sup>16</sup>																								
0x6	2 <sup>17</sup>	0x7	2 <sup>18</sup>																								
0x8	2 <sup>19</sup>	其它	2 <sup>4</sup>																								

MTC_STABLE	位	描述	初始状态
RESERVED	[31:28]	保留。	0xF
DOMAIN_ETM	[27:24]	内存电源稳定计数器，用于domain ETM。	0xF
DOMAIN_S	[23:20]	内存电源稳定计数器，用于domain S。	0xF
DOMAIN_F	[19:16]	内存电源稳定计数器，用于domain F。	0xF
DOMAIN_P	[15:12]	内存电源稳定计数器，用于domain P。	0xF

DOMAIN_I	[11:8]	内存电源稳定计数器，用于domain I。	0xF
DOMAIN_V	[7:4]	内存电源稳定计数器，用于domain V。	0xF
DOMAIN_TOP	[3:0]	内存电源稳定计数器，用于domain TOP。	0xF

MTC\_STABLE 代表若干外部振荡器（或时钟）周期。当子块从 MTC 模式返回到正常操作模式时，内部电源的稳定需要一段时间。这期间必须大于 200ns，并可以估计出使用 MTC\_STABLE 和 OSC\_FREQ 寄存器的值。

### 3.4.13 其他控制寄存器

寄存器	地址	读/写	描述	复位值
OTHERS	0x7E00_F900	读/写	其他控制寄存器。	0x0000_801E

OTHERS	位	描述	初始状态
RESERVED	[31:17]	保留。	0x0000
USB_SIG_MASK	[16]	屏蔽USB信号。 (在USB PHY被使用前，该位必须进行设置)	0
USB_STATE	[15]	USB PHY电源状态，只读位（1：正常，0：掉电）	1
RESERVED	[14]	不改变。	0
CLEAR_DBGACK	[13]	当该区域为1时，清除DBGACK信号。ARM1176声明DBGACK信号，以说明系统进入调试状态。如果DBGACK信号被声明，该状态被保留在SYSCON里，直到软件使用该区域而被清除。	0
CLEAR_BATF_INT	[12]	该位被设置时，清除中断所引起的电池故障。	0
RESERVED	[11:7]	不改变。	0x00
SYNCMUXSEL	[6]	SYNCMUX选择（0：MOUTMPLL，1：DOUTAPLL）。	0
RESEVED	[5:3]	不改变。	0x3
SPNIDEN	[2]	安全特权的非入侵式使能调试。在 ARM1176 的安全区中，该区域的使能和禁用非入侵式调试。如果该位为‘1’，非入侵式调试在所有非安全模式下被允许。否则，非入侵式调试在所有安	1



		全特权的模式下不被允许。非入侵式调试被允许，是在安全使用模式下，根据 ARM1176 的 SUNIDEN 位决定的。	
SPIDEN	[1]	安全特权的入侵式使能调试。在ARM1176的安全区中，该区域的使能和禁用入侵式调试。如果该位为‘1’，入侵式调试在所有安全模式下被允许。  否则，入侵式调试在任何安全特权的模式下不被允许。入侵式调试被允许，是在安全使用模式下，根据ARM1176的SUNIDEN位决定的。	1
CP15DISABLE	[0]	禁用写评到一些系统控制处理器ARM1176的寄存器。（0：使能，1：禁用）	0

3.4.14 状态寄存器

寄存器	地址	读/写	描述	复位值
RST_STAT	0x7E00_F904	读	复位状态寄存器。	0x0000_0001
WAKEUP_STAT	0x7E00_F908	读/写	唤醒状态寄存器。	0x0000_0000
BLK_PWR_STAT	0x7E00_F90C	读	块电源状态寄存器。	0x0000_007F

RST_STAT	位	描述	初始状态
RESERVED	[31:6]	保留。	0x0000_000
SW_RESET	[5]	通过SWRESET进行软件复位。	0
RESERVED	[4]	保留。	0
SLEEP_WAKEUP	[3]	通过睡眠模式复位唤醒。	0
WDT_RESET	[2]	通过WDTRST进行看门狗定时器复位。	0
WARM_RESET	[1]	通过XnWRESET进行温复位。当睡眠模式的唤醒源是XnWRESET时，该区域不被设置。	0
HW_RESET	[0]	通过XnRESET进行外部复位。	1

WAKEUP_STAT	位	描述	初始状态
RESERVED	[31:12]	保留。	0x0000_0
MMC2_WAKEUP	[11]	通过MMC2唤醒，通过写1被清除。	0
MMC1_WAKEUP	[10]	通过MMC1唤醒，通过写1被清除。	0
MMCO_WAKEUP	[9]	通过MMC0唤醒，通过写1被清除。	0
HSI_WAKEUP	[8]	通过HSI唤醒，通过写1被清除。	0
WRESET_WAKEUP	[7]	通过温复位唤醒，通过写1被清除。	0
BATFLT_WAKEUP	[6]	通过电池故障唤醒，通过写1被清除	0
MSM_WAKEUP	[5]	通过MSM 调制解调器唤醒，通过写1被清除。	0
KEY_WAKEUP	[4]	通过键盘唤醒，通过写1被清除。	0
TS_WAKEUP	[3]	通过触摸屏唤醒，通过写1被清除。	0
RTC_TICK_WAKEUP	[2]	通过嘀嗒信号中断唤醒，通过写1被清除。	0
RTC_ALARM_WAKEUP	[1]	通过RTC警告唤醒，通过写1被清除。	0
EINT_WAKEUP	[0]	通过外部中断唤醒，通过写1被清除。	0

BLK_PWR_STAT	位	描述	初始状态
RESERVED	[31:7]	保留。	0x0000_000
BLK_ETM	[6]	块ETM电源准备就绪。	1
BLK_S	[5]	块 S 电源准备就绪。	1
BLK_F	[4]	块 F 电源准备就绪。	1
BLK_P	[3]	块 P 电源准备就绪。	1
BLK_I	[2]	块 I 电源准备就绪。	1
BLK_V	[1]	块 V 电源准备就绪。	1
BLK_TOP	[0]	块 TOP 电源准备就绪	1

### 3.4.15 消息寄存器

寄存器	地址	读/写	描述	复位值
INFORM0	0x7E00_FA00	读/写	消息寄存器 0。	0x0000_0000
INFORM1	0x7E00_FA04	读/写	消息寄存器 1。	0x0000_0000
INFORM2	0x7E00_FA08	读/写	消息寄存器 2。	0x0000_0000
INFORM3	0x7E00_FA0C	读/写	消息寄存器 3。	0x0000_0000

INFORMn	位	描述	初始状态
INFORM	[31:0]	用户定义消息。通过声明 XnRESET 引脚，INFORM0~3 寄存器被清除。	0x0000_0000

### 3.5 部分应用程序

下面是系统控制器应用的一部分程序。有分别是系统时钟控制的程序，也有系统电源管理控制的部分程序。对这些程序进行分析将有助于更进一步理解系统控制器。

```
//用于获得指令时钟信息
//输入：NONE
//输出：NONE
void SYSC_GetClkInform( void)
{
    u8 muxAp11, muxMp11, muxSync;
    u8 divAp11, divHclx2, divHclk, divPclk;
    u16 pllM, pllP, pllS;
    u32 temp;

    ////
    // 时钟分频频率
    temp = Inp32(rCLK_DIV0);
```

```

divAp11 = temp & 0xf;
divHclkx2 = (temp>>9) & 0x7;
divHclk = (temp>>8) & 0x1;
divPclk = (temp>>12) & 0xf;

////
// 操作模式
temp = Inp32(rOTHERS);
temp = (temp>>8)&0xf;
if(temp)
{
    g_SYNCACK = 1;
}
else
{
    g_SYNCACK = 0;
}

////
// ARMCLK
muxAp11 = Inp32(rCLK_SRC) & 0x1;
if(muxAp11) //FOUT
{
    temp = Inp32(rAPLL_CON);
    p11M = (temp>>16)&0x3ff;
    p11P = (temp>>8)&0x3f;
    p11S = (temp&0x7);

    g_APLL = ((FIN>>p11S)/p11P)*p11M;

```

```

}
else    //FIN
{
    g_APLL = FIN;
}

g_ARMCLK = g_APLL/(divAp11+1);

////
// HCLK
muxSync = (Inp32(rOTHERS)>>7) & 0x1;
if(muxSync) //synchronous mode
{
    g_HCLKx2 = g_APLL/(divHclkx2+1);

    temp = Inp32(rMPLL_CON);
    p11M = (temp>>16)&0x3ff;
    p11P = (temp>>8)&0x3f;
    p11S = (temp&0x7);

    g_MPLL = ((FIN>>p11S)/p11P)*p11M;
}
else
{
    muxMp11 = (Inp32(rCLK_SRC)>>1) & 0x1;
    if(muxMp11) //FOUT
    {
        temp = Inp32(rMPLL_CON);
        p11M = (temp>>16)&0x3ff;

```

```

        pllP = (temp>>8)&0x3f;
        pllS = (temp&0x7);

        g_MPLL = ((FIN>>pllS)/pllP)*pllM;
    }
    else    //FIN
    {
        g_MPLL = FIN;
    }
    g_HCLKx2 = g_MPLL/(divHclkx2+1);
}

g_HCLK = g_HCLKx2/(divHclk+1);

////
// PCLK
g_PCLK = g_HCLKx2/(divPclk+1);

return;
}

```

/\*控制 PLL 输出频率 (APLL, MPLL, EPLL):  $F_{out} = (mdiv * Fin) / (pdiv \times 2^{sdiv})$ ,  $F_{out} = ((mdiv+k/2^{16}) * Fin) / (pdiv \times 2^{sdiv})$ \*/

```

//输入:          ePLL  : APLL, MPLL, EPLL
//              uMdiv : Mdiv 值 ( 56 ~ 1023), ( 13 ~ 255)
//              uPdiv : Pdiv 值 ( 1~63)
//              uSdiv : Sdiv 值 ( 0~5 )
//              uKdiv : PLL9025X (Not Used, 0), PLL9024X(0~65535)
//输出:  NONE

```

```

void SYSC_SetPLL(PLL_eTYPE ePLL, u32 uMdiv, u32 uPdiv, u32 uSdiv, u32 uKdiv)
{
    u32 temp, uRegValue;
    switch(ePLL)
    {
        case eAPLL:
            // 检测分频器的值
            if( uMdiv < 56 || uMdiv > 1023)
            {
                printf(" Wrong Mdiv Value, Correct Value Range = (56 ~ 1023) (%d)\n", uMdiv);
            }
            if( uPdiv < 1 || uPdiv > 63)
            {
                printf(" Wrong Pdiv Value, Correct Value Range = (1 ~ 63) (%d)\n", uPdiv);
            }
            if( uSdiv > 5)
            {
                printf(" Wrong Sdiv Value, Correct Value Range = (0 ~ 5) (%d)\n", uSdiv);
            }
            //检测 Fvco 范围

            temp = ((FIN/uPdiv)*uMdiv)/1000000;
            if( temp <1000 || temp > 2000)
            {
                #if 0//EVT1
                    printf(" Please select the proper M,P,S divider value\n");
                    printf(" Fvco Range = (1000MHz ~ 2000MHz), Current Value is (%d)MHz\n", temp);
                #endif
            }
    }
}

```

```
uRegValue = (u32)((((u32)(0x1<<31))|(uMdiv<<16)|(uPdiv<<8)|(uSdiv<<0));
Outp32(rAPLL_CON, uRegValue);
break;
```

```
case eMPLL:
```

```
// 检测分频器的值
```

```
if( uMdiv < 56 || uMdiv > 1023)
```

```
{
```

```
    printf(" Wrong   Mdiv Value, Correct Value Range = (56 ~ 1023) (%d)\n", uMdiv);
```

```
}
```

```
if( uPdiv < 1 || uPdiv > 63)
```

```
{
```

```
    printf(" Wrong   Pdiv Value, Correct Value Range = (1 ~ 63) (%d)\n", uPdiv);
```

```
}
```

```
if( uSdiv > 5)
```

```
{
```

```
    printf(" Wrong   Sdiv Value, Correct Value Range = (0 ~ 5) (%d)\n", uSdiv);
```

```
}
```

```
// 检测 Fvco 范围
```

```
temp = ((FIN/uPdiv)*uMdiv)/1000000;
```

```
if( temp <1000 || temp > 2000)
```

```
{
```

```
#if 0 //EVT1
```

```
    printf(" Please select the proper M,P,S divider value\n");
```

```
    printf(" Fvco Range = (1000MHz ~ 2000MHz),   Current Value is (%d)MHz\n", temp);
```

```
#endif
```

```
}
```



```
uRegValue = (u32)((((u32)(0x1<<31))|(uMdiv<<16)|(uPdiv<<8)|(uSdiv<<0));
Outp32(rMPLL_CON, uRegValue);
break;
```

case eEPLL:

```
// 检测分频器的值
if( uMdiv < 13 || uMdiv > 255)
{
    printf(" Wrong   Mdiv Value, Correct Value Range = (56 ~ 1023) (%d)\n", uMdiv);
}
if( uPdiv < 1 || uPdiv > 63)
{
    printf(" Wrong   Pdiv Value, Correct Value Range = (1 ~ 63) (%d)\n", uPdiv);
}
if( uSdiv > 5)
{
    printf(" Wrong   Sdiv Value, Correct Value Range = (0 ~ 5) (%d)\n", uSdiv);
}
if( uKdiv > 65535)
{
    printf(" Wrong   Kdiv Value, Correct Value Range = (0 ~ 65535) (%d)\n", uKdiv);
}
// 检测 Fvco 范围

temp = ((FIN/uPdiv)*(uMdiv+uKdiv>>16))/1000000;
if( temp < 60 || temp > 250)
{
    printf(" Please select the proper M,P,S divider value\n");
}
```

```

        printf(" Fvco Range = (60MHz ~ 250MHz),   Current Value is (%d)MHz\n", temp);
    }

    Outp32(rEPLL_CON1, uKdiv);
    uRegValue = (u32)(((u32)(0x1<<31))|(uMdiv<<16)|(uPdiv<<8)|(uSdiv<<0));
    Outp32(rEPLL_CON0, uRegValue);
    break;
}

// 得到信息
SYSC_GetClkInform();
}
//读取 MEM_CFC_STAT 寄存器
void SYSC_RdMEMCFGSTAT( void )
{
    u32 uRegValue;
    u32 uTemp;

    uRegValue = Inp32(rMEM_CFG_STAT);

    // EBI 优先权配置
    uTemp = ( uRegValue >>15 ) & 0x1;
    printf(" Current EBI Priority Scheme   (0: Fixed, 1: Circular) :    %d \n",  uTemp);
    uTemp = (uRegValue >> 12) & 0x7 ;
    printf(" Current EBI Fixed Priority setting                :    %d \n",  uTemp);

    // CF I/F
    uTemp = ( uRegValue >>10 ) & 0x1;
    printf(" Current CF I/F Setting   (0: EBI, 1: Independet)      :    %d \n",  uTemp);
}

```

```

// NAND 类型设置
uTemp = ( uRegValue >>9 ) & 0x1;
printf(" Current NAND Type (0:OneNAND, 1: NAND)           :    %d \n",  uTemp);
uTemp = ( uRegValue >>3 ) & 0x1;
printf(" Current NAND Init Setting  (0: Normal NAND, 1: Advanced NAND) :    %d \n",  uTemp);
uTemp = ( uRegValue >>2 ) & 0x1;
printf(" Show address cycle init. setting of NAND      :    %d \n",  uTemp);
uTemp = ( uRegValue >>0 ) & 0x1;
printf(" Show NAND Page Size      :    %d \n",  uTemp);

uTemp = ( uRegValue >>8 ) & 0x1;
printf(" Current CS0 Bus Width  (0: 8-bit, 1: 16-bit)      :    %d \n",  uTemp);
uTemp = ( uRegValue >>1 ) & 0x1;
printf(" Show CS0 Bus width (init. Setting)  (0:8bit, 1: 16bit)   :    %d \n",  uTemp);

uTemp = ( uRegValue >>7 ) & 0x1;
printf(" NAND Booting    (0: not Used, 1: used)           :    %d \n",  uTemp);

uTemp = ( uRegValue >>5 ) & 0x3;
printf(" Current Booting Type  (0: NFCON, 1: SROMC, 2: ONDC,  3: Internal ROM) :    %d \n",
uTemp);

uTemp = ( uRegValue >>4 ) & 0x1;
printf(" Current ADDR Expand  (0: Used MEM1 Data, 1:Used MEM0 Addr) :    %d \n",  uTemp);

}

```

## 4 存储器子系统

S3C6410 存储器包括七个存储控制器，一个 SRAM 控制器，两个 OneNAND 控制器，一个 NAND 闪存控制器，一个 CF 控制器，和两个 DRAM 控制器。通过使用 EBI，静态存储控制器和 16 位 DRAM 控制器共享存储器端口 0。

### 4.1 存储器子系统的特性

S3C6410 存储器子系统的特性如下：

- (1) 存储器子系统有一个64位AXI从属器接口，一个32位AXI从属器接口，一个32位AHB主控制器接口，两个32位从属器接口，其中一个用于数据传输，另一个用于SFR设置和一个用于DMC SFR设置的APB接口。
- (2) 存储器子系统从系统控制器获得导入方法和CS选择信息。
- (3) 内部AHB数据总线将32位AHB从属器数据总线和SRAMC, 两个 OneNANDC 和 NFCON连接起来。
- (4) 内部AHB SFR总线将32位AHB从属器SFR总线和SRAMC、两个 OneNANDC、CFCON 和NFCON连接起来。
- (5) 内部 AHB 主控制器总线用于 CFCON。
- (6) DMC0用32位AXI从属器接口和APB接口。
- (7) DMC1用64位AXI从属器接口和APB接口。
- (8) 存储器端口0通过使用EBI（外部总线接口）共享。
- (9) 仅DMC1启动用存储器端口1。
- (10) 支持使用NAND闪存或者OneNAND。
- (11) 对于CFCON，支持独立端口。
- (12) 存储器端口0中Xm0CSn[1:0]专用于SRAMC。
- (13) 存储器端口 0 中 Xm0CSn[7:6]专用于 DMC0。
- (14) 选择NAND闪存或OneNAND导入设备，nCS2用于访问导入的媒体。
- (15) EBI模块支持AMBA AXI 3.0低电源接口(CSYSREQ、CACTIVE、和CSYSACK)来阻止存储控制器来访问内存。
- (16) 通过在系统控制器中设置，存储器端口1的数据引脚[31:16]能用做端口0的地址引脚[31:16]。

(17) EBI模块支持通过存储控制器使用pad接口 (DMC0, SROMC, 两个OneNANDC, CFCON, NFCON)。

(18) 通过改变优先权来决定哪个能拥有pad接口。

(19) EBI和包含一个三线接口、EBIREQ、EBIGNT和EBIBACKOFF, 和所有活动的高位的存储器之间相互通信:

- EBIREQ信号被存储控制器访问来指示它们需要外部总线访问。
- 各自的EBIGNT被发送到高优先权的存储控制器。
- EBI 输出EBIBACKOFF来发送信号, 存储控制器必须完成当前传输释放总线。

(20) EBI保持被授权的存储控制器的跟踪, 并且在它授权给下一个存储控制器之前, 等待从存储控制器到闪存的传输。如果高优先权的存储控制器请求总线, 那么EBIBACKOFF通知当前被授权的控制器来尽快结束当前传输。

EBI模块框图如图4-1所示, 通过EBI的存储器接口如图4-2所示。

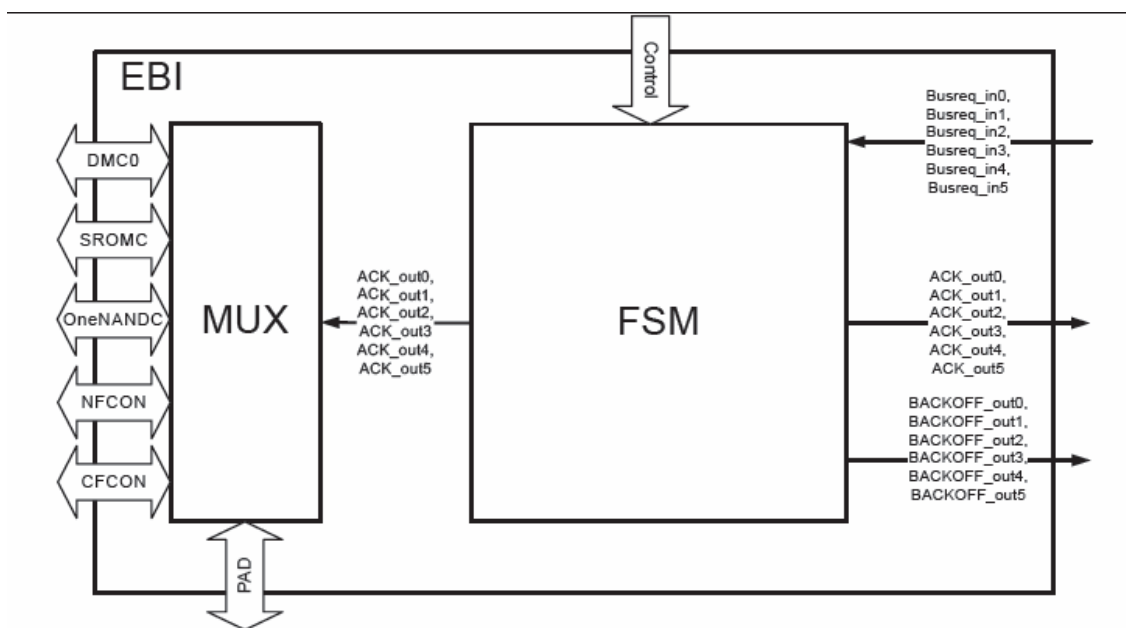
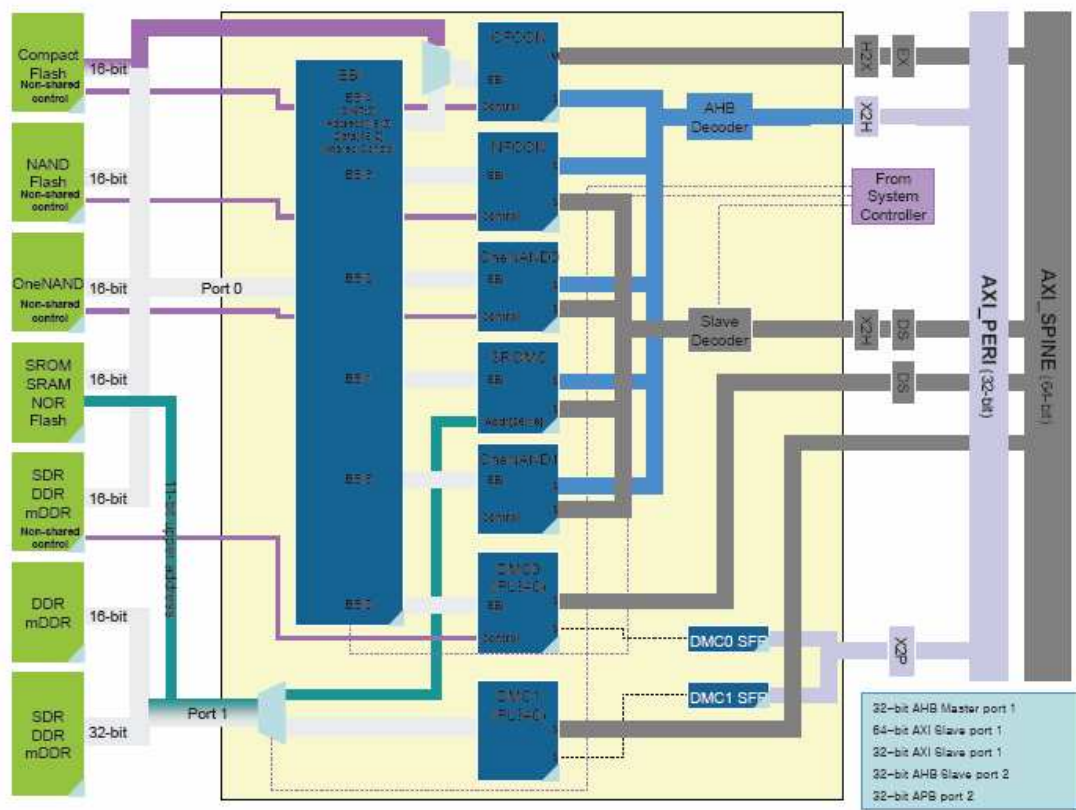


图4-1 EBI模块图



## 4.2 功能描述

存储器子系统能通过系统控制器进行配置。系统控制器中，存储器构造寄存器名称是MEM\_SYS\_CFG。该寄存器的地址是0x7E00F120。

系统控制器发送信息如下:

#### 4.2.1. 闪存信息

- (1) NAND闪存导入启动
- (2) 高级NAND闪存
- 0= 正常 NAND / 1= 高级 NAND。
- (3) 地址周期。

- 正常NAND，0= 3个周期/ 1= 4个周期。
- 高级NAND，0= 4个周期/ 1= 5个周期。

(4) NAND闪存数据总线宽度  
8位。

- (5) 页大小选择
- 正常NAND，0= 256字节/ 1= 512字节。
  - 高级NAND，0= 1KB/ 1= 2KB。

(6) 端口信息

①SROM 数据总线宽度

- 可以通过 SROM控制器 SFR设定改变。
- 0 : 8位, 1 : 16位。

②地址扩展

- 0= 用 Xm1DATA[31:16] 作为高半字数据来自存储器端口1。
- 1= 用Xm1DATA[31:16] 作为高半字地址来自存储器端口0。  
从端口1的数据引脚[31:16]为端口0借用地址位[26:16]。

③导入位置

导入位置信息能被在MEM\_CFG\_STAT[6:5]检查，如表4-1所示。

表4-1 导入位置

CfgBootLoc[1:0]	导入位置
2' b00	NFCON的Stepping Stone区域
2' b01	SROMC CS0
2' b10	OneNANDC CS0
2' b11	内部 ROM

④存储器端口0 CS选择

- 设置存储器端口0的静态存储芯片选择多路复用。
- 忽略MP0\_CS\_SEL[0]和MP0\_CS\_SEL[2]的设置。通过XSELNAND引脚值区别OneNANDC和NFCON。当XSELNAND是0，则OneNANDC被选择；当XSELNAND是1，则NFCON被选中。
- 当选择NAND导入(XOM[4:3] = 00)时，MP0\_CS\_SEL[1]和MP0\_CS\_SEL[3] 的设置值被忽略。Xm0CSn[2]

和Xm0CSn[3]用做NFCN CS0和NFCN CS1。这种情况下，XSELNAND必须设置为1。

- 当OneNAND导入 (XOM[4:1] = 0110) 时，MP0\_CS\_SEL[1]和MP0\_CS\_SEL[3]的设置值被忽略。Xm0CSn[2]和Xm0CSn[3]被用作OneNAND CS0和OneNAND CS1。这种情况下，XSELNAND必须设置为0。如表4-2所示。

表 4-2 存储器端口 0 CS 选择

MP0_CS_SEL[5:0]	=0	=1	=2	=3
{[1], XSELNAND}	SROMC CS2	SROMC CS2	OneNANDC CS0	NFCN CS0
{[3], XSELNAND}	SROMC CS3	SROMC CS3	OneNANDC CS1	NFCN CS1
[4]	SROMC CS4	CFCN CS0		
[5]	SROMC CS5	CFCN CS1		

⑤为CFCN选择独立的端口

- 0= CFCN 使用 EBI的共享端口。
- 1= CFCN 使用独立端口。

⑥CKE初始值 (SPCONSLP[4] (0x7F0088B0))

- 0 = 复位后，存储器端口0的Xm0CKE 和端口1的Xm1CKE的初始化值为0。
- 1 = 复位后，存储器端口0的Xm0CKE 和端口1的Xm1CKE的初始化值为1。

⑥EBI

- 优先类型  
0=固定优先类型 / 1= 循环优先。
- 固定优先顺序

如表4-3所示，显示了固定的优先顺序。

表 4-3 固定优先顺序

CfgFixPriTyp[2:0]	1st	2nd	3rd	4th	5th	6th
	DMC0	SROMC	OneNANDC CS0	OneNANDC CS1	NFCN	CFCN
1	DMC0	OneNANDC	OneNANDC	SROMC	NFCN	CFCN



		CS0	CS1			
2	DMC0	OneNANDC CS1	NFCON	SROMC	OneNANDC CS0	CFCON
3	DMC0	NFCON	SROMC	OneNANDC CS0	OneNANDC CS1	CFCON
4	DMC0	CFCON	SROMC	OneNANDC CS0	OneNANDC CS1	NFCON
5	SROMC	DMC0	OneNANDC CS0	OneNANDC CS1	NFCON	CFCON

(7) 总线信息

①MP0\_QOS\_OVERRIDE[15:0]

- 设置QoS 取代ID到DMC0。
- 系统控制器中，QOS\_OVERRIDE0 控制该信息。
- 该寄存器地址是 0x7E00F124。

②MP1\_QOS\_OVERRIDE[15:0]

- 设置 QoS 取代 ID到DMC1。
- 系统控制器中QOS\_OVERRIDE1控制该信息。
- 该寄存器地址是 0x7E00F128。

4.3 EBI 接口

EBI，作为外设，当它们空闲时，依靠存储控制器来释放它们的外部请求。如图4-3所示，一个简单的相互通信的例子。在这个例子里，一个设备请求外部总线，因为当先没有其它设备请求总线，所以立即产生，

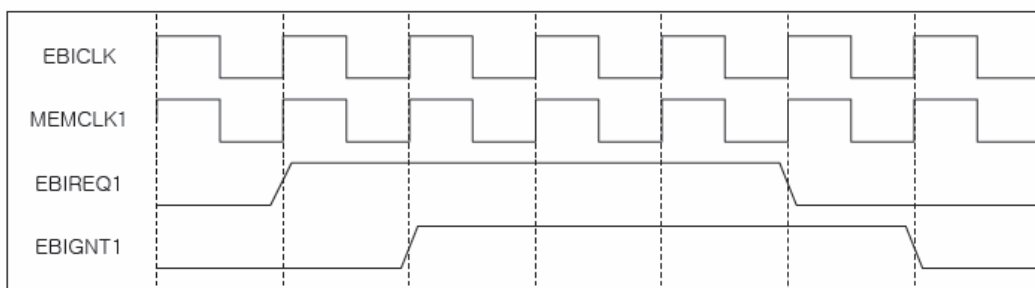


图4-3 EBIREQ, EBIGANT 信号

如果一个高优先权的设备请求总线，且是一个低优先权的设备控制该外部总线时，则EBIBACKOFF通知低优先权的设备尽快释放。

在图4-4中，一个高优先权的设备要求总线，不久这个设备就给予总线的权限。EBIBACKOFF通知设备尽早结束访问。设备一被给予总线权限并完成传输。当传输完成，设备二被给予权限来完成被中断的传输。EBIREQ2信号必须降低至少一个时钟周期，之后被释放。

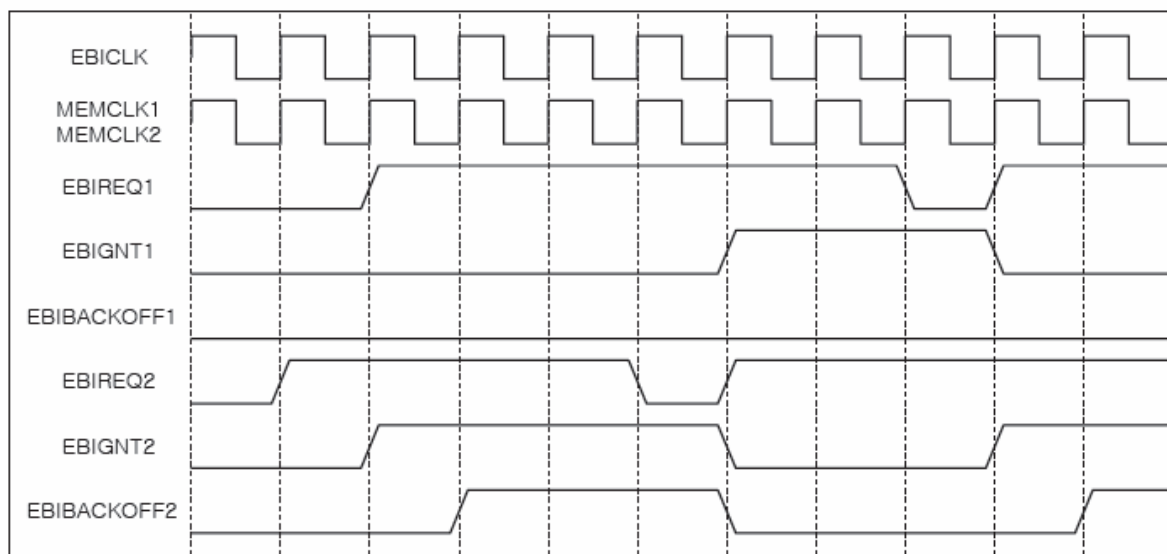


图4-4 EBIBACKOFF 信号

## 5 DRAM 控制器

基于 ARM PrimeCell CP003 AXI DMC(PL340)的 DRAM 控制器，来自 ARM PrimeCell CP003 AXI 动态存储器控制(PL340)。最初的 AMBA APB 3.0 端口主要用于可编程配置寄存器，它是利用 AxiToApb 进行转换的，使一个连接 APB 主端口的 AXI 从端口得以实现。

DRAM 控制器有 AMBA AXI 兼容总线用于设计其配置寄存器和访问 SDRAM。在 PL340 配置寄存器中，DRAM 控制器可以通过写芯片配置、ID 配置和存储器定时参数来进行编程。在用户配置寄存器中，两个较低位主要用于选择存储器的类型。

DRAM 控制器可以直接接收一个 SDRAM 或 DRAM 控制器本身的指令。通过写指令到直接指令寄存器，DRAM 控制器可发送像 ‘Prechargeall’，‘Autorefresh’，‘NOP’ 和 ‘MRS’（‘EMRS’）这样的指令到 SDRAM。

在自动刷新计数器中，当刷新计数达到刷新周期的值时，控制器便会发出一个自动刷新指令对 SDRAM 进行周期性地刷新。

### 5.1 DRAM 控制器的特性

DRAM 控制器的特性如下：

- (1) 支持 SDR SDRAM，动态 SDR SDRAM，DDR SDRAM 和动态 DDR SDRAM。
- (2) 支持两个外部存储器芯片。
- (3) 支持 32/64 位 AMBA AXI 总线。
- (4) 支持 16/32 位存储器总线。
- (5) 地址空间：每端口达到 512MB。
- (6) 在 AXI 总线和外部存储器总线间，支持异步操作。
- (7) 预动态和预充电的电源中断。
- (8) 服务质量的特性适合于低延迟传输。
- (9) 利用最优化的外部存储器总线。
- (10) 支持通过设置 SFR 来选择外部存储器的类型。
- (11) 支持 8 位重要地址。

- (12) 支持带有深度为 8 的写数据交叉。
- (13) 支持 2 种重要的唯一存取传输。
- (14) 用 SFR 可配置存储器的存取时间。
- (15) 支持扩展 MRS (EMRS) 集。
- (16) 存储器端口 1, CKE 可以单独控制。
- (17) 存储器端口 1, 不支持 16 位 SDR SDRAM 和动态 SDR SDRAM。

如图 5-1 所示，显示了 PL340 DRAM 控制器的结构框图。

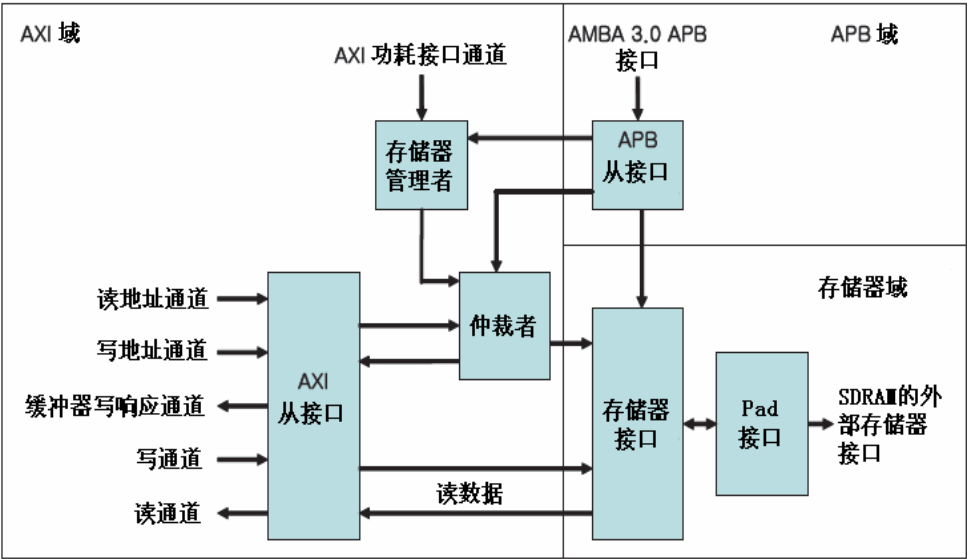


图 5-1 DRAM 控制器框图

## 5.2 SDRAM 存储器接口

DRAM 控制器最多只能支持两个同一类型的芯片，每个芯片可分配最多 256 MB 的地址空间。所有芯片在相同的端口共享所有引脚，除了时钟启动信号和片选信号。例如：DDR SDRAM 存储器接口的连接。动态 DDR SDRAM 可连接类似的 DDR SDRAM 内存。SDR SDRAM 和动态 SDR SDRAM 可连接类似的 DDR SDRAM，除了 DQS 引脚。外部存储器引脚配置如表 2-16、表 2-17 所示。

通过 SPCONSLP[4]来控制复位 CKE 的值。如果值为 0，复位时 Xm0CKE 和 Xm1CKE 为 0。如果值为 1，复位时 Xm0CKE 和 Xm1CKE 为 1。

表 5-1 存储器端口 0 引脚描述

信号	类型	描述
Xm0SCLK	输入	存储器时钟
Xm0SCLKn	输入	存储器时钟（负面）
Xm0CKE	输入	时钟启动每个芯片
Xm0CSn[6:7]	输入	芯片选择每个芯片（低有效）
Xm0RAS	输入	行地址滤波（低有效）
Xm0CAS	输入	列地址滤波（低有效）
Xm0WEndmc	输入	写使能（低有效）
Xm0ADDR[13:0]	输入	地址总线
Xm0ADDR[15:14]	输入	块选择
Xm0DATA[15:0]	输入	数据总线
Xm0DQM[1:0]	输入	数据总线屏蔽位
Xm0DQS[1:0]	输入	数据滤波输入，仅 DDR 和 mDDR

表 5-2 存储器端口 1 引脚描述

信号	类型	描述
Xm1SCLK	输入	存储器时钟
Xm1SCLKn	输入	存储器时钟（负面）
Xm1CKE[1:0]	输入	时钟启动每个芯片
Xm1CSN[1:0]	输入	芯片选择每个芯片（低有效）
Xm1RAS	输入	行地址滤波（低有效）
Xm1CAS	输入	列地址滤波（低有效）
Xm1WEN	输入	写使能（低有效）
Xm1ADDR[13:0]	输入	地址总线
Xm1ADDR[15:14]	输入	块选择
Xm1DATA[31:0]	输入	数据总线
Xm1DQM[3:0]	输入	数据总线屏蔽位

Xm1DQS[3:0]	输入	数据滤波输入，只有 DDR 和 mDDR
-------------	----	----------------------

### 5.3 SDRAM 初始化顺序

上电复位时，软件必须初始化 DRAM 控制器，SDRAM 的每一项都连接到 DRAM 控制器。仅以 SDRAM 的数据表为启动程序。

#### 1. DRAM 控制器初始化顺序

- (1) 以 ‘3'b100’ 执行 memc\_cmd，使得 DRAM 控制器输入 ‘配置’ 状态。
- (2) 写存储器时间参数，芯片配置和 ID 配置寄存器。
- (3) 等待 200  $\mu$ s 来使 SDRAM 电源和时钟稳定。当 CPU 开始工作时，电源和时钟已经被稳定下来。
- (4) 执行存储器初始化顺序。
- (5) 以 ‘3'b000’ 执行 memc\_cmd，使得 DRAM 控制器输入 ‘准备’ 状态。
- (6) 在 memc\_stat 中检查存储器状态域，直到存储器状态变为 ‘2'b01’，即 ‘准备’。

#### 2. SDR/动态 SDR SDRAM 初始化顺序

- (1) 在 direct\_cmd，以 ‘2'b10’ 执行 mem\_cmd，使得 DRAM 控制器产生 ‘NOP’ 存储器命令。
- (2) 在 direct\_cmd，以 ‘2'b00’ 执行 mem\_cmd，使得 DRAM 控制器产生 ‘Prechargeall’ 存储器命令。
- (3) 在 direct\_cmd，以 ‘2'b11’ 执行 mem\_cmd，使得 DRAM 控制器产生 ‘Autorefresh’ 存储器命令。
- (4) 在 direct\_cmd，以 ‘2'b11’ 执行 mem\_cmd，使得 DRAM 控制器产生 ‘Autorefresh’ 存储器命令。
- (5) 如果存储器类型是移动 SDR SDRAM，
  - 在 direct\_cmd，以 ‘2'b10’ 执行 mem\_cmd，使得 DRAM 控制器产生 ‘MRS’ 存储器命令。
  - EMRS 块地址必须被设置。
- (6) 在 direct\_cmd，以 ‘2'b10’ 执行 mem\_cmd，使得 DRAM 控制器产生 ‘MRS’ 存储器命令。  
MRS 块地址必须被设置。

#### 3. DDR/移动 DDR SDRAM 初始化顺序

- (1) 在 direct\_cmd，以 ‘2'b10’ 执行 mem\_cmd，使得 DRAM 控制器产生 ‘NOP’ 存储器命令。

- (2) 在 direct\_cmd, 以 ‘2'b00’ 执行 mem\_cmd, 使得 DRAM 控制器产生 ‘Prechargeall’ 存储器命令。
- (3) 在 direct\_cmd, 以 ‘2'b11’ 执行 mem\_cmd, 使得 DRAM 控制器产生 ‘Autorefresh’ 存储器命令。
- (4) 在 direct\_cmd, 以 ‘2'b10’ 执行 mem\_cmd, 使得 DRAM 控制器产生 ‘MRS’ 存储器命令。  
EMRS 块地址必须被设置。
- (5) 在 direct\_cmd, 以 ‘2'b10’ 执行 mem\_cmd, 使得 DRAM 控制器产生 ‘MRS’ 存储器命令。  
MRS 块地址必须被设置。
- (6) 在 direct\_cmd, 以 ‘2'b11’ 执行 mem\_cmd, 使得 DRAM 控制器产生 ‘Autorefresh’ 存储器命令。
- (7) 在 direct\_cmd, 以 ‘2'b11’ 执行 mem\_cmd, 使得 DRAM 控制器产生 ‘Autorefresh’ 存储器命令。
- (8) 在 direct\_cmd, 以 ‘2'b11’ 执行 mem\_cmd, 使得 DRAM 控制器产生 ‘Prechargeall’ 存储器命令。

5.4 DRAM 寄存器描述

下面以图表的形式将主要介绍 DRAM 控制器中各种寄存器的具体设置。

5.4.1 DRAM 控制状态寄存器

寄存器	地址	读/写	描述	复位值
POMEMSTAT	0x7E000000	读	16位DRAM控制状态寄存器	0xAB0
P1MEMSTAT	0x7E001000	读	32位DRAM控制状态寄存器	0xAB4

PnMEMSTAT	位	描述	初始状态
Reserved	[31:14]	读未定义	
Memory banks	[13:12]	存储块的最大数量，DRAM 控制器支持每一个芯片 00=4 个块	00
Exclusive monitors	[11:10]	高级访问数量检测资源 10=2 个监控器	10
Reserved	[9]	读总是为 0	0
Memory chips	[8:7]	不同芯片最大数量选择，DRAM 控制器能支持： 11=4 个芯片	11

		然而，S3C6410 每个 DRAM 控制器仅使用两个片选信号	
Memory type	[6:4]	SDRAM 类型的 DRAM 控制器支持： 100=支持 SDR SDRAM（标准的或动态的）和 DDR SDRAM（标准或动态的）	100
Memory width	[3:2]	外部存储器的宽度： 00=16 位    01=32 位    10=64 位    11=保留	00/01
Controllor status	[1:0]	DRAM 控制器的状态： 00=配置    01=准备    10=暂停    11=低功耗	00

### 5.4.2 DRAM 控制命令寄存器

寄存器	地址	读/写	描述	复位值
P0MEMCCMD	0x7E000004	写	16 位 DRAM 控制命令寄存器	
P1MEMCCMD	0x7E001004	写	32 位 DRAM 控制命令寄存器	

PnMEMCCMD	位	描述	初始状态
	[31:3]	未定义，写为 0	
Memc_cmd	[2:0]	DRAM 控制器状态的改变： 000=运行    001=睡眠    010=唤醒    011=暂停 100=配置    101~111=保留	

### 5.4.3 直接命令寄存器

寄存器	地址	读/写	描述	复位值
P0 DIRECTCMD	0x7E000008	写	16 位 DRAM 控制直接命令寄存器	
P1 DIRECTCMD	0x7E001008	写	32 位 DRAM 控制直接命令寄存器	

PnDIRECTCMD	位	描述	初始状态
-------------	---	----	------



	[31:22]	未定义，写为 0	
Chip number	[21:20]	位映射到外部存储器芯片的地址位	
Memory command	[19:18]	定义所需命令：  00 = Prechargeall  01 = 自动刷新  10 = MRS或EMRS  11 = NOP	
Bank address	[17:16]	当命令为 MRS 或 EMRS 访问时，位映射到外部存储器块地址位	
	[15:14]	未定义，写为 0	
Address_13_to_0	[13:0]	当命令为 MRS 或 EMRS 访问时，位映射到外部存储器地址位[13:0]	

## 5.4.4 存储配置寄存器

寄存器	地址	读/写	描述	复位值
POMEMCFG	0x7E00000C	读/写	16 位 DRAM 控制存储配置寄存器	0x01_0020
P1MEMCFG	0x7E00100C	读/写	32 位 DRAM 控制存储配置寄存器	0x01_0020

PnMEMCFG	位	描述	初始状态
cke_config	[31]	选择 CKE 控制配置。仅 P1MEMCFG  0=支持一个 CKE 控制  1=支持单独的 CKE 控制	0
保留	[30:23]	读未定义，写为 0。	
Active chips	[22:21]	使刷新命令产生有效，用于存储芯片的数量。它只能产生指令，包括芯片数量配置的定义，在 DRAM 的控制状态寄存器：  00=1 片  01=2 片  10=3 片	00

		11=4 片	
QoS master bits	[20:18]	8 位 AXI ARID 的 4 位编码，用来选择其中的 16QoS 值： 000 = ARID[3:0] 001 = ARID[4:1] 010 = ARID[5:2] 011 = ARID[6:3] 100 = ARID[7:4] 101~111 = 保留	000
Memory burst	[17:15]	执行数据访问的数量编码到 SDRAM，用于每个读和写命令： 000 = 脉冲1 001 = 脉冲2 010 = 脉冲4 011 = 脉冲8 100 = 脉冲16 101~111 = 保留 这个值也必须编写到 SDRAM 模式寄存器，使用 DIRECTCMD 寄存器，必须对它进行匹配	010
Stop_mem_clock	[14]	启动的存储器时钟还没有访问 SDRAM 就被强制停止	0
Auto power down	[13]	当设置自动掉电时，SDRAM 的存储器接口自动位通过使 CKE 无效以进入到省电状态。先入先出命令为空，用于 Power_down_prd 内存时钟周期	0
Power_down_prd	[12:7]	内存时钟周期的数量，用于 SDRAM 自动掉电	000000
AP bit	[6]	自动预充电位的编码位置在内存中的地址： 0=地址位 10 1=地址位 8	0
Row bits	[5:3]	AXI 地址位的编码数包含的行地址： 000=11 位	100

		001=12 位 010=13 位 011=14 位 100=15 位 101=16 位 110=10 位 111=9 位	
Column bits	[2:0]	AXI 地址位的编码数包含的列地址： 000=8 位 001=9 位 010=10 位 011=11 位 100=12 位 101=7 位 110=6 位 111=保留	000

5.4.5 刷新周期寄存器

寄存器	地址	读/写	描述	复位值
POREFRESH	0x7E000010	读/写	16 位 DRAM 控制刷新周期寄存器	0xA60
P1REFRESH	0x7E001010	读/写	32 位 DRAM 控制刷新周期寄存器	0xA60

PnREFRESH	位	描述	初始状态
	[31:15]	读未定义，写为 0	
Refresh perod	[14:0]	在内存时钟周期内，存储器的刷新周期	0xA60

5.4.6 CAS 等待时间寄存器

寄存器	地址	读/写	描述	复位值
POCASLAT	0x7E000014	读/写	16 位 DRAM 控制 CAS 等待时间寄存器	0x6
P1CASLAT	0x7E001014	读/写	32 位 DRAM 控制 CAS 等待时间寄存器	0x6

PnCASLAT	位	描述	初始状态
	[31:4]	读未定义，写为 0	
CAS Latency	[3:1]	内存时钟周期内 CAS 等待时间	011
CAS Half cycle	[0]	CAS 等待时间是否编码，半个内存时钟周期在位[3:1]设定值。 0=零周期偏移到位[3:1]上的值。在 MDDR 和 SDR 模式内，位[0]强制为 0 1=半个周期偏移位[3:1]上的值	0

5.4.7 T\_DQSS 寄存器

寄存器	地址	读/写	描述	复位值
P0T_DQSS	0x7E000018	读/写	16 位 DRAM 控制 t_DQSS 寄存器	0x1
P1T_DQSS	0x7E001018	读/写	32 位 DRAM 控制 t_DQSS 寄存器	0x1

PnT_DQSS	位	描述	初始状态
	[31:2]	读未定义，写为 0	
t_DQSS	[1:0]	写 DQS 到内存时钟周期内	1

5.4.8 T\_MRD 寄存器

寄存器	地址	读/写	描述	复位值
P0T_MRD	0x7E00001C	读/写	16 位 DRAM 控制 t_MRD 寄存器	0x02
P1T_MRD	0x7E00101C	读/写	32 位 DRAM 控制 t_MRD 寄存器	0x02

PnT_MRD	位	描述	初始状态
	[31:7]	读未定义，写为 0	
t_MRD	[6:0]	内存时钟周期内，设置模式寄存器命令时间	0x02

5.4.9 T\_RAS 寄存器

寄存器	地址	读/写	描述	复位值
P0T_RAS	0x7E000020	读/写	16 位 DRAM 控制 t_RAS 寄存器	0x7
P1T_RAS	0x7E001020	读/写	32 位 DRAM 控制 t_RAS 寄存器	0x7

PnT_RAS	位	描述	初始状态
	[31:4]	读未定义，写为 0	
t_RAS	[3:0]	内存时钟周期内，设置 RAS 到预充电延迟	0x7

5.4.10 T\_RC 寄存器

寄存器	地址	读/写	描述	复位值
P0T_RC	0x7E000024	读/写	16 位 DRAM 控制 t_RC 寄存器	0xB
P1T_RC	0x7E001024	读/写	32 位 DRAM 控制 t_RC 寄存器	0xB

PnT_ RC	位	描述	初始状态
	[31:4]	读未定义，写为 0	
t_ RC	[3:0]	内存时钟周期内，设置有效组件 x 到有效组件 x 延迟	0xB

### 5.4.11 T\_RCD 寄存器

寄存器	地址	读/写	描述	复位值
P0T_RCD	0x7E000028	读/写	16 位 DRAM 控制 t_ RCD 寄存器	0x1D
P1T_RCD	0x7E001028	读/写	32 位 DRAM 控制 t_ RCD 寄存器	0x1D

PnT_ RCD	位	描述	初始状态
	[31:6]	读未定义，写为 0	
scheduled_RCD	[5:3]	ACLK 周期 3 内，设置 RAS 到 CAS 最小延迟	011
t_ RCD	[2:0]	内存时钟周期内，设置 RAS 到 CAS 最小延迟	101

### 5.4.12 T\_RFC 寄存器

寄存器	地址	读/写	描述	复位值
P0T_RFC	0x7E00002C	读/写	16 位 DRAM 控制 t_ RFC 寄存器	0x212
P1T_RFC	0x7E00102C	读/写	32 位 DRAM 控制 t_ RFC 寄存器	0x212

PnT_ RFC	位	描述	初始状态
	[31:10]	读未定义，写为 0	
scheduled_ RFC	[9:5]	在 ACLK 周期，设置自动刷新指令时间	0x10
t_ RFC	[4:0]	内存时钟周期内，设置自动刷新指令时间	0x12

5.4.13 T\_RP 寄存器

寄存器	地址	读/写	描述	复位值
P0T_RP	0x7E000030	读/写	16 位 DRAM 控制 t_ RP 寄存器	0x1D
P1T_ RP	0x7E001030	读/写	32 位 DRAM 控制 t_ RP 寄存器	0x1D

PnT_ RP	位	描述	初始状态
	[31:6]	读未定义，写为 0	
scheduled_ RP	[5:3]	在 ACLK 周期内，设置预充电到 RAS 延迟	011
t_ RP	[2:0]	内存时钟周期内，设置预充电到 RAS 延迟	101

5.4.14 T\_RRD 寄存器

寄存器	地址	读/写	描述	复位值
P0T_RRD	0x7E000034	读/写	16 位 DRAM 控制 t_ RRD 寄存器	0x2
P1T_ RRD	0x7E001034	读/写	32 位 DRAM 控制 t_ RRD 寄存器	0x2

PnT_ RRD	位	描述	初始状态
	[31:4]	读未定义，写为 0	
t_ RRD	[3:0]	内存时钟周期内，设置有效页 x 到有效页 y 延迟	0x2

5.4.15 T\_WR 寄存器

寄存器	地址	读/写	描述	复位值
P0T_WR	0x7E000038	读/写	16 位 DRAM 控制 t_ WR 寄存器	0x3
P1T_ WR	0x7E001038	读/写	32 位 DRAM 控制 t_ WR 寄存器	0x3

PnT_ WR	位	描述	初始状态
	[31:3]	读未定义，写为 0	
t_ WR	[2:0]	内存时钟周期内，设置写到预充电延迟	011

### 5.4.16 T\_WTR 寄存器

寄存器	地址	读/写	描述	复位值
P0T_ WTR	0x7E00003C	读/写	16 位 DRAM 控制 t_ WTR 寄存器	0x2
P1T_ WTR	0x7E00103C	读/写	32 位 DRAM 控制 t_ WTR 寄存器	0x2

PnT_ WTR	位	描述	初始状态
	[31:3]	读未定义，写为 0	
t_ WTR	[2:0]	内存时钟周期内，设置写到读延迟	010

### 5.4.17 T\_XP 寄存器

寄存器	地址	读/写	描述	复位值
P0T_ XP	0x7E000040	读/写	16 位 DRAM 控制 t_ XP 寄存器	0x01
P1T_ XP	0x7E001040	读/写	32 位 DRAM 控制 t_ XP 寄存器	0x01

PnT_ XP	位	描述	初始状态
	[31:8]	读未定义，写为 0	
t_ XP	[7:0]	内存时钟周期内，设置退出掉电命令时间	0x01



5.4.18 T\_XSR 寄存器

寄存器	地址	读/写	描述	复位值
P0T_XSR	0x7E000044	读/写	16 位 DRAM 控制 t_ XSR 寄存器	0x0A
P1T_ XSR	0x7E001044	读/写	32 位 DRAM 控制 t_ XSR 寄存器	0x0A

PnT_ XSR	位	描述	初始状态
	[31:8]	读未定义，写为 0	
t_ XSR	[7:0]	内存时钟周期内，设置退出自刷新命令时间	0x0A

5.4.19 T\_ESR 寄存器

寄存器	地址	读/写	描述	复位值
P0T_ESR	0x7E000048	读/写	16 位 DRAM 控制 t_ ESR 寄存器	0x14
P1T_ ESR	0x7E001048	读/写	32 位 DRAM 控制 t_ ESR 寄存器	0x14

PnT_ ESR	位	描述	初始状态
	[31:8]	读未定义，写为 0	
t_ ESR	[7:0]	内存时钟周期内，设置自刷新命令时间	0x14

5.4.20 存储配置 2 寄存器

寄存器	地址	读/写	描述	复位值
POMEMCFG2	0x7E00004C	读/写	16 位 DRAM 控制配置寄存器	0x0B00
P1MEMCFG2	0x7E00104C	读/写	32 位 DRAM 控制配置寄存器	0x0B40

PnMEMCFG2	位	描述	初始状态
Reserved	[31:13]	读未定义，写为 0。	
Read delay	[12:11]	<p>当从信息包接口到允许用于传入读数据的低歪曲率时，使用编码延迟。</p> <p>00=读延迟 0 周期（通常用于 SDR SDRAM）。</p> <p>01=读延迟 1 周期（通常用于 DDR SDRAM 和移动 DDR SDRAM）</p> <p>10, 11=读延迟 2 周期。</p>	01
Memory type	[10:8]	<p>附加上 DRAM 控制器类型的 SDRAM：</p> <p>000=SDR SDRAM</p> <p>001=DDR SDRAM</p> <p>011=移动的 DDR SDRAM</p> <p>010=嵌入式的 SDRAM</p> <p>1xx=保留</p>	011
Memory width	[7:6]	<p>外部存储器的宽度：</p> <p>00=16 位    01=32 位    10=64 位    11=128 位</p>	00 / 01
Bank bits	[5:4]	<p>AXI 地址位的编码数包含的页地址：</p> <p>00=2 位</p> <p>01=1 位</p> <p>10=0 位</p> <p>11=保留</p>	00
Reserved	[3]	读为 0，写为 0。	0
DQM init	[2]	当存储器重置时，为 DQM 状态	0
Clock config	[1:0]	<p>时钟顺序支持：</p> <p>00=AXI 时钟和内存时钟异步</p> <p>01=AXI 时钟和内存时钟同步，和 AXI 时钟可以是相同的频率，或是慢于内存时钟</p> <p>S3C6410 支持同步配置。如果这个值被设置为异步，则 S3C6410 性能将持续降低</p> <p>10~11=保留</p>	00

### 5.4.21 ID\_N\_CFG 寄存器

寄存器	地址	读/写	描述	复位值
P0_id_0_cfg ~P0_id_15_cfg	0x7E000100 ~0x7E00013C	读/写	16 位 DRAM 控制 id_<n>_cfg 寄存器	0x000
P1_id_0_cfg ~P1_id_15_cfg	0x7E001100 ~0x7E00113C	读/写	32 位 DRAM 控制 id_<n>_cfg 寄存器	0x000

Pn_id_<n>_cfg	位	描述	初始状态
	[31:10]	读未定义，写为 0	
QoS_MAX	[9:2]	设置一个最高质量的服务	0x00
QoS_MIN	[1]	设置一个最低质量的服务	0
QoS_Enable	[0]	使能一个服务质量的值以适用于存储器从地址 ID<n>读	0

### 5.4.22 CHIP\_N\_CFG 寄存器

寄存器	地址	读/写	描述	复位值
P0_chip_0_cfg P0_chip_1_cfg	0x7E000200 0x7E000204	读/写	16 位 DRAM 控制 chip_<n>_cfg 寄存器	0x0FF00
P1_chip_0_cfg P1_chip_1_cfg	0x7E001200 0x7E001204	读/写	32 位 DRAM 控制 chip_<n>_cfg 寄存器	0x0FF00

Pn_chip_<n>_cfg	位	描述	初始状态
	[31:17]	读未定义，写为 0	
BRC_RBC	[16]	选择存储器结构作为从 AXI 的解码地址： 0=行-页-列结构 1=页-行-列结构	0

Address match	[15:8]	比较 AXI 地址位[31:24]的值，决定选择哪一个芯片	0xFF
Address mask	[7:0]	屏蔽 AXI 的地址位[31:24]，决定选择哪一个芯片 1=用于比较的相应地址位	0x00

### 5.4.23 用户身份寄存器

寄存器	地址	读/写	描述	复位值
P0_user_stat	0x7E000300	读	16 位 DRAM 控制用户身份寄存器	0x00
P1_user_stat	0x7E001300	读	32 位 DRAM 控制用户身份寄存器	0x00

Pn_user_stat	位	描述	初始状态
	[31:8]	读未定义，写为 0	
DQS[3] delay	[7:6]	显示输入 dqs[3]延迟	0
DQS[2] delay	[5:4]	显示输入 dqs[2]延迟	0
DQS[1] delay	[3:2]	显示输入 dqs[1]延迟	0
DQS[0] delay	[1:0]	显示输入 dqs[0]延迟	0

### 5.4.24 用户配置寄存器

寄存器	地址	读/写	描述	复位值
P0_user_cfg	0x7E000304	写	16 位 DRAM 控制用户配置寄存器。	0x00
P1_user_cfg	0x7E001304	写	32 位 DRAM 控制用户配置寄存器。	0x00

Pn_user_ cfg	位	描述	初始状态
	[31:8]	读未定义，写为 0	
DQS[3] delay	[7:6]	显示输入 dqs[3]延迟	0

DQS[2] delay	[5:4]	显示输入 dqs[2] 延迟	0
DQS[1] delay	[3:2]	显示输入 dqs[1] 延迟	0
DQS[0] delay	[1:0]	显示输入 dqs[0] 延迟	0

## 5.5 DRAM 控制器 API 功能的软件实现举例

下面的程序主要是初始化一个特定 DMC 控制器：DMC0 => X16, mDDR (512Mb), DMC1=> X16\*2, mDDR (1025Mb)。该部分程序中包含了一些寄存器的应用，通过分析程序可以对这些寄存器有更深入的认识。

```
void DMC_Init(u8 uController, DMC_eStopCLK eStopClk, DMC_eAutoPD eAutoPD, u32 uPDprd )
```

```
{
    u32 uBaseAddress;
    u32 uPara_REF,uPara_RAS, uPara_RC, uPara_RCD, uScheduled_Para;
    u32 uPara_RFC, uPara_RP, uPara_RRD, uPara_WR, uPara_WTR, uPara_XSR;
// uController — DMC 控制器号
    if(uController == 0)
    {
        uBaseAddress = DMC0_BASE;
    }
    else if(uController == 1)
    {
        uBaseAddress = DMC1_BASE;
    }
    else
    {
    }

    g_DMCCBase[uController] = (void *)uBaseAddress;

    // 获得系统时钟信息  g_HCLK
```

```
SYSC_GetClkInform();
```

```
//输入配置模式
```

```
Outp32(&DMC(uController)->rMEMCCMD, (0x4<<0)); // Memc_cmd = 0x4
```

```
// 定时参数设置
```

```
uPara_REF = (((g_HCLK / 1000 * eDDR_REF) - 1) / 1000000 + 1);
```

```
Outp32(&DMC(uController)->rREFRESH, uPara_REF);
```

```
Outp32(&DMC(uController)->rCASLAT, (0x3<<1)); // CAS 延迟 = 3
```

```
Outp32(&DMC(uController)->rT_DQSS, 0x1); // mDDR = 0x1 (0.75 ~ 1.25)
```

```
Outp32(&DMC(uController)->rT_MRD, 0x2); // 模式定时器 Cmd 定时
```

```
uPara_RAS = (((g_HCLK / 1000 * eDDR_RAS) - 1) / 1000000 + 1);
```

```
Outp32(&DMC(uController)->rT_RAS, uPara_RAS); // RAS(行激活时间)预加点延迟, (最小:45ns,  
7@133MHz, 极限:1 clock)
```

```
uPara_RC = (((g_HCLK / 1000 * eDDR_RC) - 1) / 1000000 + 1);
```

```
Outp32(&DMC(uController)->rT_RC, uPara_RC); // RC(行周期时间): 激活 bank x 到激活 bank  
x 延迟 (最小:67.5ns, 10@133MHz, 极限:1clock)
```

```
uPara_RCD = (((g_HCLK / 1000 * eDDR_RCD) - 1) / 1000000 + 1);
```

```
if (uPara_RCD < 4)
```

```
{
```

```
    uScheduled_Para = 3;
```

```
}
```

```
else
```

```
{
```

```
    uScheduled_Para = uPara_RCD;
```

```
}
```

```
uPara_RCD = ((uScheduled_Para - 3) << 3) | (uPara_RCD);
```

```
Outp32(&DMC(uController)->rT_RCD, uPara_RCD); // RAS 到 CAS 延迟, (最小:22.5ns, 4@133MHz,
```

极限:1clock)

```
uPara_RFC = (((g_HCLK / 1000 * eDDR_RFC) - 1) / 1000000 + 1);
```

```
if (uPara_RFC < 4)
```

```
{
```

```
    uScheduled_Para = 3;
```

```
}
```

```
else
```

```
{
```

```
    uScheduled_Para = uPara_RFC;
```

```
}
```

```
uPara_RFC = ((uScheduled_Para - 3) << 3) | (uPara_RFC);    // 自动刷新到 cmd 时间(>=t_RC), (最
```

小:80ns, 11@133MHz)

```
Outp32(&DMC(uController)->rT_RFC, uPara_RFC);
```

```
uPara_RP = (((g_HCLK / 1000 * eDDR_RP) - 1) / 1000000 + 1);
```

```
if (uPara_RP < 4)
```

```
{
```

```
    uScheduled_Para = 3;
```

```
}
```

```
else
```

```
{
```

```
    uScheduled_Para = uPara_RP;
```

```
}
```

```
uPara_RP = ((uScheduled_Para - 3) << 3) | (uPara_RP);
```

```
Outp32(&DMC(uController)->rT_RP, uPara_RP);    // 预加电 RAS 延迟(行预加电时间) (最
```

小:22.5ns, 4@133MHz, 极限:1clock)

```
uPara_RRD = (((g_HCLK / 1000 * eDDR_RRD) - 1) / 1000000 + 1);
```

```
Outp32(&DMC(uController)->rT_RRD, uPara_RRD);    // 激活 bank x 到激活 bank y 延迟(最小:15ns,
```

3@133MHz, 极限:1clock)

```
uPara_WR = (((g_HCLK / 1000 * eDDR_WR) - 1) / 1000000 + 1);
```

```

    Outp32(&DMC(uController)->rT_WR, uPara_WR);    // 写入预加电延迟(最小:15ns, 3@133MHz,极限:1clock)
//  uPara_WTR= (((g_HCLK /1000 *eDDR_RP) -1)/1000000 + 1) + 2;
    uPara_WTR = 2;
    Outp32(&DMC(uController)->rT_WTR, uPara_WTR);    // 写到读延迟
    Outp32(&DMC(uController)->rT_XP, 2);            // 脱离掉电 cmd 时间
    uPara_XSR= (((g_HCLK /1000 *eDDR_XSR) -1)/1000000 + 1) ;
    Outp32(&DMC(uController)->rT_XSR,uPara_XSR);    // 脱离自刷新 cmd 时间
    Outp32(&DMC(uController)->rT_ESR,uPara_XSR);    // 自刷新 cmd 时间

    if(uController == 0)
    {
        // 存储配置寄存器
        Outp32(&DMC(uController)->rMEMCFG,
            (0<<21) |    // 1 片
            (0<<18) |    // 通过 ARID[3:0]进行 Qos 主控器选择
            (2<<15) |    // 突发脉冲 4
            (eStopClk<<14) |    // 停止存储器时钟无效
            (eAutoPD<<13) |    // 自动掉电无效
            (uPDprd<<7) |    // 自动掉电周期
            (eAP_bit<<6) |    // 位 10 自动加电位
            (eRow_bit<<3) |    // 13 位列位
            (eCol_bit<<0) );    // 10 位列位

        // 存储配置寄存器 2
        Outp32(&DMC(uController)->rMEMCFG2,
            (1<<11) |    // Read delay 1 cycle from the Pad I/F ( 0x1 => mDDR)
            (3<<8) |    // Memory Type (mDDR)
            (0<<6) |    // Width ( 16bit)

```



```

        (0<<4) |          // Bank bits = 2bit
        (0<<2) |          // DQM state
        (1<<0) );        // Sync. clock scheme

// CHIP0 配置
    Outp32(&DMC(uController)->rCHIP_0_CFG, 0x140fc);
0x4000_0000 ~ 0x43ff_ffff ( 64MB)

}
else if(uController == 1)
{
    // 存储配置寄存器
    Outp32(&DMC(uController)->rMEMCFG,
        (0<<31) |      // 无效个别 CKE 控制
        (0<<21) |      // 1 片
        (0<<18) |      // ARID[3:0]的 Qos 主控器选择
        (2<<15) |      // 脉冲 4
        (eStopClk<<14) |      // 无效存储器时钟( 0)
        (eAutoPD<<13) |      // 无效自动掉电(0)
        (uPDprd<<7)  |      // 自动掉电周期
        (eAP_bit<<6) |      // 位 10 的自动预加电位
        (eRow_bit<<3)|      // 13 位行位
        (eCol_bit<<0) );    // 10 位列位

    // 存储配置寄存器 2
    Outp32(&DMC(uController)->rMEMCFG2,
        (1<<11) |          // 读取从 Pad I/F ( 0x1 => mDDR)延迟一个周期
        (3<<8) |           // 存储器类型(mDDR)
        (1<<6) |           // 宽度 ( 32 位)

```

```

(0<<4) |           // Bank 位 = 2 位
(0<<2) |           // DQM 状态
(1<<0) );          // 同步时钟配置

```

// CHIP0 配置

```

Outp32(&DMC(uController)->rCHIP_0_CFG, 0x150f8);
0x5000_0000 ~ 0x57ff_ffff ( 128MB)
}

```

// 外部存储器初始化

// NOP

```

Outp32(&DMC(uController)->rDIRECTCMD,
(0<<20) |    //片地址 - 片 0
(3<<18) );    //指令- NOP

```

// 预加电

```

Outp32(&DMC(uController)->rDIRECTCMD,
(0<<20) |    // 片地址 - Chip 0
(0<<18) );    // 指令- PALL

```

// 自动刷新两次

```

Outp32(&DMC(uController)->rDIRECTCMD,
(0<<20) |    //片地址 - 片 0
(1<<18) );    // 指令 - 自动刷新

```

```

Outp32(&DMC(uController)->rDIRECTCMD,
(0<<20) |    //片地址 - 片 0
(1<<18) );    // 指令- 自动刷新

```

```
// MRS
```

```
Outp32(&DMC(uController)->rDIRECTCMD,
```

```
    (0<<20) |    // 片地址 - 片 0
```

```
    (2<<18) |    // 指令- MRS/EMRS
```

```
    (0<<16) |    // [17:16] - MRS ( 0 )
```

```
    (0x32<<0));    // [6:4]- CAS 等待 - 3,
```

```
                // [3] -脉冲类型 (连续), [2:0] - 脉冲长度(4) -> 8
```

```
// EMRS
```

```
Outp32(&DMC(uController)->rDIRECTCMD,
```

```
    (0<<20) |    // 片地址 - 片 0
```

```
    (2<<18) |    // 指令- MRS/EMRS
```

```
    (2<<16) |    // [17:16] - EMRS (2)
```

```
    (0x0<<0));
```

```
//'GO' 模式
```

```
Outp32(&DMC(uController)->rMEMCCMD, (0x0<<0)); // Go 指令
```

```
// 检查控制器状态
```

```
while((Inp32(&DMC(uController)->rMEMSTAT))&0x3 == 1 );
```

```
}
```

## 6 SROM 控制器

本小节主要介绍 SROM 控制器的功能及使用。S3C6410 SROM 控制器(SROMC)支持外部 8,16 位 NOR Flash, PROM, SRAM 存储器。

### 6.1 SROM 控制器的特性

S3C6410 SROM 控制器的特性包括：

- (1) 支持 SRAM，不同的 ROM 和 NOR flash 存储器。
- (2) 支持仅 8 或 16 位数据总线。
- (3) 地址空间：每页高达 128MB。
- (4) 支持 6 页。
- (5) 固定内存页开始地址。
- (6) 外部等待扩展总线周期。
- (7) 支持字节和半字存取的外部存储器。

如图 6-1 所示，显示了 SROM 控制器的结构框图。

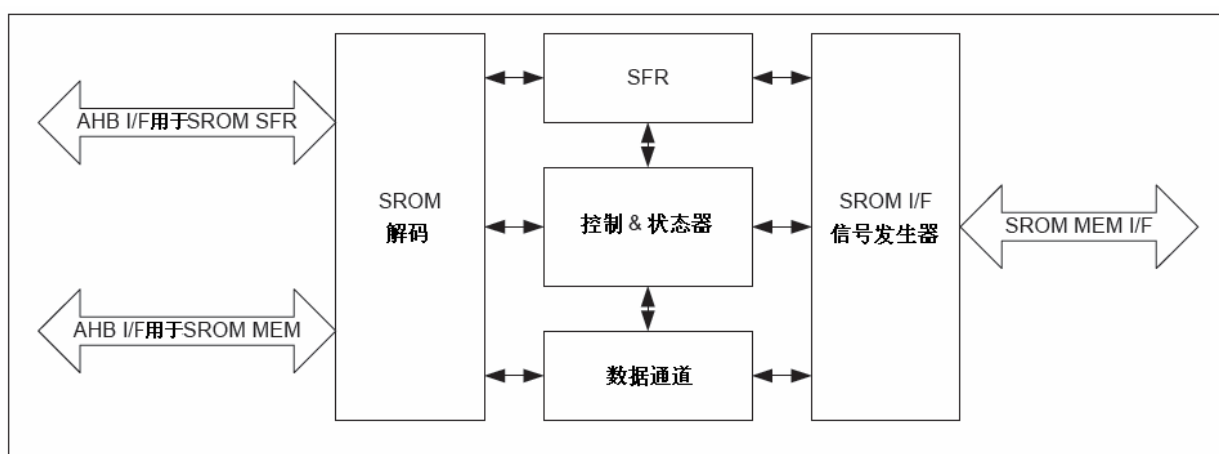


图 6-1 SROM 控制器框图

## 6.2 SROM 控制器功能描述

SROM 控制器支持从 Bank0~Bank5 的 SROM 接口。在 OneNAND 启动情况中，SROM 控制器不能控制 Bank2 和 Bank3，因为它的主要身份是在 OneNAND 控制器上。在 NAND 引号例子中，SROM 控制器同样也不能控制 Bank2 和 Bank3，因为它的控制是在 NAND Flash 控制器上。

### 6.2.1.nWAIT 引脚操作

如果 WAIT 操作相应的每个存储块都被使能，nOE 持续时间将通过外部 nWAIT 引脚被延长，存储块被激活。从  $t_{acc}-1$  中检测 nWAIT。在取样 nWAIT 为高后，下一个时钟 nOE 将低有效。new 信号和 nOE 信号有同样的关系。SROM 控制器 nWAIT 的时序框图，如图 6-2 所示。

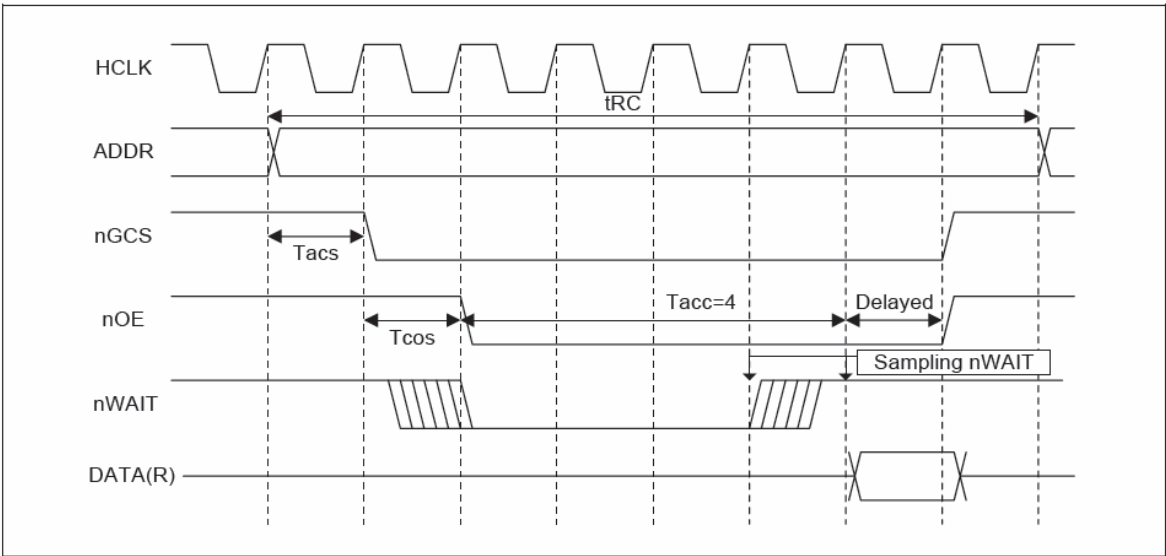


图 6-2 SROM 控制器 nWAIT 时序框图

### 6.2.2.可编程访问周期

如图 6-3 所示，描述的是 SR0M 控制器读时序的周期框图，如图 6-4 所示，描述的是 SR0M 控制器写时序的周期框图。

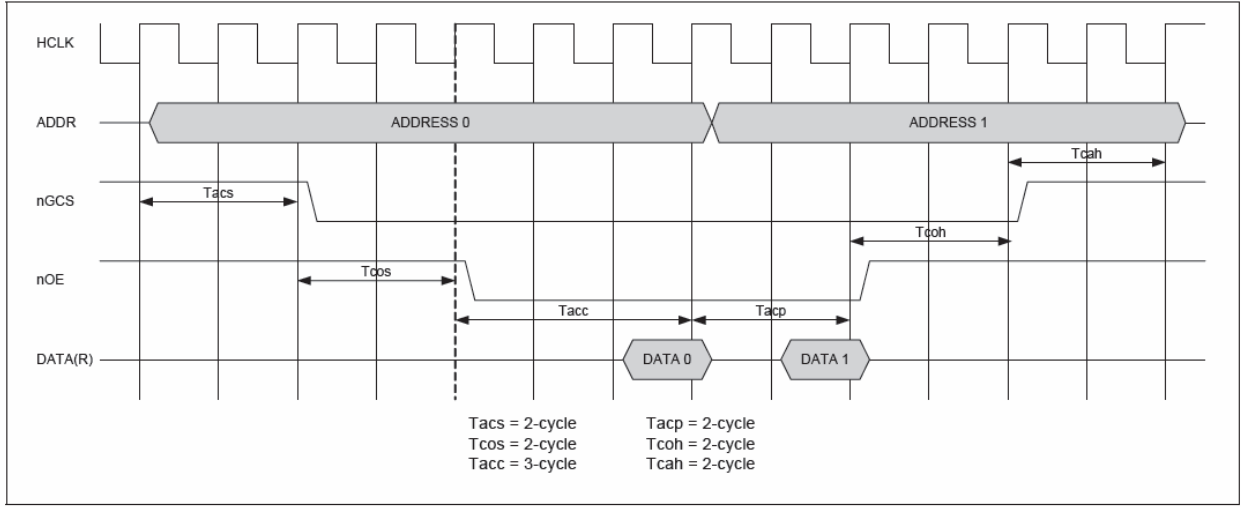


图 6-3 SR0M 控制器读时序框图

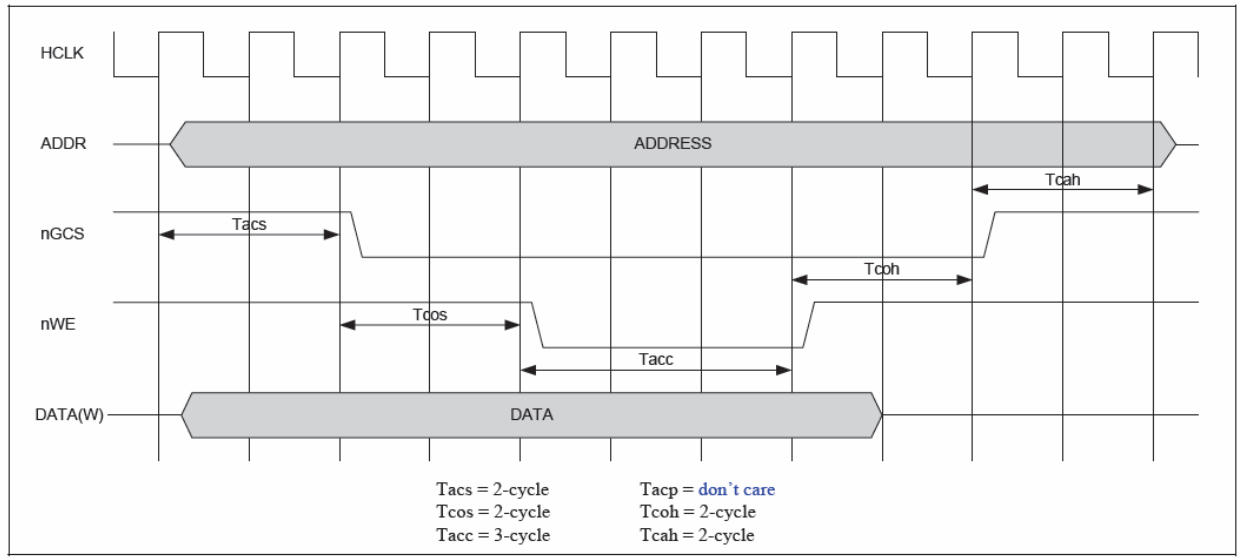


图 6-4 SR0M 控制器写时序框图

注：页模式仅支持读周期。

### 6.2.3.SROMC 块设置代码实现

以下是 S3C6410 SROM 控制器（SROMC）在 ARM11 中的代码定义，参照如下：

// SROMC\_SetBank 函数：主要功能初始化 SROMC 块

//输入：uBank：选择块数

// eByteCTL：选择块 UL/BL 控制

// eWAITCTL：选择块 WAIT 控制

// eDWidth：选择块数据宽度控制

// ePage：选择块页模式控制

// eTacs：选择块 Tacs 控制

// eTcos：选择块 Tcos 控制

// eTacc：选择块 Tacc 控制

// eTcoh：选择块 Tcoh 控制

// eTcah：选择块 Tcah 控制

// eTACP：选择块 TACP 控制

// 输出：

```
void SROMC_SetBank(u8 uBank, Byte_eCTL eByteCTL, WAIT_eCTL eWAITCTL, Data_eWidth eDWidth,
                  Page_eMode ePage, Bank_eTiming eTacs, Bank_eTiming eTcos, Bank_eTiming eTacc,
                  Bank_eTiming eTcoh, Bank_eTiming eTcah, Bank_eTiming eTACP)
```

```
{
```

```
    u32 uBaseAddress = 0;
```

```
    u32 uConValue = 0;
```

```
    volatile u32 *pSROMC_BC_Addr = NULL;
```

```
    volatile u32 *pSROMC_BW_Addr = NULL;
```

```
    uBaseAddress = SROMC_BASE;
```

```
    g_SROMCBase = (void *)uBaseAddress;
```

```
// 总线宽度& 等待控制
pSROMC_BW_Addr = &(SROMC->rSROM_BW);
uConValue = Inp32(&SROMC->rSROM_BW);
uConValue = (uConValue & ~(0xF<<(uBank*4))) | (eDWidth<<(uBank*4))|
              (eWAITCTL<<(uBank*4 + 2))|(eByteCTL<<(uBank*4+3));
*pSROMC_BW_Addr = uConValue;

//块控制寄存器
pSROMC_BC_Addr = &(SROMC->rSROM_BC0);
pSROMC_BC_Addr = pSROMC_BC_Addr + uBank;
uConValue = (eTacs<<28)|(eTcos<<24)|(eTacc<<16)|(eTcoh<<12)|(eTcah<<8)|(eTacr<<4)
              |(ePage<<0);
*pSROMC_BC_Addr = uConValue;
}
```

6.3 特殊功能寄存器描述

6.3.1. SROM 总线宽度&等待控制寄存器(SROM\_BW)

寄存器	地址	读/写	描述	复位值
SROM_BW	0x70000000	读/写	SROM 总线宽度和等待控制	0x0000_000x

SROM_BW	位	描述	初始状态
Reserved	[31:24]	保留	0
ByteEnable5	[23]	nWBE / nBE (用于UB/LB) 控制，用于存储器Bank5 0 = 不使用UB/LB (XrnWBE[1:0] 专门的nWBE[1:0]) 1 = 使用 UB/LB (XrnWBE[1:0] 专门的 nBE[1:0])	0



WaitEnable5	[22]	等待使能控制，用于存储器 Bank5 0=WAIT 禁止          1=WAIT 使能	0
Reserved	[21]	保留。	0
DataWidth5	[20]	数据总线宽度控制，用于存储器 Bank5 0=8 位                  1=16 位	0
ByteEnable4	[19]	nWBE / nBE (用于UB/LB) 控制，用于存储器Bank4 0 = 不使用UB/LB (XrnWBE[1:0]专门的nWBE[1:0]) 1 =使用 UB/LB (XrnWBE[1:0] 专门的 nBE[1:0])	0
WaitEnable4	[18]	等待使能控制，用于存储器Bank4 0=WAIT 禁止          1=WAIT 使能	0
Reserved	[17]	保留。	0
DataWidth4	[16]	数据总线宽度控制，用于存储器 Bank4 0=8 位                  1=16 位	0
ByteEnable3	[15]	nWBE / nBE (用于UB/LB) 控制，用于存储器Bank3 0 = 不使用UB/LB (XrnWBE[1:0]专门的nWBE[1:0]) 1 =使用 UB/LB (XrnWBE[1:0]专门的 nBE[1:0])	0
WaitEnable3	[14]	等待使能控制，用于存储器Bank3 0=WAIT 禁止          1=WAIT 使能	0
Reserved	[13]	保留	0
DataWidth3	[12]	数据总线宽度控制，用于存储器 Bank3 0=8 位                  1=16 位	0
ByteEnable2	[11]	nWBE / nBE (用于UB/LB) 控制，用于存储器Bank2。 0 = 不使用UB/LB (XrnWBE[1:0]专门的nWBE[1:0]) 1 =使用 UB/LB (XrnWBE[1:0]专门的 nBE[1:0])	0
WaitEnable2	[10]	等待使能控制，用于存储器Bank2 0=WAIT 禁止          1=WAIT 使能	0
Reserved	[9]	保留	0
DataWidth2	[8]	数据总线宽度控制，用于存储器 Bank2 0=8 位                  1=16 位	0

ByteEnable1	[7]	nWBE / nBE(用于UB/LB) 控制，用于存储器Bank1。 0 = 不使用UB/LB (XrnWBE[1:0]专门的nWBE[1:0]) 1 =使用 UB/LB (XrnWBE[1:0]专门的 nBE[1:0])	0
WaitEnable1	[6]	等待使能控制，用于存储器Bank1 0=WAIT 禁止            1=WAIT 使能	0
Reserved	[5]	保留	0
DataWidth1	[4]	数据总线宽度控制，用于存储器 Bank1 0=8 位                    1=16 位	0
ByteEnable0	[3]	nWBE / nBE(用于UB/LB) 控制，用于存储器Bank0 0 = 不使用UB/LB (XrnWBE[1:0]专门的nWBE[1:0]) 1 =使用 UB/LB (XrnWBE[1:0]专门的 nBE[1:0])	0
WaitEnable0	[2]	等待使能控制，用于存储器Bank0 0=WAIT 禁止            1=WAIT 使能	0
Reserved	[1]	保留	0
DataWidth0	[0]	数据总线宽度控制，用于存储器 Bank0 0=8 位                    1=16 位	H/W Set

### 6.3.2. SRAM 页控制寄存器(SRAM\_BC:XrCSn0~XrCSn2)

寄存器	地址	读/写	描述	复位值
SRAM_BC0	0x70000004	读/写	SRAM Bank0 控制寄存器	0x000F_0000
SRAM_BC1	0x70000008	读/写	SRAM Bank1 控制寄存器	0x000F_0000
SRAM_BC2	0x7000000C	读/写	SRAM Bank2 控制寄存器	0x000F_0000
SRAM_BC3	0x70000010	读/写	SRAM Bank3 控制寄存器	0x000F_0000
SRAM_BC4	0x70000014	读/写	SRAM Bank4 控制寄存器	0x000F_0000
SRAM_BC5	0x70000018	读/写	SRAM Bank5 控制寄存器	0x000F_0000

SR0M_BcN	位	描述	初始状态
Tacs	[31:28]	在 nGCS 之前，准备地址 0000=0 时钟                      0001=1 时钟 0010=2 时钟                      0011=3 时钟 ..... 1100=12 时钟                      1101=13 时钟 1110=14 时钟                      1111=15 时钟	0000
Tcos	[27:24]	在 nOE 之前，芯片准备选择 0000=0 时钟                      0001=1 时钟 0010=2 时钟                      0011=3 时钟 ..... 1100=12 时钟                      1101=13 时钟 1110=14 时钟                      1111=15 时钟	0000
Reserved	[23:21]	保留	000
Tacc	[20:16]	访问周期 00000=1 时钟                      00001=2 时钟 00010=3 时钟                      00011=4 时钟 ..... 11100=29 时钟                      11101=30 时钟 11110=31 时钟                      11111=32 时钟	0111
Tcoh	[15:12]	在 nOE 上芯片选择保持 0000=0 时钟                      0001=1 时钟 0010=2 时钟                      0011=3 时钟 ..... 1100=12 时钟                      1101=13 时钟 1110=14 时钟                      1111=15 时钟	0000
Tcah	[11:8]	nGCSn 之后，地址占用时间 0000=0 时钟                      0001=1 时钟 0010=2 时钟                      0011=3 时钟	0000

		<p>.....</p> <p>1100=12 时钟                      1101=13 时钟</p> <p>1110=14 时钟                      1111=15 时钟</p>	
Tacp	[7:4]	<p>在页模式下，页模式访问周期</p> <p>0000=0 时钟                      0001=1 时钟</p> <p>0010=2 时钟                      0011=3 时钟</p> <p>.....</p> <p>1100=12 时钟                      1101=13 时钟</p> <p>1110=14 时钟                      1111=15 时钟</p>	0000
Reserved	[3:2]	保留	
PMC	[1:0]	<p>页模式配置</p> <p>00=标准的（1 位数据）              01=4 位数据</p> <p>10=8 位数据                      11=16 位数据</p>	00

## 7 ONENAND 控制器

本节主要介绍 S3C6410 RSIC 微处理器 OneNAND 控制器的功能和使用。S3C6410 支持外部 16 位总线，用于同步和异步 OneNAND 外部存储（通过平分端口 0）。通过使用两个控制器，最大可支持 2 页。Denali OneNAND Flash 存储控制器被用做 S3C6410 的 OneNAND 控制器使用。OneNAND 控制器是由 Denali 开发、测试和许可的，具有先进的微控制器总线架构 (AMBA 2)，可兼容的系统单晶片外部设备。该控制器同时支持两组存储器。每个存储页仅支持多路复用的 OneNAND。使用 OneNAND Flash 代替 NAND Flash，其 ‘XSELNAND’ 引脚必须置 0（低电平）。

### 7.1 ONENAND 控制器的特性

OneNAND 控制器的特性包括以下几个方面：

- (1) 通过使用两个 OneNAND 控制器，支持最大 2 页。
- (2) 支持同步/异步多路复用的 OneNAND 存储器。
- (3) 支持 16 位宽的外部存储器数据通道。
- (4) 支持 SINGLE/INCR4/INCR8 脉冲传输，用于 32 位 AHB 数据总线。
- (5) 支持单一字传输，用于 32 位 AHB SFR 总线。
- (6) 支持仅 ERROR/OKAY 响应，用于两个 AHB 总线。
- (7) 数据缓冲以达到最高性能。
- (8) 在 Flash 控制器和系统总线接口间异步的先进先出，用于速度的匹配。
- (9) 通过地址映射，支持擦除命令。
- (10) 支持复制模式作为寄存器命令。
- (11) 如果 OneNAND 设备 ID 是 0x0040、0x0048 和 0x0058，支持写同步模式。
- (12) 如果 OneNAND 设备 ID 是 0x0030, 0x0034 及 OneNAND 版本 ID 位[9:8] 不是 2' b00，支持写同步模式。
- (13) 当映射 01 页访问命令被使用时，支持 LDM4/STM4。如果设备密度是 128Mb 或 256Mb，推荐不超过一个字的访问，用于 01 页访问命令。

1.ONENAND 控制器框图

如图 7-1 所示，显示了 OneNAND 控制器的结构框图。

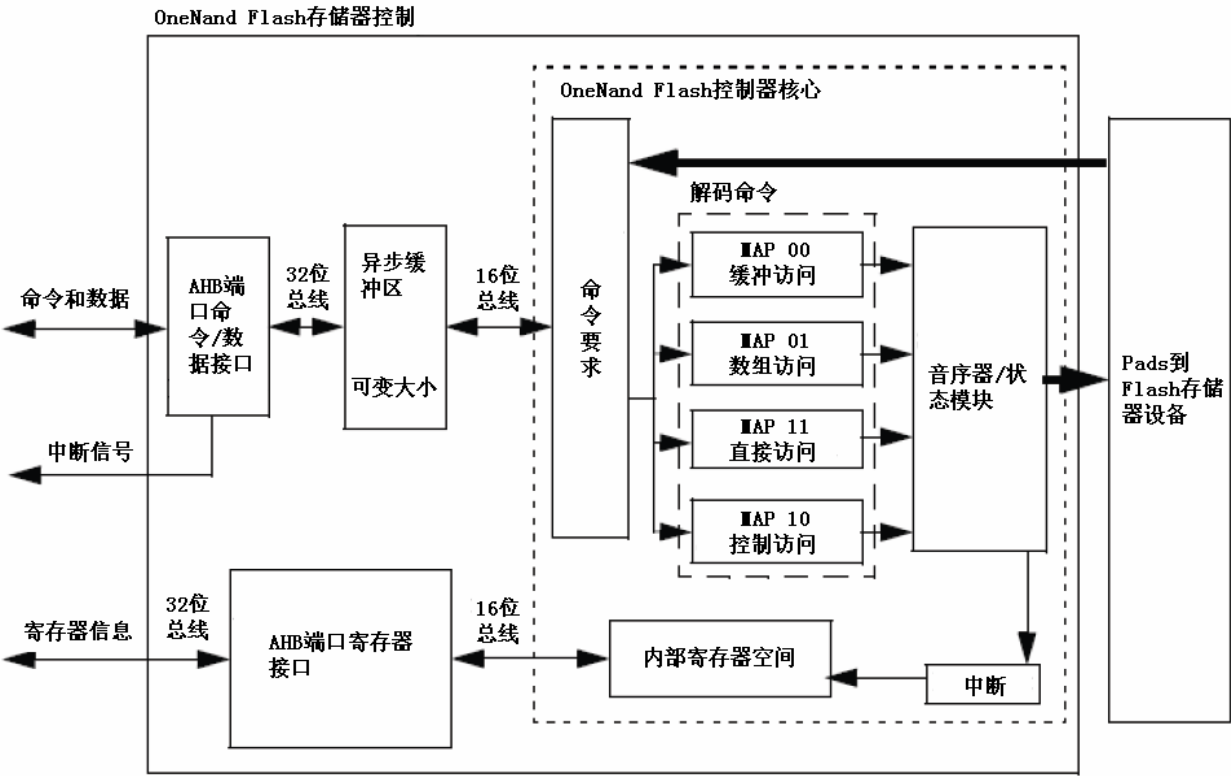


图 7-1 OneNAND 控制器结构框图

2. 信号描述

外部存储器接口的具体信号描述，如表 7-1 所示。

表 7-1 外部存储器接口信号描述

信号	I/O	描述
Xm0DATA [15:0]	IO	在内存读/写地址段期间，Xm0DATA[15:0]（数据总线）输出地址。内存读数据段时输入数据，内存写数据段时输出数据
Xm0CSn[1:0]	0	Xm0CSn[1:0]（可选芯片）被激活时，内存地址不超出每个页地址区域。通过系统控制 SFR 的设置，Xm0CSn[1:0]可以被分配到 SROMC 或 OneNAND 控制器。 低有效
Xm0WEn	0	Xm0WEn（写使能）说明当前的总线周期是写周期 低有效

Xm00En	0	Xm00en（输出使能）说明当前的总线周期是读周期 低有效
Xm0INTsm0_FWEn Xm0INTsm1_FREn	I	从 OneNAND 存储页 0, 1 中断输入 如果 OneNAND 存储器没有被使用，这些信号必须置 0
Xm0ADDRVALID	0	地址有效输出。在 POP 产品中，地址和数据复用。Xm0ADDRVALID 说明总线被使用时， 用于地址 低有效
XmORPn	0	系统复位输出，OneNAND 存储器 低有效
XmORDY0_ALE XmORDY1_CLE	I	XmORDY 是同步脉冲等待输入，外部设备使用延迟同步脉冲转移。XmORDY 在同步读模 式下，说明数据有效。当 XmOCSn 为低时，XmORDY 被激活
XmOSMCLK	0	静态存储器时钟，用于同步静态存储器设备

### 3. 输入时钟

OneNAND 控制器有三个时钟源输入。总线系统接口获得 AHB 总线时钟，即 HCLK。Flash 控制器核心获得两个 Flash 时钟，即 mclk 和 mclk\_flash。mclk 的频率必须是 mclk\_flash 的两倍，由 OneNAND flash 存储器提供。

可以设置系统控制器 SFR 的频率。当更改时钟频率的比率时，必须按照下面的程序：

- （1）确保没有存储器传输。
- （2）在系统控制器 SFR 内，转换时钟比率。
- （3）写时钟比率寄存器。
- （4）开始存储器访问。

## 7.2 存储器地址映射

OneNAND 控制器读内存设备 dev\_id 的大小区域，以决定地址映射。自动配置 MEM\_ADDR 区域的地址映射到支持的设备。如表 7-2 所示，“MEM\_ADDR 区域”决定区域的大小，用于几个 OneNAND 存储器设备。

表 7-2 MEM\_ADDR 区域

dev_id 区 域 大小	密度	所属 块	页 大 小	映射 位置	MEM_ADDR 区域					
					保留	DFS_DBS	FBA	FPA	FSA	保留
0000	128Mb	256	1KB	[23:22]	[21:17]	N/A	[16:9]	[8:3]	[2]	[1:0]
0001	256Mb	512	1KB	[23:22]	[21:18]	N/A	[17:9]	[8:3]	[2]	[1:0]
0010	512Mb	512	2KB	[23:22]	[21:19]	N/A	[18:10]	[9:4]	[3:2]	[1:0]
0011	1Gb	1024	2KB	[23:22]	[21:20]	[19]	[18:10]	[9:4]	[3:2]	[1:0]
	Dual Die									
0011	1Gb	1024	2KB	[23:22]	[21]	N/A	[19:10]	[9:4]	[3:2]	[1:0]
0100	2Gb	2048	2KB	[23:22]	[21]	[20]	[19:10]	[9:4]	[3:2]	[1:0]
	Dual Die									
0100	2Gb	2048	2KB	[23:22]	N/A	N/A	[20:10]	[9:4]	[3:2]	[1:0]
0101	4Gb	4096	2KB	[23:22]	N/A	[21]	[20:10]	[9:4]	[3:2]	[1:0]
	Dual Die									
0101	4Gb	4096	2KB	[23:22]	N/A	N/A	[21:10]	[9:4]	[3:2]	[1:0]

7.3. 命令映射

有四种命令支持 OneNand flash 存储控制器。这些命令通过引入地址的第 23 位和 22 位的值被选择。命令映射决定低 22 位的地址被使用的方式。

1. “00” =Map 00 命令。

Map 00 命令是用来存取控制器选定的缓冲区,在Flash存储设备里,包括引导,dataram0 和 dataram1。地址总是从 0x0 开始。内存控制器将映射地址到适当的缓冲区。通过 boot\_buffer\_size 和 data\_buffer\_size 寄存器,用户可以知道缓冲器的最大地址。

在大多数情况下,缓冲区主要用于引导缓冲,用户将使用这个命令从这个缓冲区读取数据。然而,Map



00 命令也用于读/修改/写操作。读/修改/写命令是有顺序的，Map 00 命令在缓冲区可以用于读或写任意字，内存控制器将自动选择 dataram 0 或 dataram 1。如表 7-3 所示，对 Map 00 地址映射进行了详细的描述。

表 7-3 Map 00 地址映射

地址位	名称	描述
31:24	AHB_int_add	AHB 端口地址
23:22	CMD_MAP	00=读或写 XIP 缓冲区
21:17	RESV	保留空间
16:1	MEM_ADDR	存储设备上的缓冲地址
0	BYTE	必须设置为 0

2. “01” =Map 01 命令。

Map 01 命令是用于标准的高速访问存储器数组。通过 FBA 和 FPA 命令的指定，用户可以读和写数据到特殊的数组页。一旦 OneNAND Flash 控制器仅支持页地址（FSA 必须设置为 0）。一个完整的页必须读或写一段时间。命令的实际使用数量，要看数据传输的大小。甚至多个命令被请求，必须使用相同的地址，直到整个块都被发送完。

每读或写数据字都将占用 32 位的大小。异步缓冲器将写入数据分成两个 16 位字词或连接读取数据转化为一个 32 位字。如表 7-4 所示，对 Map 01 地址映射进行了详细的描述。

表 7-4 Map 01 地址映射

地址位	名称	描述
31:24	AHB_int_add	AHB 端口地址
23:22	CMD_MAP	01=读或写到内存设备
21:0	MEM_ADDR	参照表 6-27

3. “10” =Map 10 命令。

Map 10 命令是用于控制特殊功能的存储器设备。这是一个数据通过路径的命令，目的是内存控制器，而不是内存设备。不像其他的命令类型，数据（输入或输出）的这些相关处理，并不影响内存的容量，而是用于指定和执行内存控制器的确切命令。输入和输出数据流将始终为 32 位。但只有低 16 位的数据通道包含相关的信息。如表 7-5 所示，对 Map 10 地址映射进行了详细的描述。

表 7-5 Map 10 地址映射

地址位	名称	描述
31:24	AHB_int_add	AHB 端口地址
23:22	CMD_MAP	10=启动一个特殊功能的 Flash 设备或读 状态内存控制器
21:0	MEM_ADDR	参照表 6-27

(1) 擦除操作

OneNand Flash 控制器支持单块或多块擦除。如果内存设备支持多块擦除，擦除是并行执行的。如果内存设备不支持多块擦除，则擦除操作将按顺序执行。当使用多块擦除动作时，用户将指定每一块的地址，然后发出一个单块擦除命令，用于最终启动整个块擦除操作。

实际擦除命令的使用是由 AHB 总线（读或写）和低字节数据输入总线上的处理类型来决定的。如表 7-6 所示。

表 7-6 擦除操作

地址	命令类型	数据输入	功能
[23:22]	写	0x00	保存当前的擦除操作状态到内存控制器
=10 DFS_DBS和 FBA被使用。 FPA 和 FSA 不被使用， 擦除操作 必须被清除。	读	-	如果前面的命令是 Map 10 写入 0x00，则这个命令返回擦除状态。如果前面的命令不是 Map 10 写入 0x00，则如擦除已经完成，返回为 0  0=没有擦除操作在进行，或擦除操作已经完成  1=擦除操作正在进行
	写	0x01	保存多块擦除的块地址。没有启动擦除操作。
	写	0x03	保存单块擦除的块地址，启动擦除操作。也用于指定最终块的多块擦除并启动多块擦除操作。

下面是 ARM11 处理器中，OneNand Flash 控制器擦除块操作的部分代码，参考如下：  
//ONENAND\_EraseBlock 函数：主要功能实现擦除存储器块。

```

//输入: Controller - OneNand 控制器端口数
//      uStartBlkAddr - 擦除块开始地址
//      uEndBlkAddr - 擦除块结束地址
#if 0
OneNAND_eINTERIOR ONENAND_EraseBlock(u32 Controller, u32 uStartBlkAddr, u32 uEndBlkAddr)
{
    u32 i;
    u32 uStatus;
    if(uEndBlkAddr < uStartBlkAddr)
        return eOND_NOERROR;
    OneNandT_oIntFlag.IntActInt = 0;
    OneNandT_oIntFlag.ErsFailInt = 0;
    OneNandT_oIntFlag.LockedBlkInt = 0;
    OneNandT_oIntFlag.ErsCmpInt = 0;
    if(uStartBlkAddr == uEndBlkAddr)
    {
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x03);
        while(!OneNandT_oIntFlag.IntActInt);
    }
    else
    {
        for(i=uStartBlkAddr ; i<uEndBlkAddr ; i++)
        {
            ONENAND_WriteCmd(Controller, i, 0, 0x01);
        }
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x03);
        while(!OneNandT_oIntFlag.IntActInt);
    }
}
#endif

```

```

        for(i=uStartBlkAddr ; i<=uEndBlkAddr ; i++)
        {
            ONENAND_WriteCmd(Controllor, i, 0, 0x03);
            while(!OneNandT_oIntFlag.IntActInt);
        }
#endif

// 校验擦除块
for(i=uStartBlkAddr ; i<=uEndBlkAddr ; i++)
{
    OneNandT_oIntFlag.IntActInt = 0;
    ONENAND_WriteCmd(Controllor, i, 0, 0x15);
    while(!OneNandT_oIntFlag.IntActInt);

    if(OneNandT_oIntFlag.ErsFailInt == 1)
    {
        OneNandT_oIntFlag.ErsFailInt = 0;
        if(OneNandT_oIntFlag.LockedBlkInt == 1)
            return (OneNAND_eINTERERROR) (eOND_ERSFAIL | eOND_LOCKEDBLK);
        return eOND_ERSFAIL;
    }
}

if(OneNandT_oIntFlag.ErsCmpInt == 1)
{
    return eOND_NOERROR;
}
else
    return eOND_ERSFAIL;
}

#if 0

```

```

uError = Inp32(&ONENAND(Controller)->rIntErrStat);
if (uError & (1<<3))
{
    //擦除操作失败
    Outp32(&ONENAND(Controller)->rIntErrAck, (1<<3));
    if(uError & (1<<8))
    {
        Outp32(&ONENAND(Controller)->rIntErrAck, (1<<8));
        return (eOND_ERSFAIL | eOND_LOCKEDBLK);
    }
    return eOND_ERSFAIL;
}
#elif 0
if(OneNandT_oIntFlag.ErsFailInt == 1)
{
    if(OneNandT_oIntFlag.LockedBlkInt == 1)
        return (OneNAND_eINTERERROR)(eOND_ERSFAIL | eOND_LOCKEDBLK);
    return eOND_ERSFAIL;
}
else if(OneNandT_oIntFlag.BlkRwCmpInt == 1)
{
    return eOND_NOERROR;
}
#endif
#if 0
    ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x00);
    ONENAND_ReadCmd(Controller, uEndBlkAddr, 0, &uData);

    // 等待擦除完成

```

```

while(uData)
{
    ONENAND_ReadCmd(Controller, uEndBlkAddr, &uData);
}
return eOND_ERSCMP;
#endif

if(OneNandT_oIntFlag.ErsCmpInt == 1)
    return eOND_NOERROR;
else
    return eOND_ERSFAIL;
}

#else

OneNAND_eINTERROR ONENAND_EraseBlock(u32 Controller, u32 uStartBlkAddr, u32 uEndBlkAddr)
{
    u32 i, j, uBlockSize, uQuotient, uRemainder;
    u32 uBlock;
    if(uEndBlkAddr < uStartBlkAddr)
        return eOND_NOERROR;
    OneNandT_oIntFlag.IntActInt = 0;
    OneNandT_oIntFlag.ErsFailInt = 0;
    OneNandT_oIntFlag.LockedBlkInt = 0;
    OneNandT_oIntFlag.ErsCmpInt = 0;
    if(uStartBlkAddr == uEndBlkAddr)
    {
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x03);
        while(!OneNandT_oIntFlag.IntActInt);
        OneNandT_oIntFlag.IntActInt = 0;
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x15);
        while(!OneNandT_oIntFlag.IntActInt);
    }
}

```

```

if(OneNandT_oIntFlag.ErsFailInt == 1)
{
    OneNandT_oIntFlag.ErsFailInt = 0;
    if(OneNandT_oIntFlag.LockedBlkInt == 1)
        return (OneNAND_eINTERERROR)(eOND_ERSFAIL | eOND_LOCKEDBLK);
    return eOND_ERSFAIL;
}
}
else
{
    uBlockSize = uEndBlkAddr - uStartBlkAddr + 1;
    uQuotient = uBlockSize/OND_MULTIERASE_SIZE;
    uRemainder = uBlockSize%OND_MULTIERASE_SIZE;
    uBlock = uStartBlkAddr;

    for(i=0 ; i<uQuotient ; i++)
    {
        for(j=uBlock ; j<(uBlock+OND_MULTIERASE_SIZE-1) ; j++)
        {
            OneNandT_oIntFlag.IntActInt = 0;
            ONENAND_WriteCmd(Controller, j, 0, 0x01);
            while(!OneNandT_oIntFlag.IntActInt);
        }
        ONENAND_WriteCmd(Controller, uBlock+OND_MULTIERASE_SIZE-1, 0, 0x03);
        while(!OneNandT_oIntFlag.IntActInt);

        //校验擦除块
        for(j=uBlock ; j<=(uBlock+OND_MULTIERASE_SIZE-1) ; j++)

```

```

{
    OneNandT_oIntFlag.IntActInt = 0;
    ONENAND_WriteCmd(Controller, j, 0, 0x15);
    while(!OneNandT_oIntFlag.IntActInt);
    if(OneNandT_oIntFlag.ErsFailInt == 1)
    {
        OneNandT_oIntFlag.ErsFailInt = 0;
        if(OneNandT_oIntFlag.LockedBlkInt == 1)
            return (OneNAND_eINTERERROR) (eOND_ERSFAIL | eOND_LOCKEDBLK);
        return eOND_ERSFAIL;
    }
}

uBlock += OND_MULTIERASE_SIZE;
}

if(uRemainder > 0)
{
    for(i=uBlock ; i<(uBlock+uRemainder-1) ; i++)
    {
        OneNandT_oIntFlag.IntActInt = 0;
        ONENAND_WriteCmd(Controller, i, 0, 0x01);
        while(!OneNandT_oIntFlag.IntActInt);
    }
    ONENAND_WriteCmd(Controller, uBlock+uRemainder-1, 0, 0x03);
    while(!OneNandT_oIntFlag.IntActInt);

    //校验擦除块
    for(i=uBlock ; i<=(uBlock+uRemainder-1) ; i++)
    {
        OneNandT_oIntFlag.IntActInt = 0;

```



```

        ONENAND_WriteCmd(Controller, i, 0, 0x15);
        while(!OneNandT_oIntFlag.IntActInt);

        if(OneNandT_oIntFlag.ErsFailInt == 1)
        {
            OneNandT_oIntFlag.ErsFailInt = 0;
            if(OneNandT_oIntFlag.LockedBlkInt == 1)
                return (OneNAND_eINTERERROR) (eOND_ERSFAIL | eOND_LOCKEDBLK);
            return eOND_ERSFAIL;
        }
    }
}

if(OneNandT_oIntFlag.ErsCmpInt == 1)
    return eOND_NOERROR;
else
    return eOND_ERSFAIL;
}
#endif

```

## （2）封锁（Lock），解锁（Unlock）和锁住（Lock-Tight）操作

OneNAND Flash 控制器支持所有的闪存加锁操作。然而，内存设备对这个功能的支持可能有所限制。如果不支持锁功能，所有有关命令都会忽略。如果不支持“unlock all”功能，将会触发中断。

一块内存区域能被加锁也能被“locked-tight”。一旦一个区域被“locked-tight”，解锁则需要复位。

如果存储设备支持区段锁定，那么通过使用多条指令可以加锁或者解锁多个范围。这种特性通过 lock\_bit\_per\_block 寄存器来控制。当使用时，如果开始和最终地址能指定范围，那么任何“unlock”命令都被转化为“unlock all”命令。实际上，使用 lock/unlock/lock-tight 命令是由 AHB 总线处理（读或写）和 datain 总线的低字节决定的。如表 7-7 所示。

表 7-7 锁（Lock），解锁（Unlock）和锁住（Lock-Tight）操作

地址	命令类型	数据输入	功能
[23:22] =10 DFS_DBS和 FBA被用 FPA 和FSA 不用于 锁操作，必须清除。	写	0x08	为解锁保存开始地址
	写	0x09	为解锁和初始化解锁保存终止地址
	写	0x0A	为加锁保存起始地址
	写	0x0B	为加锁和初始化加锁保存终止地址
	写	0x0C	为 lock-tight 保存起始地址 注：存储控制器发送“lock-tight”指令之前不会校验指定区段是否加锁，但是“lock-tight”只能在加锁区段成功执行
	写	0x0D	为 lock-tight 和初始化 lock-tight 保存终止地址。 注：存储控制器发送“lock-tight”指令之前不会校验指定区段是否加锁，但是“lock-tight”只能在加锁区段成功执行
	写	0x0E	解锁整个存储器列阵。如果不支持这个功能，将产生一个指令错误中断

下面是 ARM11 处理器中，OneNand Flash 控制器锁（Lock），解锁（Unlock）和锁住（Lock-Tight）操作的部分代码，参考如下：

1. 锁 Lock 操作的代码实现：

```
// ONENAND_LockBlock 函数：主要功能实现锁存储器块
// 输入：Controller - OneNand 控制器端口数
//      uStartBlkAddr - 锁开始块
//      uEndBlkAddr - 锁结束块
bool ONENAND_LockBlock(u32 Controller, u32 uStartBlkAddr, u32 uEndBlkAddr)
{
    //u32 uData;
    OneNandT_oIntFlag.IntActInt = 0;
    if(uStartBlkAddr == uEndBlkAddr)
```

```

    {
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x0B);
    }
    else
    {
        ONENAND_WriteCmd(Controller, uStartBlkAddr, 0, 0x0A);
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x0B);
    }

    //while(!(ONENAND(Controller)->rIntErrStat & (1<<10)));
    //Outp32(&ONENAND(Controller)->rIntErrAck, (1<<10));
    while(!OneNandT_oIntFlag.IntActInt);
#endif 0
    ONENAND_DirectRead(Controller, OND_WPROT_STATUS, &uData);
    if ((uData&0xFFFF) != 0x2)
    {
        //UART_Printf(" Failed lock block, locked status (0x%x)\n", uData);
        return TRUE;
    }
#endif
    return FALSE;
}

```

## 2. 解锁 Unlock 操作的代码实现:

```

// ONENAND_UnlockBlock 函数: 主要功能实现解锁存储器块
// 输入: Controller - OneNand 控制器端口数
//       uStartBlkAddr - 解锁开始块
//       uEndBlkAddr - 解锁结束块

```

```

bool ONENAND_UnlockBlock(u32 Controller, u32 uStartBlkAddr, u32 uEndBlkAddr)
{
    //u32 uData;
    OneNandT_oIntFlag.IntActInt = 0;
    if(uStartBlkAddr == uEndBlkAddr)
    {
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x09);
    }
    else
    {
        ONENAND_WriteCmd(Controller, uStartBlkAddr, 0, 0x08);
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x09);
    }

    //while(!(ONENAND(Controller)->rIntErrStat & (1<<10)));
    //Outp32(&ONENAND(Controller)->rIntErrAck, (1<<10));
    while(!OneNandT_oIntFlag.IntActInt);
    #if 0
        ONENAND_DirectRead(Controller, OND_WPROT_STATUS, &uData);
        if ((uData&0xFFFF) != 0x4)
        {
            //UART_Printf(" Failed unlock block, locked status (0x%x)\n", uData);
            return TRUE;
        }
    #endif
    return FALSE;
}

```

### 3. 锁住 Lock-Tight 操作的代码实现:

```

// ONENAND_LockTightBlock 函数：主要功能实现锁住存储器块
// 输入：Controller - OneNand 控制器端口数
//      uStartBlkAddr - 锁住开始块
//      uEndBlkAddr - 锁住结束块
bool ONENAND_LockTightBlock(u32 Controller, u32 uStartBlkAddr, u32 uEndBlkAddr)
{
    //u32 uData;
    OneNandT_oIntFlag.IntActInt = 0;
    if(uStartBlkAddr == uEndBlkAddr)
    {
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x0D);
    }
    else
    {
        ONENAND_WriteCmd(Controller, uStartBlkAddr, 0, 0x0C);
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x0D);
    }

    //while(!(ONENAND(Controller)->rIntErrStat & (1<<10)));
    //Outp32(&ONENAND(Controller)->rIntErrAck, (1<<10));
    while(!OneNandT_oIntFlag.IntActInt);
#if 0
    ONENAND_DirectRead(Controller, OND_WPROT_STATUS, &uData);
    if ((uData&0xFFFF) != 0x1)
    {
        //UART_Printf(" Failed lock block, locked status (0x%x)\n", uData);
        return TRUE;
    }
#endif
}
#endif

```

```

return FALSE;
}

```

### (3) 回拷贝式操作

OneNand Flash 控制器支持拷贝操作。然而，内存设备对这个功能的支持可能有所限制。如果不支持回拷贝式功能，将会触发一个中断。如果在目的块被指定前，源块没有进行设置，也将有一个中断被触发。

参数 PP 用来设置回拷贝的页数。这是用来在一个命令中拷贝多个连续页面。实际的回拷贝式命令的使用是通过 AHB 总线（读或写）和低两个字节的数据输入总线上的处理类型来决定的。如表 7-8 所示。

表 7-8 回拷贝式操作

地址	命令类型	数据输入	功能
[23:22] =10	写	0x1000	为拷贝保留源地址
DFS_DBS, FBA, FPA 和FSA被使用。	写	0x20PP	为拷贝保留目的地址 从源地址到目的地址启动一个 PP 页的拷贝

下面是 ARM11 处理器中，OneNand Flash 控制器回拷贝式操作的部分代码，参考如下：

//ONENAND\_CopyBack 函数：主要功能实现回拷贝试操作

// 输入：Controller - OneNand 控制器端口数

// uSourceBlkAddr - 源块数

// uSourcePageAddr - 源页数

// uDestinationBlkAddr - 目的块数

// uDestinationPageAddr - 目的页数

// uPageSize - 被复制的页大小

```

bool ONENAND_CopyBack(u32 Controller, u32 uSourceBlkAddr, u8 uSourcePageAddr, u32
uDestinationBlkAddr, u8 uDestinationPageAddr, u32 uPageSize)
{
    OneNandT_oIntFlag.IntActInt = 0;
    OneNandT_oIntFlag.BlkRwCmpInt = 0;
    OneNandT_oIntFlag.PgmCmpInt = 0;

```

```

    ONENAND_WriteCmd(Controller, uSourceBlkAddr, uSourcePageAddr, 0x1000);
    ONENAND_WriteCmd(Controller,          uDestinationBlkAddr,          uDestinationPageAddr,
(0x2000|(uPageSize&0xFF)));

    //while(!(ONENAND(Controller)->rIntErrStat & (1<<10)));
    //Outp32(&ONENAND(Controller)->rIntErrAck, (1<<10));
    while(!OneNandT_oIntFlag.IntActInt);

    //while(!(ONENAND(Controller)->rIntErrStat & (1<<7)));
    //Outp32(&ONENAND(Controller)->rIntErrAck, (1<<7));
    while(!OneNandT_oIntFlag.BlkRwCmpInt);

    //while(!(ONENAND(Controller)->rIntErrStat & (1<<5)));
    //Outp32(&ONENAND(Controller)->rIntErrAck, (1<<5));
    while(!OneNandT_oIntFlag.PgmCmpInt);

    return TRUE;
}

```

#### (4) OTP 和备用区域访问操作

Flash 数组有三个主要部分，OneNand flash 控制器支持对所有部分的访问。然而，内存设备对这些区域的支持可能会有所限制。如果不支持访问到 OTP 或备用区域，将会触发一个中断。命令是冲突的或不能并行执行，将被阻断，直到冲突的命令完成。

实际上，访问命令的使用是通过 AHB 总线（读或写）和低字节的数据输入总线上的处理类型来决定的。如表 7-9 所示。

表 7-9 OTP 和备用区域访问操作

地址	命令类型	数据输入	功能
[23:22] =10	写	0x12	控制器进行配置，以获得 Flash 的 OTP 区域
DFS_DBS, FBA, FPA 和FSA被使用	写	0x14	控制器进行配置，以获得 Flash 的主数据区域 这是默认区域，但是在访问 OTP 和备用区域后需要复位

(5) 读校验操作

多区段擦除必须通过读校验指令校验，这个指令连续操作并且一次校验一个区段。在多区段擦除中，每个区段都必须校验。如果某个区段没有校验，将会产生中断（Ers\_Fail）。

使用的读校验指令由 AHB 总线的处理（读或写）类型和 datain 总线的低字节来决定。如表 7-10 所示。

表 7-10 读校验操作

地址	指令类型	数据输入	功能
[23:22] = 10  DFS_DBS和FBA 被使用  FPA 和 FSA 没有使用，必须清除	写	0x15	为擦除校验保存地址和为初始化校验指令

(6) 管线预读和预写操作

OneNAND 闪存控制器支持管线预先读取和写入操作。但是，存储设备可能限制支持这些功能，如果不支持管线预读取和写入操作，这些指令将被忽略。

管线预读功能允许连续读取存储器，甚至没有给控制器发出读请求。管线预读功能利用控制器中的一系列寄存器来控制所给的地址中数据起始页。读取的内容作为连续的内容被访问。数据要先于发送读请求到达存储控制器，这使内存控制器能够减少返回读取数据到 AHB 总线的延迟。

管线预写入功能允许 AHB 接口接受写入数据，即使写请求没有发送到控制器。管线预写入功能利用控制器中的一系列寄存器来管理所给的地址中数据开始页，这些数据作为连续内容。通过数据在写请求发出之前到达存储控制器，而使内存控制器减少写取数据到闪存的延迟。

注意，缓冲寄存器中预读和预写指令是相同的。执行操作前，数据预读必须读到 AHB 接口，数据预写必须写入闪存。

管线预读和预写操作如表 7-11 所示。



表 7-11 管线预读和预写操作

地址	命令类型	数据输入	功能
[23:22] = 10 DFS_DBS、FBA、FPA和FSA 被用	写	0x4WPP	为读和写保存初始地址。对于读，“W”=0，将初始化从存储器到存储控制器的 PP 页。对于写，“W”=1，将初始化从 AHB 接口到存储控制器

注：对于单一区域，管线预读和预写要求必须至少两页。

（7）读/修改/写操作

用户可能需要读取指定页，或者只修改少量的字、字节或位。这种情况下，可以使用读/写/修改操作。

读指令将数据从存储器输入到缓冲区。用户可以修改缓冲区的信息，然后发送写指令写回到存储器。

使用读/修改/写指令由 AHB 总线的处理（读或写）类型和 datain 总线的低字节决定。如表 7-12 所示。

表 7-12 读/修改/写操作

地址	命令类型	数据输入	功能
[23:22] = 10 DFS_DBS、FBA、FPA和FSA 被使用	写	0x10	通过FBA、 FPA 和FSA 载入指定页到 map 00 XIP 缓冲寄存器
	写	0x11	在 map 00 XIP 缓冲区通过 FBA、 FPA 和 FSA。写数据到指定页

4. “11”=Map 11 命令。

Map 11 指令用于存储器直接访问存储设备。这个指令主要用于测试和调试错误，通过用户直接和读/写数据到指定地址或寄存器的部分。虽然这种访问类型允许直接和存储器（boot，数据 RAM0，数据 RAM1，spare, 备用命令寄存器, 寄存器和 OTP）接触，但是缓慢而繁琐并且仅用于必要时。只有低半字用于数据传输。如表 7-13 所示。

表 7-13 Map 11 地址映射

地址位	名称	描述
31:24	AHB_int_add	AHB 端口地址
23:22	CMD_MAP	11 = OneNAND 读或写
21:18	RESV	保留空间
17:2	Address	存储设备地址

1:0	RESV	必须置为 0
-----	------	--------

## 7.4 管线预读/写指令

当收到管线预读或者预写指令后，控制器处于空闲状态，这个指令将立即开始一个载入操作到存储设备的数据随机存储器。注意的是，预读命令并不返回数据到 AHB 接口，预写指令也不写数据到闪存地址。读数据载入后，只有当 map 01 指令要求读这些数据时，读数据才被返回到 AHB 端口。同样，写数据载入后，只有当 map 01 指令要求写这些数据时，数据才被写入闪存。

对于预读指令，控制器内部管理读操作请求。控制器将为每一页请求发出缓存读取到闪存设备。如果设置了 transfer\_spare 寄存器，那么在管线预读请求期间，主要数据区和备用数据区将转移到存储控制器。

管线预读/预写指令可以被顺序输入。如果现有的预读/预写指令已经读/写了所有数据，管线预读指令后面可能跟着是一个管线预写指令，反之亦然。

如果正在处理当前组的读或者写，那么在这一系列的操作完成前，内存控制器将会挂起新的指令。OneNand 闪存控制器最多可以同时执行 3 条预读或者预写指令。所有的预读管线数据从缓冲区读取前，或者所有的预写数据通过 map 01 指令写入内存前，控制器不会返回空闲状态。

管线预读/预写操作优先于寄存器编程操作。

### 1. 为管线预读设置一个单区域

为管线预读建立一个单区域的步骤如下：

(1) 用户必须设置“Map 10”中的 CMD\_MAP，并且在地址的 FBA、FPA 和 FSA 设置预读区段的起始地址。

(2) 指令类型必须设置为“写”，并且数据输入总线必须设置为“0x40PP”。其中，“0”表示预读，PP 是预读的页数。

(3) 要读这些数据，必须发送 Map 01 到存储控制器以相同的起始地址和目的页。如果按照管线预读请求被接收的读指令不是页预读，那么中断为将会被设置，管线预读/写寄存器将会清除。必须发送一个新的管线预读请求来重载相同的数据。

(4) 在控制器回到空闲状态前所有被预读的数据必须被通过 Map 01 命令读。

注：对于单区域的管线预读或预写要求至少两页。

## 2. 为管线预写设置一个单区域

为管线预写建立一个单区域的步骤如下：

（1）用户必须设置“Map 10”中的 CMD\_MAP，并且在地址的 FBA、FPA 和 FSA 设置预写区段的起始地址。

（2）指令类型必须设置为“写”，并且数据输入总线必须设置为“0x41PP”。其中，“1”表示预写，PP 是预写的页数。

（3）要写这些数据，必须发送 Map 01 到存储控制器以相同的起始地址和目的页。如果按照管线预写请求被接收的写指令不是页预写，那么中断为将会被设置，管线预读/写寄存器将会被清除。必须发送一个新的管线预写请求来重新载入相同的数据。

（4）在控制器回到空闲状态前所有被预写的数据必须通过 Map 01 命令写入。

注：对于单区域的管线预读或预写要求至少两页。

## 3. 为管线预读设置多区域

为管线预读建立多区域的步骤如下：

（1）用户必须设置“Map 10”中的 CMD\_MAP，并且在地址的 FBA、FPA 和 FSA 设置预读的第一个区段的起始地址。

（2）指令类型必须设置为“写”，并且数据输入总线必须设置为“0x40PP”。其中，“0”表示预读，PP 是预读的页数。

（3）如果用户希望任何数据返回 AHB 接口，Map 01 指令必须发送到存储控制器以相同的起始地址和目的页。这些读指令必须是预读页中的地址。在发送其它管线预读指令前，用户不需要读所有的数据甚至任何数据。

（4）用户必须设置“Map 10”中的 CMD\_MAP，并且在地址的 FBA、FPA 和 FSA 设置第二个预读区段的起始地址。

（5）指令类型必须设置为“写”，并且数据输入总线必须设置为“0x40PP”。其中，“0”表示预读，PP 是预读的页数。

（6）要从第一个或者第二个预读页读取任何数据，首先必须发送 Map 01 指令到任意地址。

（7）按照（4）～（6）步骤读其它的区段。Map 01 读能最多交叉执行三个管线预读或者管线预写指

令。但是在所有的预读数据通过 AHB 接口读完前，OneNand 闪存控制器不会返回空闲状态。

#### 4. 为管线预写设置多区段

为管线预写设置多区段的步骤如下：

（1）用户必须设置“Map 10”中的 CMD\_MAP，并且在地址的FBA、FPA 和FSA设置预写的第一个区段的起始地址。

（2）指令类型必须设置为“写”，并且数据输入总线必须设置为“0x41PP”。其中，“1”表示预写，PP是预写的页数。

（3）如果用户希望任何数据写入闪存，Map 01 指令必须发送到存储控制器以相同的起始地址和目的页。这些写指令必须是预写页中的地址。在发送其它管线预写指令前，用户不需要写所有的数据甚至任何数据。

（4）用户必须设置“Map 10”中的 CMD\_MAP，并且在地址的 FBA、FPA 和 FSA 设置第二个预写区段的起始地址。

（5）指令类型必须设置为“写”，并且数据输入总线必须设置为“0x41PP”。其中，“1”表示预写，PP 是预写的页数。

（6）要向第一个或者第二个预写页写入任何数据，首先必须发送 Map 01 指令到任意地址。

（7）按照（4）～（6）步骤预写其它的区段。Map 01 读能最多交叉执行三个管线预读或者管线预写指令。但是在所有的预写数据通过 AHB 接口写完前，OneNand 闪存控制器不会返回空闲状态。

## 7.5 控制器使用中其它事项

（1）在复位寄存器时，OneNAND闪存控制器不作任何时间限制。

（2）OneNAND控制器的脉冲长度必须设置为等于或者小于OneNAND闪存设备的脉冲长度值。

（3）在一页中，AMBA处理必须是相同的脉冲类型。在页发送过程中，在数据随机存储器为了另一个传输而释放前，整页必须被发送到存储器或者AMBA总线。

（4）访问寄存器必须使用一个信号的传输类型。

（5）数据存取必须一个字的传输大小。

（6）当使用OneNAND控制器的时，尤其是‘11’指令，建议设置IOBE领域。

## 7.6 用 ONENAND 控制器启动

- (1) 外部复位无效。
- (2) 系统控制器全部复位。
- (3) OneNAND 控制器通过内核时钟开始计数“FLASH\_COLD\_RST\_DELAY”。
- (4) 在ARM内核复位无效后，开始取指令，但被OneNAND控制器暂停。
- (5) 但当计数值达到预先确定的FLASH\_COLD\_RST\_DELAY值（为 $534 \times \text{ACCESS\_CLK}$ 周期）时，OneNAND 控制器开始读相应的存储器中的信息。
  - ①相应的存储器信息由厂商ID号、设备ID号、版本ID号、数据缓冲区大小、引导缓冲区大小、缓冲区数量和技术组成。
  - ②当发送第一次读信息请求时，OneNAND控制器使XmORPn无效。
- (6) 读完相关信息时，OneNAND开始为进行数据处理请求服务。

## 7.7 ONENAND 寄存器描述

下面主要针对 ONENAND 控制器中各个寄存器进行详细的描述。

各种不同的存储器的映射。

寄存器	地址	读/写	描述	复位值
MEM_CFG0	0x70100000	读/写	Bank0 内存设备配置寄存器。	0x0000
BURST_LEN0	0x70100010	读/写	Bank0 脉冲长度寄存器。	0x0000
MEM_RESET0	0x70100020	读/写	Bank0 内存复位寄存器。	0x0000
INT_ERR_STAT0	0x70100030	读/写	Bank0 中断错误状态寄存器。	0x0000
INT_ERR_MASK0	0x70100040	读/写	Bank0 中断错误屏蔽寄存器。	0x0000
INT_ERR_ACK0	0x70100050	写	Bank0 中断错误确认寄存器。	0x0000
ECC_ERR_STAT0	0x70100060	读/写	Bank0 ECC 错误状态寄存器。	0x0000
MANUFACT_ID0	0x70100070	读	Bank0 制造商 ID 寄存器。	Mem. dep.
DEVICE_ID0	0x70100080	读	Bank0 设备 ID 寄存器。	Mem. dep.
DATA_BUF_SIZE0	0x70100090	读	Bank0 数据缓冲大小寄存器。	Mem. dep.

BOOT_BUF_SIZE0	0x701000A0	读	Bank0 引导缓冲大小寄存器。	Mem. dep.
BUF_AMOUNT0	0x701000B0	读	Bank0 缓冲数量寄存器。	Mem. dep.
TECH0	0x701000C0	读	Bank0 技术寄存器。	Mem. dep.
FBA_WIDTH0	0x701000D0	读/写	Bank0 FBA 宽度寄存器。	0x000A
FPA_WIDTH0	0x701000E0	读/写	Bank0 FPA 宽度寄存器。	0x0006
FSA_WIDTH0	0x701000F0	读/写	Bank0 FSA 宽度寄存器。	0x0002
REVISION0	0x70100100	读	Bank0 修正寄存器。	0x0002
DATARAM00	0x70100110	读/写	Bank0 Dataram0 编码寄存器。	0x0002
DATARAM10	0x70100120	读/写	Bank0 Dataram1 编码寄存器。	0x0003
SYNC_MODE0	0x70100130	读	Bank0 同步模式寄存器。	0x0000
TRANS_SPARE0	0x70100140	读/写	Bank0 传输大小寄存器。	0x0000
LOCK_BIT0	0x70100150	读/写	Bank0 每块锁位寄存器。	0x0000
DBS_DFS_WIDTH0	0x70100160	读/写	Bank0 DBS_DFS 宽度寄存器。	0x0000
PAGE_CNT0	0x70100170	读	Bank0 页计数寄存器。	0x0000
ERR_PAGE_ADDRO	0x70100180	读	Bank0 错误页地址寄存器。	0x0000
BURST_RD_LAT0	0x70100190	读	Bank0 脉冲读延迟寄存器。	0x0006
INT_PIN_ENABLE0	0x701001A0	读/写	Bank0 中断引脚使能寄存器。	0x0000
INT_MON_CYC0	0x701001B0	读/写	Bank0 中断监控周期计数寄存器。	0x01F4
ACC_CLOCK0	0x701001C0	读/写	Bank0 地址时钟寄存器。	0x0003
SLOW_RD_PATH0	0x701001D0	读/写	Bank0 缓慢读通道寄存器。	0x0000
ERR_BLK_ADDRO	0x701001E0	读	Bank0 错误块地址寄存器。	0x0000
FLASH_VER_ID0	0x701001F0	读	Bank0 Flash 版本 ID 寄存器。	Mem. dep.

寄存器	地址	读/写	描述	复位值
MEM_CFG1	0x70180000	读/写	Bank1 内存设备配置寄存器。	0x0000
BURST_LEN1	0x70180010	读/写	Bank1 脉冲长度寄存器。	0x0000
MEM_RESET1	0x70180020	读/写	Bank1 内存复位寄存器。	0x0000
INT_ERR_STAT1	0x70180030	读/写	Bank1 中断错误状态寄存器。	0x0000

INT_ERR_MASK1	0x70180040	读/写	Bank1 中断错误屏蔽寄存器。	0x0000
INT_ERR_ACK1	0x70180050	读/写	Bank1 中断错误确认寄存器。	0x0000
ECC_ERR_STAT1	0x70180060	写	Bank1 ECC 错误状态寄存器。	0x0000
MANUFACT_ID1	0x70180070	读	Bank1 制造商 ID 寄存器。	Mem. dep.
DEVICE_ID1	0x70180080	读	Bank1 设备 ID 寄存器。	Mem. dep.
DATA_BUF_SIZE1	0x70180090	读	Bank1 数据缓冲大小寄存器。	Mem. dep.
BOOT_BUF_SIZE1	0x701800A0	读	Bank1 引导缓冲大小寄存器。	Mem. dep.
BUF_AMOUNT1	0x701800B0	读	Bank1 缓冲数量寄存器。	Mem. dep.
TECH1	0x701800C0	读	Bank1 技术寄存器。	Mem. dep.
FBA_WIDTH1	0x701800D0	读/写	Bank1 FBA 宽度寄存器。	0x000A
FPA_WIDTH1	0x701800E0	读/写	Bank1 FPA 宽度寄存器。	0x0006
FSA_WIDTH1	0x701800F0	读/写	Bank1 FSA 宽度寄存器。	0x0002
REVISION1	0x70180100	读	Bank1 修正寄存器。	0x0002
DATARAM01	0x70180110	读/写	Bank1 Dataram0 编码寄存器。	0x0002
DATARAM11	0x70180120	读/写	Bank1 Dataram1 编码寄存器。	0x0003
SYNC_MODE1	0x70180130	读	Bank1 同步模式寄存器。	0x0000
TRANS_SPARE1	0x70180140	读/写	Bank1 传输大小寄存器。	0x0000
LOCK_BIT1	0x70180150	读/写	Bank1 每块锁位寄存器。	0x0000
DBS_DFS_WIDTH1	0x70180160	读/写	Bank1 DBS_DFS 宽度寄存器。	0x0000
PAGE_CNT1	0x70180170	读	Bank1 页计数寄存器。	0x0000
ERR_PAGE_ADDR1	0x70180180	读	Bank1 错误页地址寄存器。	0x0000
BURST_RD_LAT1	0x70180190	读	Bank1 脉冲读延迟寄存器。	0x0006
INT_PIN_ENABLE1	0x701801A0	读/写	Bank1 中断引脚使能寄存器。	0x0000
INT_MON_CYC1	0x701801B0	读/写	Bank1 中断监控周期计数寄存器。	0x01F4
ACC_CLOCK1	0x701801C0	读/写	Bank1 地址时钟寄存器。	0x0003
SLOW_RD_PATH1	0x701801D0	读/写	Bank1 缓慢读通道寄存器。	0x0000
ERR_BLK_ADDR1	0x701801E0	读	Bank1 错误块地址寄存器。	0x0000
FLASH_VER_ID1	0x701801F0	读	Bank1 Flash 版本 ID 寄存器。	Mem. dep.

注) Mem. dep. :存储器依赖。

以下主要针对个别寄存器做详细的介绍。

7.7.1. 内存设备配置寄存器

可编程的值，将取决于实际的内存设备的使用。这个值与 OneNAND Flash 的系统配置寄存器 1(0xF221) 有关系。这一数据用来配置 Flash 硬件和软件环境，并可能包括脉冲长度、读延迟、传输模式、ECC 配置和极性级等等。这个寄存器是通过初始化软件时被设置的，并且是在没有 OneNAND 存储器访问的时候。

下面描述的是 kfm1g16q2a。

寄存器	地址	读/写	描述	复位值
MEM_CFG0	0x70100000	读/写	Bank0 内存设备配置寄存器	0x00000000
MEM_CFG1	0x70180000			

MEM_CFGn	位	描述	初始状态
Reserved	[31:16]	保留	
RM	[15]	设置传输模式，用于作同步或异步的读取操作。默认值是 0x0. 在初始化时通过软件来设置 <ul style="list-style-type: none"><li>• 0=异步模式</li><li>• 1=激活同步模式</li></ul>	0
BURST_RD_LAT	[14:12]	周期内设置脉冲读延迟 <ul style="list-style-type: none"><li>• 000~010=保留</li><li>• 011=3（高达 40MHz）</li><li>• 100 = 4</li><li>• 101 = 5</li><li>• 110 = 6</li><li>• 111 = 7</li></ul>	100
BURST_LENGTH	[11:9]	设置脉冲长度 <ul style="list-style-type: none"><li>• 000 = 连续</li><li>• 001 = 4位半字</li></ul>	0



		<ul style="list-style-type: none"> <li>• 010 = 8位半字</li> <li>• 011 = 16位半字（推荐）</li> <li>• 100 = 32位半字</li> <li>• 101 = 1K半字（只读块）</li> <li>• 110~111 = 保留</li> </ul>	
ECC	[8]	错误校正操作 <ul style="list-style-type: none"> <li>• 0=校正</li> <li>• 1= 未经校正（通过）</li> </ul>	0
RDYPOL	[7]	RDY 信号性 <ul style="list-style-type: none"> <li>• 0=低信号读</li> <li>• 1=高信号读</li> </ul>	0
INTPOL	[6]	INT 信号性 <ul style="list-style-type: none"> <li>• 0=低中断等待</li> <li>• 1=高中断等待</li> </ul>	0
IOBE	[5]	I/O 缓冲使能 INT 和 RDY 信号，INT 和 RDY 开始输出 hi-Z，IOBE 被设置为 1 后，位 7 和位 6 变为有效。IOBE 只有通过冷复位可以被复位 <ul style="list-style-type: none"> <li>• 0=禁用</li> <li>• 1=使能</li> </ul>	0
RDYCONF	[4]	RDY 配置 <ul style="list-style-type: none"> <li>• 0=激活有效数据</li> <li>• 1=有效数据前激活时钟</li> </ul>	0
Reserved	[3]	保留	
HF	[2]	高频使能 <ul style="list-style-type: none"> <li>• 0=高频禁用（66MHz以下）</li> <li>• 1=高频使能（66MHz 以上）</li> </ul>	0
WM	[1]	设置传输模式为写操作，作为同步或异步。默认值为 0x0 通过软件进行初始设置 <ul style="list-style-type: none"> <li>• 0=异步模式</li> <li>• 1=激活同步模式</li> </ul>	0

BWPS	[0]	启动缓冲器为写保护状态 <ul style="list-style-type: none"> <li>0=锁住</li> <li>1=解锁</li> </ul>	0
------	-----	--	---

### 7.7.2. 脉冲长度寄存器

寄存器	地址	读/写	描述	复位值
BURST_LEN0	0x70100010	读/写	Bank0 脉冲长度寄存器	0x0000
BURST_LEN1	0x70180010			

BURST_LENn	位	描述	初始状态
Reserved	[31:16]	保留	0
BURST_LENGTH	[5:0]	<p>设置脉冲长度（半字数量）的控制器。这个寄存器没有默认值。推荐相同或少于 OneNAND 0xF221 脉冲长度范围的值通过软件进行初始化设置</p> <p>4：支持单一和 INCR4</p> <p>8：支持单一和 INCR4</p> <p>16：支持单一，INCR4 和 INCR8</p> <p>其他：保留</p>	0

### 7.7.3. 内存复位寄存器

寄存器	地址	读/写	描述	复位值
MEM_RESET0	0x70100020	读/写	Bank0 内存复位寄存器	0x0000
MEM_RESET1	0x70180020			

MEM_RESETn	位	描述	初始状态
Reserved	[31:3]	保留	0

RESET_CODE	[2:0]	设置复位编码。该寄存器是在复位顺序完成以后复位到 0x0。主要是通过软件来控制的 <ul style="list-style-type: none"> <li>• 001 =温复位</li> <li>• 010 =冷复位</li> <li>• 011 =热复位</li> <li>• 其他保留</li> </ul>	0
------------	-------	---	---

### 7.7.4. 中断错误状态寄存器

寄存器	地址	读/写	描述	复位值
INT_ERR_STAT0	0x70100030	读/写	Bank0 中断错误状态寄存器	0x0000
INT_ERR_STAT1	0x70180030			

INT_ERR_STATn	位	描述	初始状态
Reserved	[31:14]	保留。	0
CACHE_OP_ERR	[12]	在高速缓冲存储器读或写设置或操作期间发生错误	0
RST_CMP	[12]	控制器完成其复位和初始化过程。确保检查在 OneNAND 访问前该位是否被设置为 1	0
RDY_ACT	[11]	内存设备的 RDY 引脚激活转换	0
INT_ACT	[10]	内存设备的 INT 引脚激活转换	0
UNSUP_CMD	[9]	接收一个不支持的命令。当接收一个无效的命令或是命令顺序被破坏时，该中断被设置	0
LOCKED_BLK	[8]	在保护块内，地址被烧写或擦除	0
BLK_RW_CMP	[7]	块读或写转换完成。可以参照“读校验”和“管线预读”命令	0
ERS_CMP	[6]	擦除操作完成。该中断是在擦除操作开始时自动复位的	0
PGM_CMP	[5]	烧写操作完成。该中断是在烧写操作开始时自动复位的	0
LOAD_CMP	[4]	下载操作完成	0
ERS_FAIL	[3]	擦除操作没有成功	0

PGM_FAIL	[2]	烧写操作没有成功	0
INT_TO	[1]	中断时间	0
LD_FAIL_ECC_ERR	[0]	双重目的中断位。下载操作没有成功或者是一个 ECC 错误	0

注：当 OneNAND 控制器接到冷复位时，它将一直处于等待状态，直到作为冷复位预设置的等待时间结束。冷复位等待周期结束后，它自动产生读取处理以从 OneNAND 存储器获得存储器依赖信息。读处理完成后，RST\_CMP 位变为 1。

### 7.7.5 中断错误屏蔽寄存器

寄存器	地址	读/写	描述	复位值
INT_ERR_MASK0	0x70100040	读/写	Bank0 中断错误屏蔽寄存器	0x0000
INT_ERR_MASK1	0x70180040			

INT_ERR_MASKn	位	描述	初始状态
Reserved	[31: 14]		0
CACHE_OP_ERR	[13]	当一位设置为 0 时，屏蔽 INT_ERR_STAT 寄存器中符合该位的中断。当一位设置为 1 时，允许 INT_ERR_STAT 寄存器中符合该位的中断。	0
RST_CMP	[12]		
RDY_ACT	[11]		
INT_ACT	[10]	设置该位为 1 允许控制器发出该中断类型。通过软件设置。	
UNSUP_CMD	[9]		
LOCKED_BLK	[8]		
BLK_RW_CMP	[7]		
ERS_CMP	[6]		
PGM_CMP	[5]		
LOAD_CMP	[4]		
ERS_FAIL	[3]		
PGM_FAIL	[2]		

INT_TO	[1]		
LD_FAIL_ECC_ERR	[0]		

### 7.7.6. 中断错误确认寄存器

寄存器	地址	读/写	描述	复位值
INT_ERR_ACK0	0x70100050	写	Bank0 中断错误确认寄存器。	0x0000
INT_ERR_ACK1	0x70180050			

INT_ERR_ACKn	位	描述	初始状态
Reserved	[31: 14]		0
CACHE_OP_ERR	[13]	确认 INT_ERR_STAT 寄存器中符合该位的位。将这个位复位或确认中断关系。通过软件设置。	0
RST_CMP	[12]		
RDY_ACT	[11]		
INT_ACT	[10]		
UNSUP_CMD	[9]		
LOCKED_BLK	[8]		
BLK_RW_CMP	[7]		
ERS_CMP	[6]		
PGM_CMP	[5]		
LOAD_CMP	[4]		
ERS_FAIL	[3]		
PGM_FAIL	[2]		
INT_TO	[1]		
LD_FAIL_ECC_ERR	[0]		

7.7.7. ECC 错误状态寄存器

寄存器	地址	读/写	描述	复位值
ECC_ERR_STAT0	0x70100060	读/写	Bank0ECC 错误状态寄存器	0x0000
ECC_ERR_STAT1	0x70180060			

ECC_ERR_STATn	位	描述	初始状态
Reserved	[31:16]		0
ECC_ERR_STAT	[15:0]	该值依赖于实际使用的存储器设备。该数据用于报告 ECC 错误信息	0

7.7.8. 厂商 ID 寄存器

寄存器	地址	读/写	描述	复位值
MANUFACT_ID0	0x70100070	读	Bank0 厂商 ID 寄存器	由存储器决定的
MANUFACT_ID1	0x70180070			

MANUFACT_IDn	位	描述	初始状态
Reserved	[31:16]		
MANUFACT_ID	[15:0]	该值依赖于实际使用的存储器设备。复位后，该寄存器通过闪存控制器设置。只读	

7.7.9. 设备 ID 寄存器

寄存器	地址	读/写	描述	复位值
DEVICE_ID0 DEVICE_ID1	0x70100080 0x70180080	读	Bank0 设备 ID 寄存器	由存储器决定的

DEVICE_IDn	位	描述	初始状态
Reserved	[31:16]		
DEVICE_ID	[15:9]	该值依据实际使用的存储器。复位后，该寄存器通过闪存控制器被设置。只读	
BOOT_INFO	[8]	导入信息。只读  0=底部启动  1=保留	
DENSITY	[7:4]	OneNAND的密度。只读  0000 = 128 Mb  0001 = 256 Mb  0010 = 512 Mb  0011 = 1 Gb  0100 = 2 Gb  0101 = 4 Gb.	
DDP	[3]	数据路径。只读  0 = 单数据路径  1 = 双数据路径	
MUXED	[2]	只读  0 = 多路  1 = 分路	
VCC	[1:0]	只读  00 = 1.8V  01/10/11 = 保留	

7.7.10. 数据缓冲区大小寄存器

寄存器	地址	读/写	描述	复位值
DATA_BUF_SIZE0	0x70100090	读	Bank0 数据缓冲区大小寄存器	由存储器决定
DATA_BUF_SIZE1	0x70180090			

DATA_BUF_SIZE <sub>n</sub>	位	描述	初始状态
Reserved	[31:16]		
DATA_BUF	[15:0]	该值依赖于实际使用的存储设备。复位后，该寄存器通过闪存控制器设置。只读	

7.7.11. 导入缓冲区大小寄存器

寄存器	地址	读/写	描述	复位值
BOOT_BUF_SIZE0	0x701000A0	读	Bank0 导入缓冲区大小寄存器	由存储器决定的
BOOT_BUF_SIZE1	0x701800A0			

BOOT_BUF_SIZE <sub>n</sub>	位	描述	初始状态
Reserved	[31:16]		
BOOT_BUF	[15:0]	该值依赖实际使用的存储设备。复位后，该寄存器通过闪存控制器被设置。只读	

7.7.12. 缓冲区总数寄存器

寄存器	地址	读/写	描述	复位值
BUF_AMOUNT0	0x701000B0	读	Bank0 缓冲区寄存器的总数	由存储器决定的
BUF_AMOUNT1	0x701800B0			



BUF_AMOUNTn	位	描述	初始状态
Reserved	[31:16]		
AMOUNT	[15:0]	该值依赖实际使用的存储设备。复位后，该寄存器通过闪存控制器被设置。 只读	

### 7.7.13. 技术寄存器

寄存器	地址	读/写	描述	复位值
TECH0	0x701000C0	读	Bank0 技术寄存器	由存储器决定的
TECH1	0x701800C0			

TECHn	位	描述	初始状态
Reserved	[31:16]		
TECHNOLOGY	[15:0]	该值依赖实际使用的存储设备。复位后，该寄存器通过闪存控制器。只读	

### 7.7.14. FBA 宽度寄存器

寄存器	地址	读/写	描述	复位值
FBA_WIDTH0	0x701000D0	读/写	Bank0 FBA 宽度寄存器	0x000A
FBA_WIDTH1	0x701800D0			

FBA_WIDTHn	位	描述	初始状态
Reserved	[31:5]		0
FBA	[4:0]	设置块的数量。默认值为 0x0A。初始化期间设置通过软件	0x0A

7.7.15. FPA 宽度寄存器

寄存器	地址	读/写	描述	复位值
FPA_WIDTH0	0x701000E0	读/写	Bank0FPA 宽度寄存器	0x0006
FPA_WIDTH1	0x701800E0			

FPA_WIDTHn	位	描述	初始状态
Reserved	[31:5]	保留。	0
FPA	[4:0]	设置数值，将用来代表页的数值。默认值为 6。初始化期间通过软件设置	0x06

7.7.16. FSA 宽度寄存器

寄存器	地址	读/写	描述	复位值
FSA_WIDTH0	0x701000F0	读/写	Bank0 FSA 宽度寄存器	0x0002
FSA_WIDTH1	0x701800F0			

FSA_WIDTHn	位	描述	初始状态
Reserved	[31:3]		0
FSA	[2:0]	设置位的数值，将用来表示扇区的数值。默认值为 0x2。初始化期间通过软件设置	0x2

7.7.17. 修正寄存器

寄存器	地址	读/写	描述	复位值
REVISION0	0x70100100	读	Bank0 修正寄存器	0x00000002
REVISION1	0x70180100			

REVISIONn	位	描述	初始状态
Reserved	[31:16]		0x0000
REVISION	[15:0]	保存控制器修正数。默认值为 0x1。只读	0x0002

### 7.7.18. DATARAM0 代码寄存器

寄存器	地址	读/写	描述	复位值
DATARAM00	0x70100110	读/写	Bank0 Dataram0 代码寄存器	0x0002
DATARAM01	0x7018110			

DATARAM0n	位	描述	初始状态
Reserved	[31:16]		0
DATARAM0	[3:0]	ram0 设置数据的非扇区部分。默认值为 0x2。在初始化期间通过软件设置	2

### 7.7.19. DATARAM1 代码寄存器

寄存器	地址	读/写	描述	复位值
DATARAM10	0x70100120	读/写	Bank0 Dataram1 代码寄存器	0x0003
DATARAM11	0x7018120			

DATARAM1n	位	描述	初始状态
Reserved	[31:4]		0
DATARAM1	[3:0]	ram1 设置数据的非扇区部分。默认值为 0x3。在初始化期间通过软件设置	3

7.7.20. 同步模式寄存器

寄存器	地址	读/写	描述	复位值
SYNC_MODE0	0x70100130	读	Bank0 同步模式寄存器	由存储器决定的
SYNC_MODE1	0x70180130			

SYNC_MODEn	位	描述	初始状态
Reserved	[31:1]		0
RM	[1]	为读操作设置同步或异步传输模式。默然值为0x0。在初始化期间通过软件设置。该值从MEM_CFG寄存器[15]拷贝。只读  • 0 = 异步模式 • 1 = 激活同步模式	0
WM	[0]	为写操作设置同步或异步传输模式。默认值为0x0。在初始化期间通过软件设置。该值从MEM_CFG寄存器[1]拷贝。只读  • 0 = 异步模式 • 1 = 激活同步模式	0

7.7.21. 传输备用寄存器

寄存器	地址	读/写	描述	复位值
TRANS_SPARE0	0x70100140	读/写	Bank0 传输大小寄存器	0x0000
TRANS_SPARE1	0x70180140			

TRANS_SPAREn	位	描述	初始状态
Reserved	[31:1]		0
TSRF	[0]	通过映射01，涉及到所有的读写指令。如果该位	0

		被设置, 存储器备用区的数据将被存储控制器异步FIFO。备用区的大小依赖扇区数 <ul style="list-style-type: none"> <li>• 0 =只传输数据</li> <li>• 1 = 增加扇区大小</li> </ul> 页的主要数据区首先传输, 然后是备用区	
--	--	--	--

### 7.7.22. DBS-DFS 宽度寄存器

寄存器	地址	读/写	描述	复位值
DBS_DFS_WIDTH0	0x70100160	读/写	Bank0 页 DBS_DFS 宽度	由存储器决定
DBS_DFS_WIDTH1	0x70180160		寄存器	的

DBS_DFS_WIDTHn	位	描述	初始状态
Reserved	[31:2]		0
WIDTH	[1:0]	设置 DBS 和 DFS 宽度。在初始化期间通过软件设置。如果没关系则忽略	0

### 7.7.23. 页计数寄存器

寄存器	地址	读/写	描述	复位值
PAGE_CNT0	0x70100170	读	Bank0 页计数寄存器	0x0000
PAGE_CNT1	0x70180170			

PAGE_CNTn	位	描述	初始状态
Reserved	[31:8]		0
PAGE_COUNT	[7:0]	通过当前执行的多页回拷贝指令保存回拷贝页数。只读	0

7.7.24. 错误页地址寄存器

寄存器	地址	读/写	描述	复位值
ERR_PAGE_ADDR0	0x70100180	读	Bank0 错误页地址寄存器	0x0000
ERR_PAGE_ADDR1	0x70180180			

ERR_PAGE_ADDRn	位	描述	初始状态
Reserved	[31:6]		0
FAIL_PAGE_ADDR	[5:0]	执行后，加载或者擦除错误中断，该寄存器将保存没有成功操作的页的地址。只读	0

7.7.25. 脉冲读出等待寄存器

寄存器	地址	读/写	描述	复位值
BURST_RD_LAT0	0x70100190	读	Bank0 脉冲读出延迟寄存器	由存储器决定的
BURST_RD_LAT1	0x70180190			

BURST_RD_LATn	位	描述	初始状态
Reserved	[31:3]		0
BURST_RD_LAT	[2:0]	在周期中设置脉冲读出。默认值是 0x6。该值从 MEM_CFG[14:12]复制。只读	寄存器

7.7.26. 中断引脚使能寄存器

寄存器	地址	读/写	描述	复位值
INT_PIN_ENABLE0	0x701001A0	读/写	Bank0 中断引脚使能寄存器。	0x0000
INT_PIN_ENABLE1	0x701801A0			

INT_PIN_ENABLEn	位	描述	初始状态
Reserved	[31:1]		0
INT	[0]	<p>中断引脚有效。</p> <p>如果需要使用中断引脚或者控制器要从状态寄存器查看中断信息，该位有效。初始化期间通过软件设置。</p> <ul style="list-style-type: none"> <li>• 0 = 使用状态寄存器</li> <li>• 1 = 使用中断引脚</li> </ul>	0

### 7.7.27. 中断监控周期寄存器

寄存器	地址	读/写	描述	复位值
INT_MON_CYC0	0x701001B0	读/写	Bank0 中断监控寄存器	0x01F4
INT_MON_CYC1	0x701801B0			

INT_MON_CYCn	位	描述	初始状态
Reserved	[31:12]	保留。	0
INT_MON_CYC	[11:0]	在 INT_ERR_STA 寄存器检测和存储状态寄存器之间设置周期值。如果闪存配置寄存器位 IOBE 被清除，则使用该寄存器	500

### 7.7.28. 存取时钟寄存器

寄存器	地址	读/写	描述	复位值
ACC_CLOCK0	0x701001C0	读/写	Bank0 存取时钟寄存器	0x0003
ACC_CLOCK1	0x701001C1			

ACC_CLOCKn	位	描述	初始状态
Reserved	[31:3]	保留。	0
ACCESS_CLK	[2:0]	设置周期值，要求来完成闪存设备存取的时间。遵循下面的公式： (35ns/时钟周期+1)  系 统 时 钟      闪存时钟（MHz）      访问时钟 （MHz）  166              83                      3 134              67                      3 100              50                      2 60                30                      2	3

7.7.29.    慢读路径寄存器

寄存器	地址	读/写	描述	复位值
SLOW_RD_PATH0	0x701001D0	读/写	Bank0 缓慢读通道寄存器	0x0000
SLOW_RD_PATH1	0x701801D0			

SLOW_RD_PATHn	位	描述	初始状态
Reserved	[31:1]	保留	0
SRP	[0]	默认值是 0x0	0

7.7.30.    错误块地址寄存器

寄存器	地址	读/写	描述	复位值
ERR_BLK_ADDR0	0x701001E0	读	Bank0 错误块地址寄存器	0x0000
ERR_BLK_ADDR1	0x701801E0			



ERR_BLK_ADDRn	位	描述	初始状态
Reserved	[31:12]		0
FAIL_BLK_ADDR	[11:0]	执行后，载入或者擦除中断，寄存器将保存操作失败的时钟地址。只读	0

### 7.7.31. 闪存版本 ID 寄存器

寄存器	地址	读/写	描述	复位值
FLASH_VER_ID0	0x701001F0	读	Bank0 闪存版本 ID 寄存器	由存储器决定的
FLASH_VER_ID1	0x701801F0			

FLASH_VER_IDn	位	描述	初始状态
Reserved	[31:16]	保留	
FLASH_VER	[15:0]	该值依赖实际使用的存储器设备。复位后，该寄存器被闪存控制器设置。只读	

## 7.8 ONENAND 控制器应用举例

下面是几段实现 ONENAND 部分功能的程序，通过对程序进行分析，有助于对该控制器的理解。

### 7.8.1 ONENAND 初始化程序

```
//输出为:TRUE – 存储器设备复位; FALSE: 由于错误, 存储器设备不复位
bool ONENAND_Init(u32 Controller)
{
    u32 uBaseAddress;
    // Controller — OneNand 控制器端口数
```

```

if(Controller == 0)
{
    uBaseAddress = ONENAND0_BASE;
}
else if(Controller == 1)
{
    uBaseAddress = ONENAND1_BASE;
}
else
{
    return FALSE;
}

```

```

ONENAND_pBase[Controller] = (void *)uBaseAddress;

```

```

#if (ONENAND_VERSION == ONENAND_EVT0)
    //存储器端口 0 驱动激增(OneNand 控制信号)

```

```

    GPIO_SetMem0DrvStrength(0x55000411);

```

```

#endif

```

```

ONENAND_SetFlashClock(Controller, eDiv2_HCLKx2);

```

```

ONENAND_SetAccClock(Controller);

```

```

ONENAND_GetDevInformation(Controller);

```

```

ONENAND_SetMemSpace(Controller);

```

```

ONENAND_EnableAllInterrupt(Controller);

```

```

ONENAND_DisableInterrupt(Controller, eOND_RDYACT);

```

```

    ONENAND_EnableECCCorrection(Controller, ONENAND_WITH_CORRECT);
    ONENAND_EnableIOBE(Controller, 1);
    ONENAND_SetBurstLatency(Controller, eOND_LATENCY4);
    ONENAND_SetSyncMode(Controller, eOND_SYNC_BURST16);

    //ONENAND_EnableAllInterrupt(Controller);
    //ONENAND_DisableInterrupt(Controller, eOND_RDYACT);

#if (OND_TRANS_MODE == OND_DMA)
    DMAC_InitCh(DMA0, DMA_ALL, &g_oONDDmac0);
    INTC_SetVectAddr(NUM_DMA0, ONENAND_DmaISR);
    INTC_Enable(NUM_DMA0);
#endif

    return TRUE;
}

```

## 7.8.2.设置脉冲读取延迟

```

//eLatency - (OneNAND_eLATENCY)等待周期
void ONENAND_SetBurstLatency(u32 Controller, OneNAND_eLATENCY eLatency)
{
    u32 uBurstReadLatency;

    uBurstReadLatency = Inp32(&ONENAND(Controller)->rMemCfg);
    uBurstReadLatency &= ~(0x7<<12);

    uBurstReadLatency |= (eLatency<<12);
}

```

```

        Outp32(&ONENAND(Controller)->rMemCfg, uBurstReadLatency);

    }

```

### 7.8.3. 设置 ONENAND 内部闪存时钟

```

void ONENAND_SetFlashClock(u32 Controller, OneNAND_eFlashClock eFlashClock)
{
    u32 uClkDivider;

    uClkDivider = SYSC_GetDIV0();
    uClkDivider = (uClkDivider & ~(3<<16)) | (eFlashClock<<16);

    SYSC_SetDIV0_all(uClkDivider);
}

```

### 7.8.4. 设置周期数

该段程序是用来设置覆盖闪存设备访问时间的周期数。

```

void ONENAND_SetAccClock(u32 Controller)
{
    u32 uClkDivider, uAccClock;

    uClkDivider = SYSC_GetDIV0();
    uClkDivider = (uClkDivider & (3<<16))>>16;

    OneNand_Inform[Controller].uFlashClock = g_HCLKx2/(uClkDivider+1);
}

```

```

    uAccClock = (35*(OneNand_Inform[Controller].uFlashClock/1000))/1000000 + 1;
    Outp32(&ONENAND(Controller)->rAccClock, uAccClock&0x7);
}

```

### 7.8.5.解锁存储器模块

```

bool ONENAND_UnlockBlock(u32 Controller, u32 uStartBlkAddr, u32 uEndBlkAddr)
{
    //u32 uData;

    OneNandT_oIntFlag.IntActInt = 0;
    //uStartBlkAddr –解锁的开始块， uEndBlkAddr – 解锁的结尾块
    if(uStartBlkAddr == uEndBlkAddr)
    {
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x09);
    }
    else
    {
        ONENAND_WriteCmd(Controller, uStartBlkAddr, 0, 0x08);
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x09);
    }

    //中断
    while(!OneNandT_oIntFlag.IntActInt);

    #if 0
        ONENAND_DirectRead(Controller, OND_WPROT_STATUS, &uData);

        if ((uData&0xFFFF) != 0x4)

```

```

    {
        return TRUE;
    }
#endif

    return FALSE;
}

```

### 7.8.6.锁定模块

```

bool ONENAND_LockBlock(u32 Controller, u32 uStartBlkAddr, u32 uEndBlkAddr)
{
    //u32 uData;

    OneNandT_oIntFlag.IntActInt = 0;

    if(uStartBlkAddr == uEndBlkAddr)
    {
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x0B);
    }
    else
    {
        ONENAND_WriteCmd(Controller, uStartBlkAddr, 0, 0x0A);
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x0B);
    }

    //中断

```

```
while(!OneNandT_oIntFlag.IntActInt);

#if 0
    ONENAND_DirectRead(Controller, OND_WPROT_STATUS, &uData);

    if ((uData&0xFFFF) != 0x2)
    {
        //UART_Printf(" Failed lock block, locked status (0x%x)\n", uData);
        return TRUE;
    }
#endif

    return FALSE;
}
```

## 8 NAND FLASH 控制器

目前的 NOR FLASH 存储器价格比较昂贵，而 SDRAM 和 NAND FLASH 存储器的价格相对来说比较合适，这就是为什么消费者更喜欢从 NAND FLASH 启动引导系统，而在 SDRAM 上执行主程序代码的原因。

S3C6410 恰好满足这一要求，它可以实现从 NAND FLASH 上执行引导程序。S3C6410 具备了一个内部 SRAM 缓冲器，叫做“STEPPINGSTONE”，支持 NAND FLASH 的系统引导。当系统启动时，NAND FLASH 存储器的前面 4KB 将被自动载入到 STEPPINGSTONE 中，然后系统自动执行这些载入的引导代码。

通常情况下，这 4K 的引导代码需要将 NAND FLASH 中程序内容拷贝到 SDRAM 中，在引导代码执行完毕后跳转到 SDRAM 执行。使用 S3C6410 内部硬件 ECC 功能可以对 NAND FLASH 的数据进行有效性的检测。

### 8.1 NAND FLASH 控制器的特性

NAND FLASH 控制器的特性如下：

（1）自动导入模式：复位后，引导代码被送入 4KB 的 STEPPINGSTONE 中，引导代码移动完毕，引导代码将在 STEPPINGSTONE 中执行。

注：在导入期间，NAND FLASH 控制器不支持 ECC 纠正。

（2）NAND FLASH 控制器 I/F：支持 512 字节和 2KB 页。

（3）软件模式：用户可以直接访问 NAND FLASH 控制器。例如这个特性可以用于读/擦/编程 NAND FLASH 存储器。

（4）接口：8 位 NAND FLASH 存储器接口总线。

（5）硬件 ECC 产生、检测和标志（软件纠正）。

（6）支持 SLC 和 MLC 的 NAND FLASH 控制器：1 位 ECC 用于 SLC，4 位 ECC 用于 MLC 的 NAND FLASH。

（7）特殊功能寄存器 I/F：支持字节/半字/字数据的访问和 ECC 的数据寄存器，用字来访问其他寄存器。

（8）STEPPINGSTONE I/F：支持字节/半字/字的访问。

（9）4KB 内部 SRAM 缓冲器 STEPPINGSTONE，在 NAND FLASH 引导后可以作为其他用途使用。



NAND FLASH 控制器的结构，如图 8-1 所示。

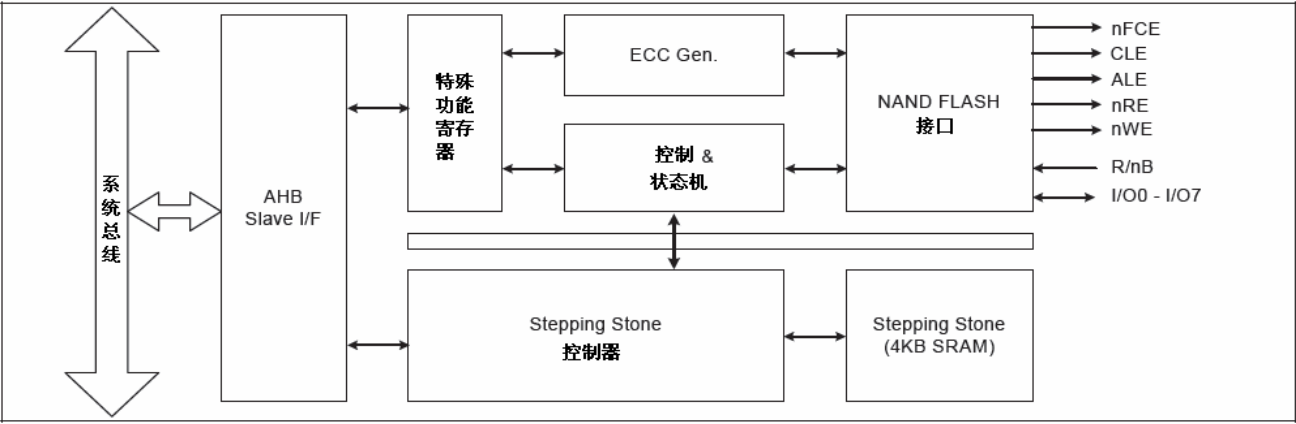


图 8-1 NAND FLASH 控制器结构图

1. NAND FLASH 控制器工作机制

NAND FLASH 控制器的工作机制，如图 8-2 所示。在上电复位时，NAND FLASH 控制器将通过 XOM（参照引脚配置）引脚状态来获得关于连接 NAND FLASH 的信息。在上电或系统复位之后，NAND FLASH 控制器自动加载 4KB 的启动代码。加载完成后，启动代码将在 STEPPINGSTONE 中被执行。

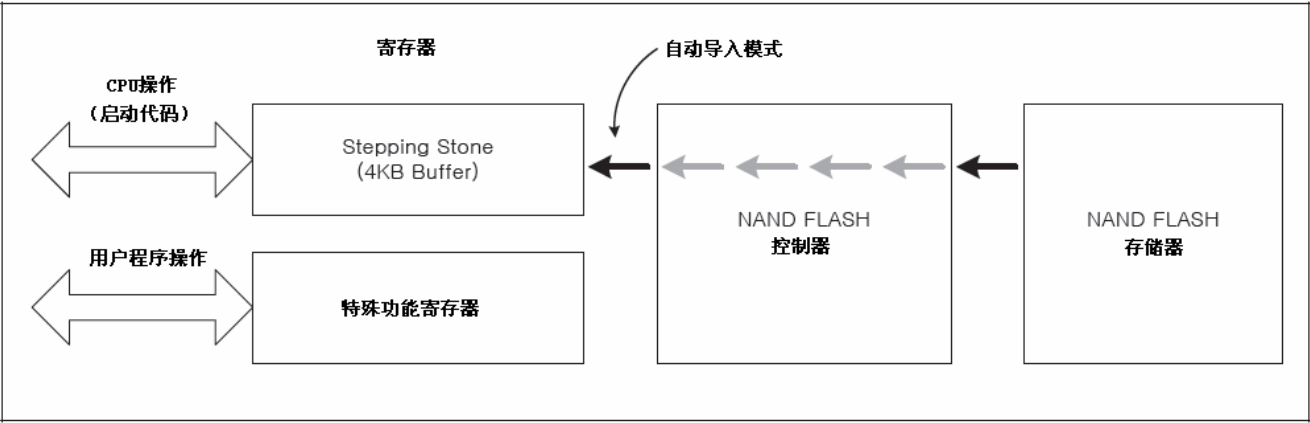


图 8-2 NAND FLASH 控制器的工作机制

注：在自动导入期间，ECC 是未被选中状态。因此，前 4KB 的 NAND FLASH 绝不能有位错误。

## 2. 引脚配置

以下是相应的引脚配置如表 8-1 所示。

表 8-1 引脚配置

OM[4:0]	Adv 闪存	页大小	地址周期	总线宽度
0000x	0: Normal NAND	1: 512 字节	0: 3 周期	0: 8 位数据总线
0001x	0: Normal NAND	1: 512 字节	1: 4 周期	0: 8 位数据总线
0010x	1: Advance NAND	1: 2K 字节	0: 4 周期	0: 8 位数据总线
0011x	1: Advance NAND	1: 2K 字节	1: 5 周期	0: 8 位数据总线

以上的引脚配置可适用于 NAND FLASH 被用作启动存储器的情况，如果 NAND FLASH 不能用作启动存储器，引脚的配置可以通过设置 NFCON SFR 'NFCNF' (0x70200000) 来改变。

## 3. NAND FLASH 存储器时序

NAND FLASH 存储器时序，CLE 和 ALE 时序如图 8-3 所示，nWE 和 nRE 时序如图 8-4 所示。

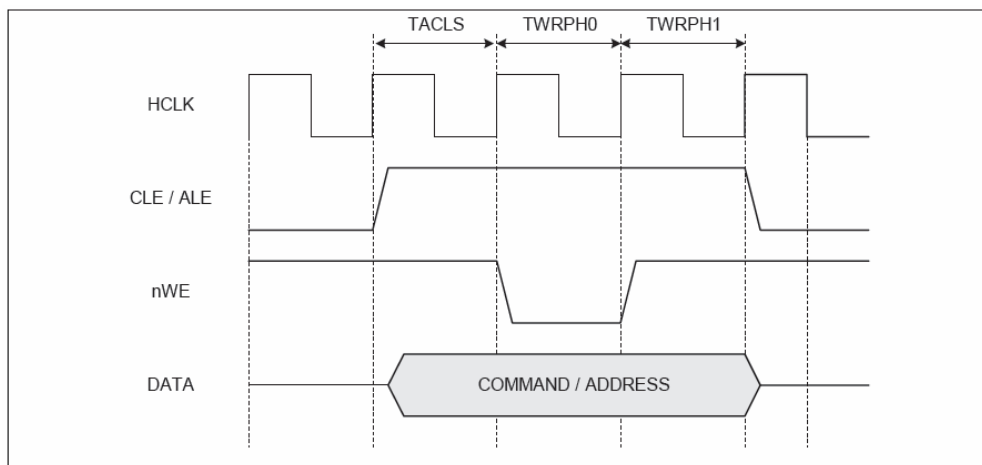


图 8-3 CLE 和 ALE 时序 (TACLS=1, TWRPH0=0, TWRPH1=0)

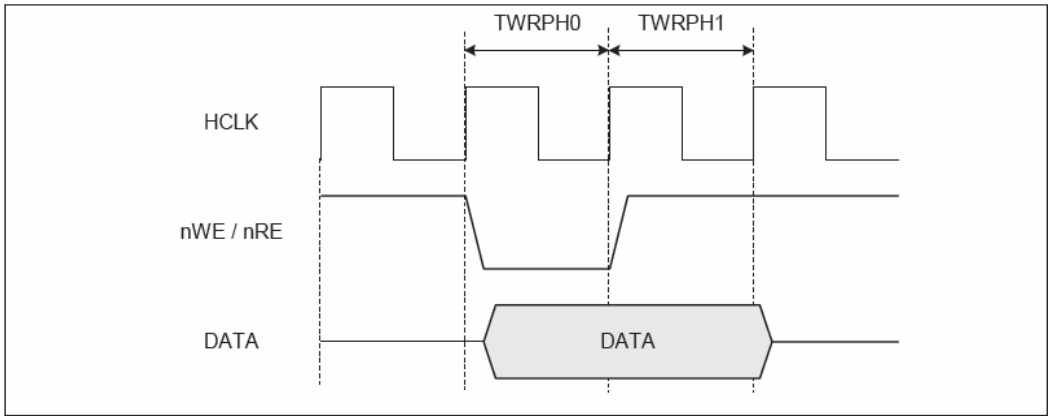


图 8-4 nWE 和 nRE 时序 (TWRPH0=0, TWRPH1=0)

#### 4. 软件模式

S3C6410 仅支持软件模式访问。使用这种模式完全地进入 NAND FLASH 存储器。NAND FLASH 控制器支持直接访问 NAND FLASH 存储器接口。

- (1) 写命令寄存器=NAND FLASH 存储器命令周期。
- (2) 写地址寄存器=NAND FLASH 存储器地址周期。
- (3) 写数据寄存器=写数据到 NAND FLASH 存储器（写周期）。
- (4) 读数据寄存器=从 NAND FLASH 存储器读数据（读周期）。
- (5) 读主 ECC 寄存器和备用 ECC 寄存器=从 NAND FLASH 存储器读数据。

注：在软件模式下，必须通过利用检测和中断来检查 RnB 输入引脚的状态。

#### 5. 数据寄存器配置

8 位 NAND FLASH 存储器接口。

##### A. 字访问

NAND FLASH 存储器接口的字访问，如表 8-2 所示。

表 8-2 字访问描述

寄存器	端	位[31:24]	位[23:16]	位[15:8]	位[7:0]
NFDATA	小端	4 <sup>th</sup> I/O[ 7:0]	3 <sup>rd</sup> I/O[ 7:0]	2 <sup>nd</sup> I/O[ 7:0]	1 <sup>st</sup> I/O[ 7:0]

B. 半字访问

NAND FLASH 存储器接口的半字访问，如表 8-3 所示。

表 8-3 半字访问描述

寄存器	端	位[31:24]	位[23:16]	位[15:8]	位[7:0]
NFDATA	小端	无效值	无效值	2 <sup>nd</sup> I/O[ 7:0]	1 <sup>st</sup> I/O[ 7:0]

C. 字节访问

NAND FLASH 存储器接口的字节访问，如表 8-4 所示。

表 8-4 字节访问描述

寄存器	端	位[31:24]	位[23:16]	位[15:8]	位[7:0]
NFDATA	小端	无效值	无效值	无效值	1 <sup>st</sup> I/O[ 7:0]

8.2 STEPPINGSTONE (4KB SRAM)

NAND FLASH 控制器使用 Steppingstone 作为缓冲器引导，也可以使用它进行各种其他用途。

1.SLC / MLC ECC（错误纠正码）

NAND FLASH 控制器有 4 个 ECC（错误纠正码）模块用于 SLC 类型的 NAND FLASH 存储器，1 个 ECC 模块用于 MLC 类型的 NAND FLASH 存储器。

SLC 类型的 NAND FLASH 存储器接口，NAND FLASH 控制器组成 4 个 ECC 模块。这些模块可用于（高达）2048 字节 ECC 奇偶校验码的产生，以及其他可用于（高达）4 字节 ECC 奇偶校验码的产生。

MLC 类型的 NAND FLASH 存储器接口，NAND FLASH 控制器组成 1 个 ECC 模块。这些模块仅用于 512 字节的 ECC 奇偶校验码的产生。8 位存储器接口，MLC 的 ECC 模块每 512 字节生成奇偶校验码。然而，SLC 的 ECC 模块生成奇偶校验码是将每个字节分开。

以下是 ECC 奇偶校验码和两个 SLC 的 ECC 模块表格：

- 28 位 ECC 奇偶校验码=22 位行奇偶校验+6 位列奇偶校验
- 10 位 ECC 奇偶校验码=4 位行奇偶校验+6 位列奇偶校验

(1) 2048 字节 SLC 的 ECC 奇偶校验码分配表，如表 8-5 所示。

表 8-5 SLC 的 ECC 奇偶校验码分配表（2048 字节）

	DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
MECCn_0	~P64	~P64'	~P32	~P32'	~P16	~P16'	~P8	~P8'
MECCn_1	~	~P1024'	~P512	~	~P256	~P256'	~P128	~P128'
MECCn_2	P1024			P512'				
	~P4	~P4'	~P2	~P2'	~P1	~P1'	~P2048	~P2048'
MECCn_3	1	1	1	1	~P8192	~P8192'	~P4096	~P4096'

(2) 4 字节 SLC 的 ECC 奇偶校验码分配表，如表 8-6 所示。

表 8-6 SLC 的 ECC 奇偶校验码分配表（4 字节）

SECCn_0	DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
SECCn_1	~P2	~P2'	~P1	~P1'	~P16	~P16'	~P8	~P8'
	1	1	1	1	1	1	~P4	~P4'

下面是 NAND FLASH 控制器 ECC 错误校验的一个操作实例，其具体代码实现如下：

```
// NAND_CheckECCError 函数：主要功能实现 ECC 错误校验。
// 输入：Controller - Nand 控制器端口数
// 输出：ECC 错误类型
NAND_eERROR NAND_CheckECCError(u32 Controller)
{
    u32 uEccError0;
    //u32 uEccError1;
    NAND_eERROR eError;
    eError = eNAND_NoError;
    if((NAND_Inform[Controller].uNandType == NAND_Normal8bit) ||
        (NAND_Inform[Controller].uNandType == NAND_Advanced8bit) )
    {
        uEccError0 = Inp32(&NAND(Controller)->rNFECCECERR0);
        switch (uEccError0 & 0x03)
```

```

{
    case 0x00 :    eError = eNAND_NoError;
                  break;
    case 0x01 :    eError = eNAND_1bitEccError;
                  break;
    case 0x02 :    eError = eNAND_MultiError;
                  break;
    case 0x03 :    eError = eNAND_EccAreaError;
                  break;
}

if(NAND_Inform[Controller].uPerformanceCheck == 0)
{
    switch ((uEccError0 & 0x0C)>>2)
    {
        case 0x00 :    eError |= eNAND_NoError;
                      break;
        case 0x01 :    eError |= eNAND_Spare1bitEccError;
                      break;
        case 0x02 :    eError |= eNAND_SpareMultiError;
                      break;
        case 0x03 :    eError |= eNAND_SpareEccAreaError;
                      break;
    }
}

else if(NAND_Inform[Controller].uNandType == NAND_MLC8bit)
{
    uEccError0 = Inp32(&NAND(Controller)->rNFECCERR0);
}

```

```

//uEccError1 = Inp32(&NAND(Controller)->rNFECCERR1);
switch ((uEccError0>>26) & 0x07)
{
    case 0x00 :    eError = eNAND_NoError;
                  break;
    case 0x01 :    eError = eNAND_1bitEccError;
                  break;
    case 0x02 :    eError = eNAND_2bitEccError;
                  break;
    case 0x03 :    eError = eNAND_3bitEccError;
                  break;
    case 0x04 :    eError = eNAND_4bitEccError;
                  break;
    case 0x05 :    eError = eNAND_UncorrectableError;
                  break;
}
}
return eError;
}

```

## 1. ECC 模块特性

ECC 的产生是通过 ECC 锁定 (MainECCLock, SpareECCLock) 位的控制寄存器来控制的。当 ECCLock 为低时, ECC 校验码通过 H/W ECC 模块产生。

## 2. SLC ECC 寄存器配置

以下各表显示 SLC ECC 从外部 NAND FLASH 存储器的备用区中读取值的配置。比较 ECC 奇偶校验码是通过 H/W 模块来产生的, 从存储器读取 ECC 的格式是非常重要的。

注: MLC ECC 译码方式不同于 SLC ECC。

8 位 NAND FLASH 存储器接口, 如表 8-7 所示。

表 8-7 8 位 NAND FLASH 存储器接口

寄存器	位[31:24]	位[23:16]	位[15:8]	位[7:0]
NFMECCD0	4 <sup>th</sup> ECC for I/O[7:0]	3 <sup>rd</sup> ECC for I/O[7:0]	2 <sup>nd</sup> ECC for I/O[7:0]	1 <sup>st</sup> ECC for I/O[7:0]
NFMECCD1	没有使用			

寄存器	位[31:24]	位[23:16]	位[15:8]	位[7:0]
NFSECCD	没有使用	2 <sup>nd</sup> ECC 用于 I/O[7:0]	1 <sup>st</sup> ECC 用于 I/O[7:0]	

3. SLC ECC 设计向导

(1) 在使用 SLC ECC 软件模式时，复位 ECCType 为 ‘0’ (SLC ECC 使能)。当 MainECCLock (NFCON[7]) 和 SpareECCLock (NFCON[6]) 开启 (‘0’) 时，ECC 模块生成的 ECC 奇偶校验码用于所有数据的读/写。在读数据或写数据之前，必须复位 ECC 的值，方法是把 InitMECC (NFCONT[5]) 和 InitSECC (NFCON[4]) 的位设为 ‘1’，同时把 MainECCLock (NFCONT[7]) 的位设为 ‘0’ (开启)。

ECC 奇偶校验码的产生与否主要是通过 MainECCLock (NFCONT[7]) 和 SpareECCLock (NFCONT[6]) 位来控制的。

(2) 无论是读数据还是写数据，ECC 模块都产生 ECC 奇偶校验码来记录 NFMECC0/1。

(3) 在完成读或写页 (不包括备用区数据) 后，设置 MainECCLock 位为 ‘1’ (锁定)。ECC 奇偶校验码是锁定的，同时 ECC 状态寄存器的值将不会被改变。

(4) 要产生备用区 ECC 奇偶校验码，SpareECCLock (NFCONT[6]) 位需要清零 ‘0’ (开启)。

(5) 无论是读数据还是写数据，备用区 ECC 模块都产生 ECC 奇偶校验码来记录 NFSECC。

(6) 在完成读或写备用区后，设置 SpareECCLock 位为 ‘1’ (锁定)。ECC 奇偶校验码是锁定的，同时 ECC 状态寄存器的值将不会被改变。

(7) 可以使用这些值来记录备用区或检查误码。

(8) 例如, 检查误码的主要数据区在页的读操作，在主数据区产生 ECC 校验码后，必须移动 ECC 奇偶校验码 (存储到备用区) 到 NFMECCD0 和 NFMECCD1 寄存器中。这时，NFECERR0 和 NFECERR1 将产生有效的错误状态值。

注：NFSECCD 是 ECC 备用区 (通常情况下，用户将产生的 ECC 值从主数据区写入到备用区中，其值与



NFMECC0/1 一样), 它是从主数据区产生的。

#### 4. MLC ECC 设计向导 (编码)

(1) 在使用 MLC ECC 软件模式时, 设置 MsgLength 为 '0' (512 字节信息长度), 同时设置 ECCType 为 '0' (使能 MLC ECC)。ECC 模块生成的 ECC 奇偶校验码用于 512 字节写数据。所以, 必须复位 ECC 的值, 方法是在写数据之前, 把 InitMECC (NFCNT[5]) 的位写 '1', 同时把 MainECCLock (NFCNT[7]) 的位清零 '0' (开启)。

ECC 奇偶校验码的产生与否主要是通过 MainECCLock (NFCNT[7]) 位来控制的。

(2) 无论何时进行写数据, MLC ECC 模块都会在内部产生 ECC 奇偶校验码。

(3) 在完成 512 字节的写数据后 (不包括备用区数据), 奇偶校验码将自动更新 NFMECC0, NFMECC1 寄存器。如果使用 512 字节的 NAND FLASH 存储器, 可以编程序将这些值到备用区。然而, 如果使用 NAND FLASH 存储器超过 512 字节页以上, 这时将不能直接编程。在这种情况下, 必须复制这些奇偶校验码到其他的存储器中, 比如 DRAM。之后写入所有主数据, 可以将写入 ECC 的值复制到备用区。奇偶校验码有自我纠正信息, 包括它本身。

(4) 要产生备用区 ECC 奇偶校验码, 需要设置 MsgLength 为 1 (24 字节信息长度), 同时设置 ECCType 为 '1' (使能 MLC ECC)。ECC 模块生成的 ECC 奇偶校验码用于 24 字节写数据。所以, 你必须复位 ECC 的值, 方法是在写数据之前, 把 InitMECC (NFCNT[5]) 的位写 '1', 同时把 MainECCLock (NFCNT[7]) 的位清零 '0' (开启)。

ECC 奇偶校验码的产生与否主要是通过 MainECCLock (NFCNT[7]) 位来控制的。

(5) 无论何时进行写数据, MLC ECC 模块都会在内部产生 ECC 奇偶校验码。

(6) 在完成 24 字节元数据或额外数据写入后, 奇偶校验码将自动更新 NFMECC0, NFMECC1 寄存器。可以编程将这些奇偶校验码移到备用区。奇偶校验码有自我纠正信息, 包括它本身。

#### 5. MLC ECC 设计向导 (译码)

(1) 在四个 512 字节主区域中, 备用区是由四个 7 字节的奇偶区域, 24 字节的元数据以及 7 字节用于这个元数据的奇偶区组成的。

(2) 在使用 MLC ECC 软件模式时, 设置 MsgLength 为 0 (512 字节信息长度), 同时设置 ECCType 为 '1' (使能 MLC ECC)。ECC 模块生成的 ECC 奇偶校验码用于 512 字节读数据。所以, 你必须复位 ECC 的值, 方法是在读数据之前, 把 InitMECC (NFCNT[5]) 的位写 '1', 同时把 MainECCLock (NFCNT[7]) 的位

清零 ‘0’（开启）。

ECC 奇偶校验码的产生与否主要是通过 MainECCLock (NFCONT[7]) 和 SpareECCLock(NFCONT[6])位来控制的。

(3) 无论何时进行写数据，MLC ECC 模块都会在内部产生 ECC 奇偶校验码。

(4) 在完成 512 字节读数据（不包括备用区数据）后，需要读奇偶校验码。MLC ECC 模块需要奇偶校验码检测是否有错误位。所以在读 512 字节后，读 ECC 奇偶校验码是有必要的。一旦 ECC 奇偶校验码被读，MLC ECC 引擎开始在内部搜索任何错误。MLC ECC 的错误搜索引擎可以在最小为 155 周期内找到任何错误。在这段时间内，可以继续从外部 NAND FLASH 存储器中读主数据。ECCDecDone(NFSTAT[6])可以用于检测 ECC 译码是否被完成。

(5) 当 ECCDecDone (NFSTAT[6])设置为 ‘1’ 时，NFECERR0 显示是否有错误位存在。如果有任何一种错误存在的话，则可以通过参照 NFECERR0/1 和 NFMLCBITPT 寄存器来进行修正。

(6) 如果有很多主数据要读，继续第 2 步。

(7) 元数据错误检测，设置 MsgLength 为 1（24 字节信息长度），同时设置 ECCType 为 ‘1’（使能 MLC ECC）。ECC 模块生成的 ECC 奇偶校验码用于 24 字节读数据。所以，必须复位 ECC 的值，方法是在读数据之前，把 InitMECC (NFCONT[5])的位写 ‘1’，同时把 MainECCLock (NFCONT[7])的位清零 ‘0’（开启）。

ECC 奇偶校验码的产生与否主要是通过 MainECCLock (NFCONT[7])位来控制的。

(8) 无论何时进行写数据，MLC ECC 模块都会在内部产生 ECC 奇偶校验码。

(9) 在完成 24 字节读数据后，需要读奇偶校验码。MLC ECC 模块需要奇偶校验码检测是否有错误位。所以在读 24 字节后，读 ECC 奇偶校验码是有必要的。一旦 ECC 奇偶校验码被读，MLC ECC 引擎开始在内部搜索任何错误。MLC ECC 的错误搜索引擎可以在最小为 155 周期内找到任何错误。在这段时间内，可以继续从外部 NAND FLASH 存储器中读主数据。ECCDecDone(NFSTAT[6])可以用于检测 ECC 译码是否被完成。

(10) 当 ECCDecDone (NFSTAT[6])设置为 ‘1’ 时，NFECERR0 显示是否有错误位存在。如果有任何一种错误存在的话，则可以通过参照 NFECERR0/1 和 NFMLCBITPT 寄存器来进行修正。

## 6. NAND FLASH 存储器映射

关于 NAND FLASH 存储器的映射，可以参看图 8-5 所示。

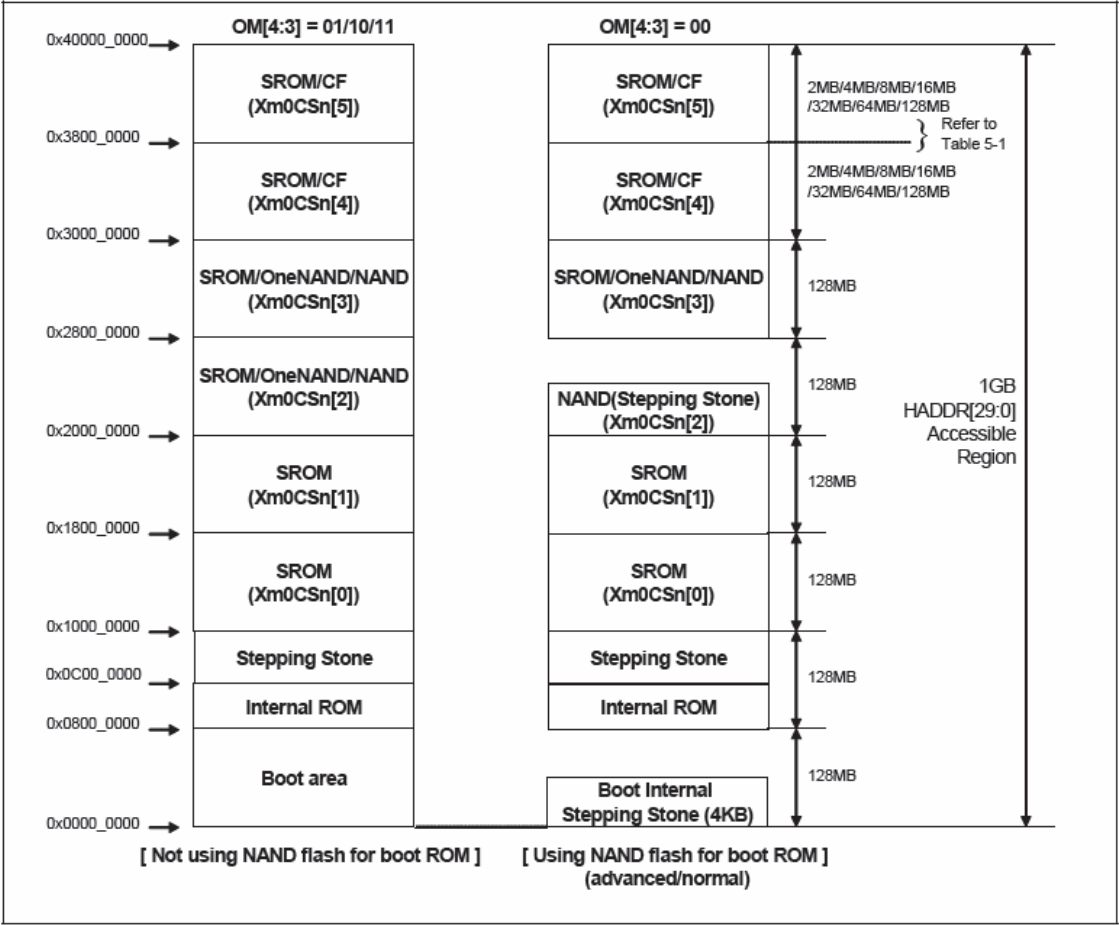


图 8-5 NAND FLASH 存储器映射结构图

7. NAND FLASH 存储器结构

以下是 NAND FLASH 存储器的结构图，如图 8-6 所示。

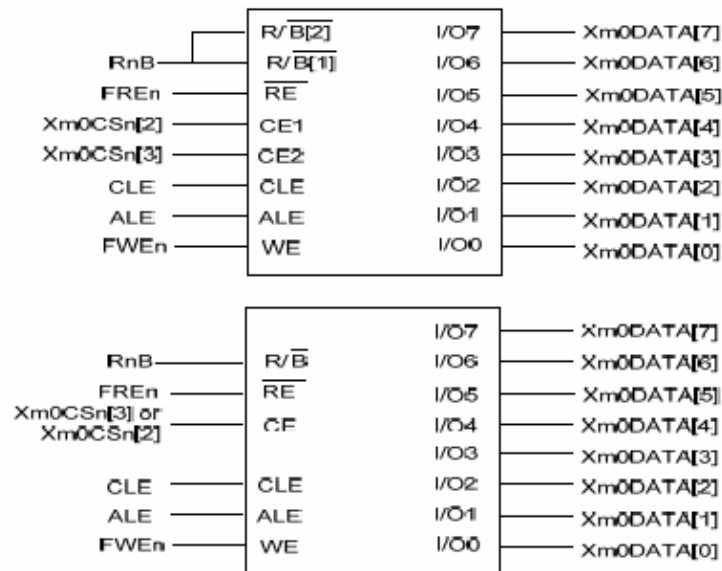


图 8-6 一个 8 位 NAND FLASH 存储器接口框图

注：NAND 控制器可以支持控制两个 NAND FLASH 存储器。

如表 8-8 所示，显示了 NAND FLASH 的两个控制器。

表 8-8 两个 NAND FLASH 控制器

	NAND	BOOT
Xm0CSn[2]	NAND 控制器 CS0	可配置
Xm0CSn[3]	NAND 控制器 CS1	可配置

如果想要从 NAND 启动，则必须使用 Xm0CSn[2]进行导入。

## 8.3 NAND FLASH 特殊控制寄存器

下面主要针对 NAND FLASH 控制器中的寄存器做详细的介绍，能够让读者对其中的每个寄存器的功能有所了解。

### 8.3.1 NAND FLASH 控制寄存器列表

NAND FLASH 控制寄存器中的不同类别寄存器定义，如表 8-9 所示。

表 8-9 NAND FLASH 控制寄存器列表

地址	读/写	复位值	名称	描述
Base + 0x00	读/写	0x0000100X	NFCNF	配置寄存器
Base + 0x04	读/写	0x000100C6	NFCNT	控制寄存器
Base + 0x08	读/写	0x00	NFCMD	命令寄存器
Base + 0x0c	读/写	0x0000XX00	NFADDR	地址寄存器
Base + 0x10	读/写	0xXXXX	NFDATA	数据寄存器
Base + 0x14	读/写	0x00000000	NFMECCD0	第一个和第二个主 ECC 数据寄存器
Base + 0x18	读/写	0x00000000	NFMECCD1	第三个和第四个主 ECC 数据寄存器
Base + 0x1c	读/写	0x00000000	NFSECCD	备用 ECC 读寄存器
Base + 0x20	读/写	0x000000	NFSBLK	可编程开始块地址寄存器
Base + 0x24	读/写	0x000000	NFEBLK	可编程结束块地址寄存器
Base + 0x28	读/写	0x0080001D	NFSTAT	NAND 状态寄存器
Base + 0x2C	读	0x007FFFFA	NFECCERR0	ECC 错误状态 0 寄存器
Base + 0x30	读	0x007FFFFA	NFECCERR1	ECC 错误状态 1 寄存器
Base + 0x34	读	0xXXXXXX	NFMECC0	生成 ECC 状态 0 寄存器
Base + 0x38	读	0xXXXXXX	NFMECC1	生成 ECC 状态 1 寄存器
Base + 0x3C	读	0xXXXXXX	NFSECC	生成备用区 ECC 状态寄存器
Base + 0x40	读	0x00000000	NFMLCBITPT	4 位 ECC 错误位模式寄存器
* Base = 0x7020_0000				

### 8.3.2. NAND FLASH 配置寄存器

寄存器	地址	读/写	描述	复位值
NFCONF	0x70200000	读/写	NAND FLASH 配置寄存器	0x0000100X

NFCONF	位	描述	初始状态
NANDBoot	[31]	只读。显示是否使用 NAND 启动 1=NAND Flash 存储器启动	0
ECCClkCon	[30]	时钟控制 4 位 ECC 引擎 0: 当系统时钟大于 66MHz 时, 推荐使用 1: 当系统时钟小于 66MHz 时, 推荐使用	0
Reserved	[29:26]	保留	0000
MsgLength	[25]	信息 (数据) 长度为 4 位的 ECC (用于 MLC NAND) 0: 512 字节为主数据区 1: 24 字节为元数据	0
ECCType	[24]	ECC 类型选择: 0: SLC (1 位修正) ECC 1: MLC (4 位修正) ECC	0
Reserved	[15]	保留	0
TACLS	[14:12]	CLE & ALE 持续时间设置值 (0~7) 持续时间 = HCLK × TACLS	001
Reserved	[11]	保留	0
TWRPH0	[10:8]	TWRPH0 持续时间设置值 (0~7) 持续时间= HCLK × ( TWRPH0 + 1 )	000
Reserved	[7]	保留	0
TWRPH1	[6:4]	TWRPH1 持续时间设置值 (0~7) 持续时间= HCLK × ( TWRPH1 + 1 )	000
AdvFlash	[3]	预先 NAND Flash 存储器用于自启动 0: 支持 512 字节/页 NAND Flash 存储器 1: 支持 2048 字节/页 NAND Flash 存储器	H/W Set

		该位通过 OM[2] 引脚的状态来决定是否从睡眠模式复位和唤醒。该位也可以通过软件来改变	
Reserved	[2]	保留，必须写 1	1
AddrCycle	[1]	NAND Flash 存储器地址周期用于自启动 AdvFlash AddrCycle 当 AdvFlash 为 0 时， 0: 3 地址周期            1: 4 地址周期 当 AdvFlash 为 1 时， 0: 4 地址周期            1: 5 地址周期 该位通过 OM[1] 引脚的状态来决定是否从睡眠模式复位和唤醒。该位也可以通过软件来改变	H/W Set
Reserved	[0]	保留，必须写 0	0

8.3.3. NAND FLASH 控制寄存器

寄存器	地址	读/写	描述	复位值
NFCONT	0x70200004	读/写	NAND Flash 控制寄存器	0x000100C6

NFCONT	位	描述	初始状态
Reserved	[31:19]	保留	0
ECC	Direction	4 位 ECC 编码/解码控制 0: 解码 4 位 ECC，用于对页的读取 1: 译码 4 位 ECC，用于对页的编程	0
Lock-tight	[17]	Lock-tight 配置： 0: 禁用 Lock-tight            1: 使能 Lock-tight 一旦该位被设置为 1，将不能被清除。只有复位或从睡眠模式唤醒才可以使得该位被禁用（不能通过软件清除） 当该位设置为 1 时，则该区域设置在 (0x70200020) 到	0

		<p>NFEBLK (0x70200024)-1 上开启，除了这部分区域，写或擦除命令是无效的，只有读命令是有效的</p> <p>当试图进行写或擦除被锁定的区域时，将会产生非法操作（NFSTAT [5]被置位）</p> <p>如果 NFSBLK 和 NFEBLK 同样，整个区域将被锁住</p>	
Soft Lock	[16]	<p>软件锁存配置：</p> <p>0：禁用锁            1：使能锁</p> <p>软锁区通过软件可在任何时候进行修改。</p> <p>当该位被设置为1时，设置区域在NFSBLK (0x70200020) 到 NFEBLK (0x70200024)-1 被开启，除了这部分区域，写或擦除命令是无效的，只有读命令是有效的</p> <p>当试图进行写或擦除被锁定的区域时，将会产生非法操作（NFSTAT [5]被置位）</p> <p>如果 NFSBLK 和 NFEBLK 同样，整个区域将被锁住</p>	1
Reserved	[15:13]	保留，可以写 0	000
EnbECCDecINT	[12]	<p>4 位 ECC 的解码完成中断控制：</p> <p>0：禁用中断            1：使能中断</p>	0
Reserved	[11]	保留.	0
EnbIllegalAccINT	[10]	<p>非法访问中断控制：</p> <p>0：禁用中断            1：使能中断</p> <p>当 CPU 试图烧写或擦除锁存区域（在 NFSBLK (0x70200020) 到 NFEBLK (0x70200024)- 1) 设置区域）时，将发生非法访问中断</p>	0
EnbRnBINT	[9]	<p>RnB 状态输入信号转换中断控制：</p> <p>0：禁用 RnB 中断    1：使能 RnB 中断</p>	0
RnB_TransMode	[8]	<p>RnB 转换检测配置：</p> <p>0：检测上升沿        1：检测下降沿</p>	0
MainECCLock	[7]	<p>锁存主区 ECC 生成：</p> <p>0：开启主区 ECC    1：锁存主区 ECC</p>	1



		主区 ECC 状态寄存器是 NFMECC0/1 (0x70200034/38)	
SpareECCLock	[6]	锁存备用区 ECC 生成 0: 开启备用区 ECC    1: 锁存备用区 ECC 备用区 ECC 状态寄存器是 NFSECC (0x7020003C)	1
InitMECC	[5]	1: 初始化主区 ECC 解码器/编码器 (只写)	0
InitSECC	[4]	1: 初始化备用区 ECC 解码器/编码器 (只写)	0
Reserved	[3]	保留 (HW_nCE)	0
Reg_nCE1	[2]	NAND Flash 存储器 nGCS[3] 信号控制: 0: 强制 nGCS[3] 为低 (使能片选) 1: 强制 nGCS[3] 为高 (禁用片选) 注: 即使 Reg_nCE1 和 Reg_nCE0 同时被设置为 0, 它们之中也只有一个被声明	1
Reg_nCE0	[1]	NAND Flash 存储器 nGCS[2] 信号控制。 0: 强制 nGCS[2] 为低 (使能片选) 1: 强制 nGCS[2] 为高 (禁用片选) 注: 在引导程序期间, 自动控制。 只有当 MODE 位为 1 时, 它的值才有效。	1
MODE	[0]	NAND Flash 控制器操作模式: 0: NAND Flash 控制器禁用 (不工作) 1: NAND Flash 控制器使能	0

### 8.3.4. NAND FLASH 命令寄存器

寄存器	地址	读/写	描述	复位值
NFCMMD	0x70200008	读/写	NAND Flash 命令设置寄存器	0x00

NFCMMD	位	描述	初始状态
Reserved	[31:8]	保留	0x00

NFCMMD	[7:0]	NAND Flash 存储器命令值	0x00
--------	-------	-------------------	------

### 8.3.5. NAND FLASH 地址寄存器

寄存器	地址	读/写	描述	复位值
NFADDR	0x7020000C	读/写	NAND Flash 地址设置寄存器	0x0000XX00

REG_ADDR	位	描述	初始状态
Reserved	[31:8]	保留	0x00
NFADDR	[7:0]	NAND Flash 存储器地址值	0x00

### 8.3.6. NAND FLASH 数据寄存器

寄存器	地址	读/写	描述	复位值
NFDATA	0x70200010	读/写	NAND Flash 数据寄存器	0xFFFF

NFDATA	位	描述	初始状态
NFDATA	[31:0]	NAND Flash 读/烧写数据值用于 I/O (注) 参照数据寄存器配置	0xFFFF

### 8.3.7. 主数据区 ECC 寄存器

寄存器	地址	读/写	描述	复位值
NFMECCD0	0x70200014	读/写	NAND Flash ECC 第一个和第二个寄存器用于主数据区读取	0x00000000
NFMECCD1	0x70200018	读/写	NAND Flash ECC 第三个和第四个寄存器用于主数据区读取	0x00000000

NFMECCD0	位	描述	初始状态
Reserved	[31:24]	不使用	0x00
ECCData1	[23:16]	ECC1 用于 I/O[7:0]	0x00
Reserved	[15:8]	不使用	0x00
ECCData0	[7:0]	ECC0 用于 I/O[7:0]	0x00

（注）只有字访问有效。

NFMECCD1	位	描述	初始状态
Reserved	[31:24]	不使用	0x00
ECCData3	[23:16]	ECC3 用于 I/O[7:0]	0x00
Reserved	[15:8]	不使用	0x00
ECCData2	[7:0]	ECC2 用于 I/O[7:0]	0x00

### 8.3.8. 备用区 ECC 寄存器

寄存器	地址	读/写	描述	复位值
NFSECCD	0x7020001C	读/写	NAND Flash ECC（错误纠正码）寄存器用于备用区数据的读取	0x00000000

NFSECCD	位	描述	初始状态
保留	[31:24]	不使用	0x00
SECCData1	[23:16]	第二个备用区 ECC 用于 I/O[7:0]	0x00
Reserved	[15:8]	不使用	0x00
SECCData0	[7:0]	第一个备用区 ECC 用于 I/O[ 7:0]	0x00

（注）只有字或半字访问有效。

8.3.9. 可编程块地址寄存器

寄存器	地址	读/写	描述	复位值
NFSBLK	0x70200020	读/写	NAND Flash 可编程起始块地址	0x000000
NFSBLK	0x70200024	读/写	NAND Flash 可编程结束块地址  Nand Flash 可以在开始和结束地址间编程  当 Soft lock 或 Lock-tight 使能,且开始和结束地址有相同值时, 整个 NAND Flash 区域将被锁住	0x000000

NFSBLK	位	描述	初始状态
Reserved	[31:24]	保留	0x00
SBLK_ADDR2	[23:16]	第三块地址用于块擦除操作	0x00
SBLK_ADDR1	[15:8]	第二块地址用于块擦除操作	0x00
SBLK_ADDR0	[7:0]	第一块地址用于块擦除操作  (只有位[7:5]是有效的)	0x00

注：预先的 Flash 的块地址从 3 地址周期启动。所以块地址寄存器只需要 3 字节。

NFEBLK	位	描述	初始状态
Reserved	[31:24]	保留	0x00
EBLK_ADDR2	[23:16]	第三块地址用于块擦除操作	0x00
EBLK_ADDR1	[15:8]	第二块地址用于块擦除操作	0x00
EBLK_ADDR0	[7:0]	第一块地址用于块擦除操作  (只有位[7:5]是有效的)	0x00

注：预先的 Flash 的块地址从 3 地址周期开始。所以块地址寄存器只需要 3 字节。

当 Soft lock 位(NFCNT[16])使能时,NFSLK 和 NFEBLK 可以被改变。但是,当 Lock-tight(NFCNT[17])置位时, NFSLK 和 NFEBLK 不可以被改变。如图 8-7 所示。

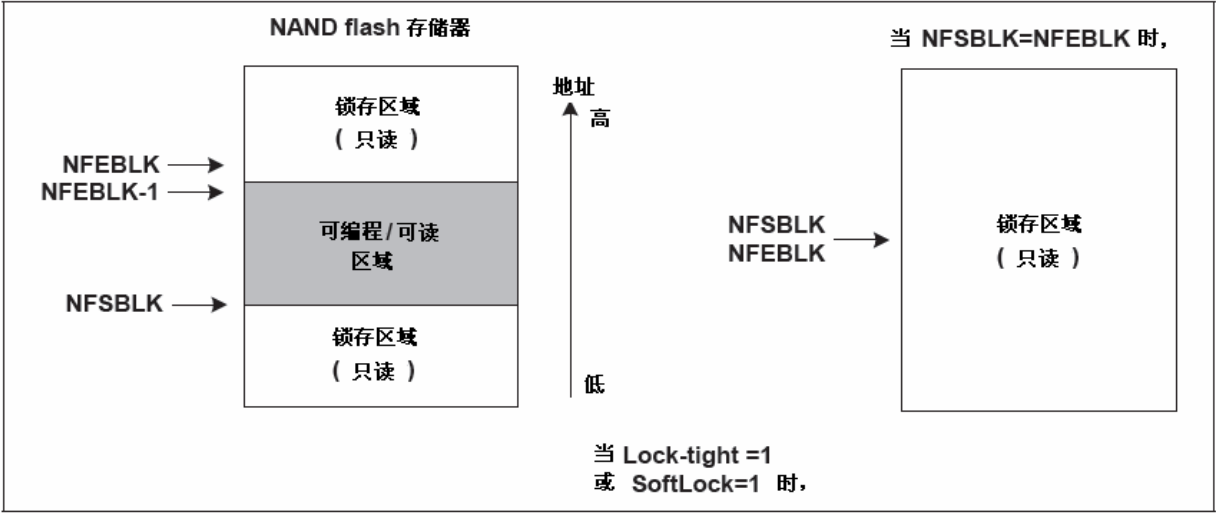


图 8-7 NFSLK 和 NFEBLK 框图

8.3.10. NFCON 状态寄存器

寄存器	地址	读/写	描述	复位值
NFSTAT	0x70200028	读/写	NAND Flash 操作状态寄存器	0x0080001D

NFSTAT	位	描述	初始状态
Reserved	[31:24]	未定义读	0x00
BootDone	[23]	当 NAND Flash 做为引导设备设置时，它将显示加载引导区的 Stepping stone 是否被完成	1
Reserved	[22:7]	保留	0x00
ECCDecDone	[6]	当 4 位 ECC 解码完成时，如果使能其值设置和问题中断 NFMLCBITPT, NFMLCLO 和 NFMLCEL1 有效值写 ‘1’ 1: 4 位 ECC 解码完成	0
IllegalAccess	[5]	一旦 Soft Lock 或 Lock-tight 使能，非法访问（烧写，擦除）存储器将使该位被置位	0

RnB_TransDetect	[4]	当 RnB 由低到高转换发生时，如果使能，其值设置和发生中断。 该位写 ‘1’ 。  0：RnB 转换没被检测 1：RnB 转换被检测  转换配置是设置 RnB_TransMode (NFCONT[8])	1
NCE[1] (只读)	[3]	状态 nCE[1]输出引脚	1
NCE[0] (只读)	[2]	状态 nCE[0]输出引脚	1
Reserved	[1]	保留	0
RnB (只读)	[0]	状态 RnB 输入引脚：  0：NAND Flash 存储器忙 1：NAND Flash 存储器读操作	1

8.3.11. ECC0/1 错误状态寄存器

寄存器	地址	读/写	描述	复位值
NFECERR0	0x7020002C	读	NAND Flash ECC 错误状态寄存器用于 I/O [7:0]。	0x007FFFA
NFECERR1	0x70200030	读	NAND Flash ECC 错误状态寄存器用于 I/O [7:0]。	0x007FFFA

1. 当 ECCType 为 SLC 时。

NFECERR0	位	描述	初始状态
Reserved	[31:25]	保留。	0x00
SErrorDataNo	[24:21]	在备用区，显示数字数据错误。	0011
SErrorBitNo	[20:18]	在备用区，显示位错误。	111
MErrorDataNo	[17:7]	在主数据区，显示数字数据错误。	0x7FF

MErrorBitNo	[6:4]	在主数据区，显示位错误。	111
SpareError	[3:2]	显示备用区是否在位失败时错误发生： 00：无错误            01：1 位错误（纠正） 10：多重错误        11：ECC 区错误	10
MainError	[1:0]	显示主数据区是否在位失败时错误发生： 00：无错误            01：1 位错误（纠正） 10：多重错误        11：ECC 区错误	10

NFECERR1	位	描述	初始状态
Reserved	[31:0]	保留。	0x00

注：当 ECC 寄存器和 ECC 状态寄存器有有效值时，以上值是唯一有效的。

2. 当 ECCType 为 MLC 时。

NFECERR0	位	描述	初始状态
ECC Busy	[31]	显示 4 位 ECC 解码引擎搜索是否有错误存在。 0：空闲                    1：忙	0
ECC Ready	[30]	ECC 准备就绪位。	1
Free Page	[29]	从 NAND flash 中显示页数据所有的‘FF’值。	0
MLC MECC Error	[28:26]	4位ECC解码结果。 000：无错误            001：1位错误 010：2位错误            011：3位错误 100：4位错误            101：无法纠正的 11x：保留	000
2nd Bit Error Location	[25:16]	第二个位错误区域。	0x00
Reserved	[15:10]	保留。	
1st Bit Error Location	[9:0]	第一个位错误区域。	0x00

注：当 ECCDecDone (NFSTAT[6])位设置为(‘1’)时，这些值将被更新。

NFECERR1	位	描述	初始状态
Reserved	[31:26]	保留。	0x00
4th Bit Error Location	[25:16]	第四个错误区域。	0x00
Reserved	[15:10]	保留。	
3rd Bit Error Location	[9:0]	第三个错误区域。	0x00

注：当 ECCDecDone (NFSTAT[6])位设置为(‘1’)时，这些值将被更新。

8.3.12. 主数据区 ECC0 状态寄存器

寄存器	地址	读/写	描述	复位值
NFMECC0	0x70200034	读	SLC 或 MLC NAND Flash ECC 状态寄存器。	0xFFFFFFFF
NFMECC1	0x70200038	读	MLC NAND Flash ECC 状态寄存器。	0xFFFFFFFF

1. 当 ECCType 为 SLC 时。

NFMECC0	位	描述	初始状态
MECC0_3	[31:24]	ECC3 用于数据[7:0]。	0xFF
MECC0_2	[23:16]	ECC2 用于数据[7:0]。	0xFF
MECC0_1	[15:8]	ECC1 用于数据[7:0]。	0xFF
MECC0_0	[7:0]	ECC0 用于数据[7:0]。	0xFF

NFMECC1	位	描述	初始状态
Reserved	[31:0]	保留。	0x00000000

注：当 MainECCLock(NFCONT[7])位设置为 ‘0’（开启），读或写主数据区时，NAND Flash 控制器生成 NFMECC。



2. 当 ECCType 为 MLC 时。

NFMECC0	位	描述	初始状态
4th Parity	[31:24]	第四个从主区（512 字节）生成奇偶校验。	0x00
3rd Parity	[23:16]	第三个从主区（512 字节）生成奇偶校验。	0x00
2nd Parity	[15:8]	第二个从主区（512 字节）生成奇偶校验。	0x00
1st Parity	[7:0]	第一个从主区（512 字节）生成奇偶校验。	0x00

NFMECC1	位	描述	初始状态
Reserved	[31:24]	保留。	0x00
7th Parity	[23:16]	第七个从主区（512 字节）生成奇偶校验。	0x00
6th Parity	[15:8]	第六个从主区（512 字节）生成奇偶校验。	0x00
5th Parity	[7:0]	第五个从主区（512 字节）生成奇偶校验。	0x00

注：当MainECCLock（NFCON[7]）位设置为‘0’（开启），写主数据区时，NAND Flash控制器生成这些ECC奇偶校验码。

8.3.13. 备用区 ECC 状态寄存器

寄存器	地址	读/写	描述	复位值
NFSECC	0x7020003C	读	NAND Flash ECC 寄存器用于 I/O [7:0]。	0xFFFFFFFF

NFSECC	位	描述	初始状态
Reserved	[31:16]	保留。	0xFFFF
SECC0_1	[15:8]	备用区 ECC1 状态寄存器用于 I/O [7:0]。	0xFF

SECC0_0	[7:0]	备用区 ECC0 状态寄存器用于 I/O [7:0]。	0xXX
---------	-------	-----------------------------	------

注：当SpareECCLock(NFCNT[6])位设置为 ‘0’（开启），读或写备用数据区时，NAND flash控制器生成NFSECC。

### 8.3.14. MLC 4 位 ECC 错误模式寄存器

寄存器	地址	读/写	描述	复位值
NFMLCBITPT	0x70200040	读	NAND Flash 4 位 ECC 错误模式寄存器用于数据[7:0]。	0x00000000

NFMLCBITPT	位	描述	初始状态
4th Error bit pattern	[31:24]	第四个错误位模式。	0x00
3rd Error bit pattern	[23:16]	第三个错误位模式。	0x00
2nd Error bit pattern	[15:8]	第二个错误位模式。	0x00
1st Error bit pattern	[7:0]	第一个错误位模式。	0x00

## 8.4 NAND 控制器应用

这小节主要通过部分 NAND 控制器的程序来看一下寄存器的应用。

### 8.4.1. 读取

```
NAND_eERROR NAND_ReadPageSLC(u32 Controller, u32 uBlock, u32 uPage, u8 *pBuffer, u8 *pSpareBuffer)
{
    u32 i, uBlockPage, uPageSize;
    u32 uMecc, uSecc;
    NAND_eERROR eError;
```

```

u32 *pBufferW;

pBufferW = (u32 *)pBuffer;

if(NAND_Inform[Controller].uNandType == NAND_Normal8bit)
{
    uBlockPage=(uBlock<<5) + uPage;
    uPageSize = NAND_Inform[Controller].uPageSize;
}
else {
    uBlockPage=(uBlock<<6) + uPage;
    uPageSize = NAND_Inform[Controller].uPageSize;
}

NF_RSTTECC(Controller);    //初始化 ECC
NF_MECC_UnLock(Controller);

NF_nFCE_L(Controller);
NF_CLEAR_RnB(Controller);

//if(NAND_Inform[Controller].uNandType == NAND_Normal8bit)
if(NAND_Inform[Controller].uAddrCycle == 4)
{
    NF_CMD(Controller, 0x00); // 读取指令
    NF_ADDR(Controller, 0);
}
else if(NAND_Inform[Controller].uAddrCycle == 5)
{
    NF_CMD(Controller, 0x00); // 第一个读取指令

```

```

    NF_ADDR(Controller, 0);
    NF_ADDR(Controller, 0);
}

```

```

NF_ADDR(Controller, uBlockPage&0xff);
NF_ADDR(Controller, (uBlockPage>>16)&0xff); //块和页的数量
NF_DETECT_RnB(Controller);

```

```

if(NAND_Inform[Controller].uNandType == NAND_Advanced8bit)
    NF_CMD(Controller, 0x30); // 第二个指令

```

```

NF_DETECT_RnB(Controller);
    #if NAND_TRANSFER_MODE == POLLING_TRANSFER
    #if 0
        for(i=0 ; i<uPageSize ; i++)
        {
            *pBuffer++ = NF_RDDATA8(Controller); // 读取一页
        }
    #else
        for(i=0 ; i<uPageSize/4 ; i++)
        {
            *pBufferW++ = NF_RDDATA(Controller); // 读取一页
        }
    #endif

```

```

    #elif NAND_TRANSFER_MODE == DMA_TRANSFER

```

```

        Nand_DmaDone = 0;

```

```

        DMACH_Setup(DMA_A, 0x0, (u32)(&(NAND(Controller)->rNFDATA)), 1, (u32)pBufferW, 0,
WORD, uPageSize/4, DEMAND, DMA1_NAND_TX, MEM, BURST128, &g_oNandDmac1);

```

```

    DMACH_Start(&g_oNandDmac1);
    while(!Nand_DmaDone);
#endif

NF_MECC_Lock(Controller);

if(NAND_Inform[Controller].uNandType == NAND_Normal8bit)
{
    NF_SECC_UnLock(Controller);

    //主区域 ECC 检测(1st ~ 4th byte : 主区域 ECC 信息)
    uMecc = NF_RDDATA(Controller);

    if(NAND_Inform[Controller].uSpareECCtest == 0)
    {
        Outp32(&NAND(Controller)->rNFMECCD0, ((uMecc&0xff00)<<8)|(uMecc&0xff) );
        Outp32(&NAND(Controller)->rNFMECCD1,
((uMecc&0xff000000)>>8)|((uMecc&0xff0000)>>16) );
    }
    else if(NAND_Inform[Controller].uSpareECCtest == 1)
    {
        //正确地写入 MECC 值以用于 SpareECC 测试
        u32 uTempMecc;

        uTempMecc = ((u32)aNand_Spare_Data_Temp[3]<<24|
((u32)aNand_Spare_Data_Temp[2]<<16)|
((u32)aNand_Spare_Data_Temp[1]<<8)|
((u32)aNand_Spare_Data_Temp[0]));
        Outp32(&NAND(Controller)->rNFMECCD0, ((uTempMecc&0xff00)<<8)|(uTempMecc&0xff) );
    }
}

```

```

        Outp32(&NAND(Controller)->rNFMECCD1,
((uTempMecc&0xff000000)>>8)|((uTempMecc&0xff0000)>>16) );
    }

    NF_SECC_Lock(Controller);

    aNand_Spare_Data[0] = (u8)(uMecc&0xFF);
    aNand_Spare_Data[1] = (u8)((uMecc&0xFF00)>>8);
    aNand_Spare_Data[2] = (u8)((uMecc&0xFF0000)>>16);
    aNand_Spare_Data[3] = (u8)((uMecc&0xFF000000)>>24);

    aNand_Spare_Data[4] = NF_RDDATA8(Controller);        //读取第五个字节
}
else if(NAND_Inform[Controller].uNandType == NAND_Advanced8bit)
{
    aNand_Spare_Data[0] = NF_RDDATA8(Controller);        // 无效的块检测数据

    NF_SECC_UnLock(Controller);

    // 主区域 ECC 检测
    uMecc = NF_RDDATA(Controller);

    if(NAND_Inform[Controller].uSpareECCTest == 0)
    {
        Outp32(&NAND(Controller)->rNFMECCD0, ((uMecc&0xff00)<<8)|(uMecc&0xff) );
        Outp32(&NAND(Controller)->rNFMECCD1,
((uMecc&0xff000000)>>8)|((uMecc&0xff0000)>>16) );
    }
}

```

```

    NF_SECC_Lock(Controller);

    aNand_Spare_Data[1] = (u8)(uMecc&0xFF);
    aNand_Spare_Data[2] = (u8)((uMecc&0xFF00)>>8);
    aNand_Spare_Data[3] = (u8)((uMecc&0xFF0000)>>16);
    aNand_Spare_Data[4] = (u8)((uMecc&0xFF000000)>>24);
}

for(i=5; i<NAND_Inform[Controller].uSpareSize ; i++)
{
    aNand_Spare_Data[i] = NF_RDDATA8(Controller);
}

for(i=0; i<NAND_Inform[Controller].uSpareSize ; i++)
    *pSpareBuffer++ = aNand_Spare_Data[i];

// 剩余区域 ECC 检测
uSecc = (aNand_Spare_Data[9]<<8) | (aNand_Spare_Data[8]);

Outp32(&NAND(Controller)->rNFSECCD, ((uSecc&0xff00)<<8)|(uSecc&0xff) );

NF_nFCE_H(Controller);

eError = NAND_CheckECCErrror(Controller);

return eError;
}

```

## 8.4.2. 写入

```
NAND_eERROR NAND_WritePageSLC(u32 Controller, u32 uBlock, u32 uPage, u8 *pBuffer, u8
*aSpareBuffer)
{
//Controller - Nand 控制器端口数
//uBlock – 写入块
//uPage – 写入页
//pBuffer – 写入数据缓冲区
//aSpareBuffer - Memory to store the spare data
    u32 i, uBlockPage, uPageSize, uSpareSize, uTemp;
    u32 uMecc, uSecc, uStatus;
    NAND_eERROR eError;
    u32 *pBufferW;

    pBufferW = (u32 *)pBuffer;
    g_Nand_RnBTransition = 0;

    uTemp = NAND_ReadNFCONRegister(Controller);
    uTemp &= ~((1<<13)|(1<<12));    // 4 位 ECC 编码解码完成中断无效
    uTemp |= (1<<10) | (1<<9);    // 非法访问 RnB 转换中断有效
    Outp32(&(NAND(Controller)->rNFCONT), uTemp);

    Outp32(&(NAND(Controller)->rNFSTAT), (1<<5)|(1<<4)); //非法访问和 RnB 中断处理位清除
    INTC_Enable(NUM_NFC);

    if(NAND_Inform[Controller].uNandType == NAND_Normal8bit)
    {
        uBlockPage=(uBlock<<5) + uPage;
```



```

        uPageSize = NAND_Inform[Controller].uPageSize;
        uSpareSize = NAND_Inform[Controller].uSpareSize;
    }
else //if(NAND_Inform[Controller].uNandType == NAND_Advanced8bit)
{
    uBlockPage=(uBlock<<6) + uPage;
    uPageSize = NAND_Inform[Controller].uPageSize;
    uSpareSize = NAND_Inform[Controller].uSpareSize;
}

NF_RSTTECC(Controller);    // 初始化 ECC
NF_MECC_UnLock(Controller);

NF_nFCE_L(Controller);
NF_CLEAR_RnB(Controller);

NF_CMD(Controller, 0x80); //程序指令
NF_ADDR(Controller, 0);

//if(NAND_Inform[Controller].uNandType == NAND_Advanced8bit)
if(NAND_Inform[Controller].uAddrCycle == 5)
    NF_ADDR(Controller, 0);

NF_ADDR(Controller, uBlockPage&0xff);    //
NF_ADDR(Controller, (uBlockPage>>8)&0xff); //块和页数
NF_ADDR(Controller, (uBlockPage>>16)&0xff);    //

#ifdef NAND_TRANSFER_MODE == POLLING_TRANSFER
#endif

```

```

    for(i=0 ; i<uPageSize ; i++)
    {
        NF_WRDATA8(Controller, *pBuffer++);    // 写入一页
    }
#else
    for(i=0 ; i<uPageSize/4 ; i++)
    {
        NF_WRDATA(Controller, *pBufferW++); // 读取一页
    }
#endif
#elif NAND_TRANSFER_MODE == DMA_TRANSFER
    Nand_DmaDone = 0;
    DMACH_Setup(DMA_A, 0x0, (u32)pBufferW, 0, (u32>(&(NAND(Controller)->rNFDATA)), 1,
WORD, uPageSize/4, DEMAND, MEM, DMA1_NAND_RX, BURST128, &g_oNandDmac1);

    DMACH_Start(&g_oNandDmac1);
    while(!Nand_DmaDone);
#endif

NF_MECC_Lock(Controller);
g_Nand_IllegalAccError = 0;

if(NAND_Inform[Controller].uNandType == NAND_Normal8bit)
{
    // Main Area ECC
    uMecc = Inp32(&NAND(Controller)->rNFMECC0);

    aNand_Spare_Data[0] = (u8)(uMecc&0xFF);
    aNand_Spare_Data[1] = (u8)((uMecc&0xFF00)>>8);
}

```

```

aNand_Spare_Data[2] = (u8)((uMecc&0xFF0000)>>16);
aNand_Spare_Data[3] = (u8)((uMecc&0xFF000000)>>24);
aNand_Spare_Data[5] = (u8)0xFF;           //标记有效块

NF_SECC_UnLock(Controller);

for(i=0;i<4;i++)
{
    NF_WRDATA8(Controller, aNand_Spare_Data[i]); //写入剩余阵列(主要是 ECC)
    //NF8_Spare_Data[i]=aNand_Spare_Data[i];
}

NF_SECC_Lock(Controller);

//剩余空间 ECC
uSecc = Inp32(&NAND(Controller)->rNFSECC) & 0xFFFF;

aNand_Spare_Data[8] = (u8)(uSecc&0xFF);
aNand_Spare_Data[9] = (u8)((uSecc&0xFF00)>>8);

for(i=4 ; i<uSpareSize; i++)
{
    NF_WRDATA8(Controller, aNand_Spare_Data[i]); //写入剩余阵列(剩余 ECC 标记)
    //NF8_Spare_Data[i]=aNand_Spare_Data[i];
}

NF_CLEAR_RnB(Controller);
NF_CMD(Controller, 0x10); // 写入第二个指令
}

```

```

else if(NAND_Inform[Controller].uNandType == NAND_Advanced8bit)
{
    // Main Area ECC
    uMecc = Inp32(&NAND(Controller)->rNFMECC0);

    aNand_Spare_Data[0] = (u8)0xFF;           //标记有效块
    aNand_Spare_Data[1] = (u8)(uMecc&0xFF);
    aNand_Spare_Data[2] = (u8)((uMecc&0xFF00)>>8);
    aNand_Spare_Data[3] = (u8)((uMecc&0xFF0000)>>16);
    aNand_Spare_Data[4] = (u8)((uMecc&0xFF000000)>>24);

    NF_WRDATA8(Controller, aNand_Spare_Data[0]); //写入标记有效块

    NF_SECC_UnLock(Controller);
    for(i=1;i<5;i++)
    {
        NF_WRDATA8(Controller, aNand_Spare_Data[i]); //写入剩余阵列 (主要 ECC)
        //NF8_Spare_Data[i]=aNand_Spare_Data[i];
    }
    NF_SECC_Lock(Controller);

    // 剩余 ECC
    uSecc = Inp32(&NAND(Controller)->rNFSECC) & 0xFFFF;

    aNand_Spare_Data[8] = (u8)(uSecc&0xFF);
    aNand_Spare_Data[9] = (u8)((uSecc&0xFF00)>>8);

    for(i=5 ; i<uSpareSize; i++)
    {

```

```

        NF_WRDATA8(Controller, aNand_Spare_Data[i]); //写入剩余阵列(剩余 ECC 和标记)
        //NF8_Spare_Data[i]=aNand_Spare_Data[i];
    }

    NF_CLEAR_RnB(Controller);
    NF_CMD(Controller, 0x10); //写入第二个指令
}

while(!g_Nand_RnBTransition)
{
    if(g_Nand_IllegalAccError == 1)
    {
        NAND_Reset(Controller);
        g_Nand_IllegalAccError = 0;
        return eNAND_ProgramError;
    }
}

//Read Program Status
NF_CMD(Controller, 0x70);

while(!((uStatus = NF_RDDATA8(Controller)) & (1<<6)));
if(uStatus & 0x01)
    eError = eNAND_ProgramError;    // 执行中错误
else
    eError = eNAND_NoError;

NF_nFCE_H(Controller);

```

```

uTemp = NAND_ReadNFCONRegister(Controller);
uTemp &= ~((1<<10) | (1<<9));      // 非法访问 RnB 转换中断无效
Outp32(&(NAND(Controller)->rNFCONT), uTemp);

if(NAND_Inform[Controller].uECCtest == 1)
{
    for(i=0 ; i<uSpareSize ; i++)
        aNand_Spare_Data_Temp[i] = aNand_Spare_Data[i];
}

for(i=0 ; i<uSpareSize ; i++)
{
*aSpareBuffer++ = aNand_Spare_Data[i];
}
INTC_Disable(NUM_NFC);
return eError;
}

```

## 9 CF 控制器

下面主要介绍 CF 控制器在 S3C6410X RISC 微处理器中的功能及应用。

CF 控制器只有一个插槽。CF 控制器由两部分组成，分别是 PC 卡存储器和 ATA 控制器。在实际应用中只能选择一种模式，PC 卡存储器模式或者 True-IDE 模式。CF 控制器有一个顶级的 SFR，其中包括了卡电源启用位、输出端口使能位和模式选择(True-IDE 或 PC 卡)位。

### 9.1. PC 卡控制器的性能

PC 卡控制器有 2 个半字大小的写缓冲区和 4 个半字大小的读缓冲区。

PC 卡控制器有 5 个字尺寸大小的特殊功能寄存器：其中 3 个字为时序配置寄存器；一个字为状态控制配置寄存器；另一个字为中断源和屏蔽寄存器。

时序配置寄存器由三部分组成：安装程序、命令和运行。PC 卡接口包括 IDLE、SETUP、COMMAND、HOLD 四个区域，寄存器的不同部分指明了不同区域的运行时间。

### 9.2. ATA 控制器的性能

- (1) .ATA 控制器与 ATA 标准相兼容。
- (2) .ATA 控制器有一个 16x32 位的 FIFO。
- (3) .ATA 控制器有内部 DMA 控制器（由 ATA 设备到内存或由内存到 ATA 设备）
- (4) .AHB （DMA 控制器）支持 8 个突发字转换。
- (5) .DMA 控制器支持直接模式和间接模式

直接模式：只支持 UDMA 模式。

间接模式：支持 I/O 模式、存储器模式、True-IDE 模式（除了 UDMA 模式）。

9.3. I/O 描述

表 9-1 I/O 信号描述

间接模式	直接模式 (UDMA 模式)	I/O	描述
Xm0CSn[4]	XhiCSn	输出	卡使能触发 PC 卡模式：低字节使能触发 Ture-IDE 模式：芯片选择（nCS0）
	XhiADR[8]		
Xm0CSn[5]	XhiCSn-main	输出	卡使能触发 PC 卡模式：高字节使能触发 Ture-IDE 模式：芯片选择（nCS1）
	XhiADR[9]		
Xm0REGate	Xm0REGate	输出	CF 卡内触发寄存器 PC 卡模式：用来访问 CF 卡内寄存器 Ture-IDE 模式：DMA 应答
	XhiADDR[6]		
Xm00Eata	Xm00Eata	输出	输出使能触发 PC 卡模式：存储器的输出使能触发 Ture-IDE 模式：GND 地
Xm0RESETata	Xm0RESETata	输出	CF 卡复位 PC 卡模式：高电平有效 Ture-IDE 模式：低电平有效
	XhiADDR[4]		
Xm0WEata	Xm0Weata	输出	写使能触发 PC 卡模式：存储器输出使能触发 Ture-IDE 模式：VCC 电源
Xm00En	XhiCSn_sub	输出	I/O 模式读触发 UDMA 模式：主触发
	XhiARD[10]		
Xm00En	XhiWEn	输出	I/O 模式写触发
	XhiARD[110]		
Xm0ADDR[0]	Xm0ADDR[0]	输出	CF 卡 地址



	XuRXD[2] XmmcDATA1[4]		PC 卡模式：所有地址有用 Ture-IDE 模式：只有 ADDR[2:0]有用 其他地址与地连接
Xm0ADDR[1]	Xm0ADDR[1] XuRXD[2] XmmcDATA1[5]	输出	
Xm0ADDR[2]	Xm0ADDR[2] XmmcDATA1[6] XuRXD[3]	输出	
Xm0ADDR[3]		输出	
Xm0DATA[15:0]	XhiDATA[0]	B	CF 数据总线
	XhiDATA[1]	B	
	XhiDATA[2]	B	
	XhiDATA[3]	B	
	XhiDATA[4]	B	
	XhiDATA[5]	B	
	XhiDATA[6]	B	
	XhiDATA[7]	B	
	XhiDATA[8] (XhiDATA[16])	B	
	XhiDATA[9] (XhiDATA[17])	B	
	XhiDATA[10] (XhiCSn)	B	
	XhiDATA[11] (XhiCSn_main)	B	
	XhiDATA[12] (XhiCSn_sub)	B	
	XhiDATA[13] (XhiWE)	B	

	XhiDATA[14] (XhiOEn)	B	
	XhiDATA[15] (XhiRQn)	B	
Xm0CData	Xm0CData	输入	卡检测信号
	XhiADDR[7]		
Xm0INTata	Xm0INTata	输入	CF 卡的中断请求 PC 卡模式：低电平有效（存储器模式：水平触发； I/O 模式：边沿触发） Ture-IDE 模式：高电平有效
	XhiADDR[3]		
Xm0WAITn	XhiADR[12]	输入	CF 卡等待信号 UDMA 模式：设备触发
	XhiOEnl		
Xm0INPACKata	Xm0INPACKata	输入	I/O 模式输入响应： PC 卡模式：还未应用 True-IDE 模式：DMA 请求
	XhiADDR[5]		

9.4. 模块图

CF 控制器模块图如下图 9-1 所示。

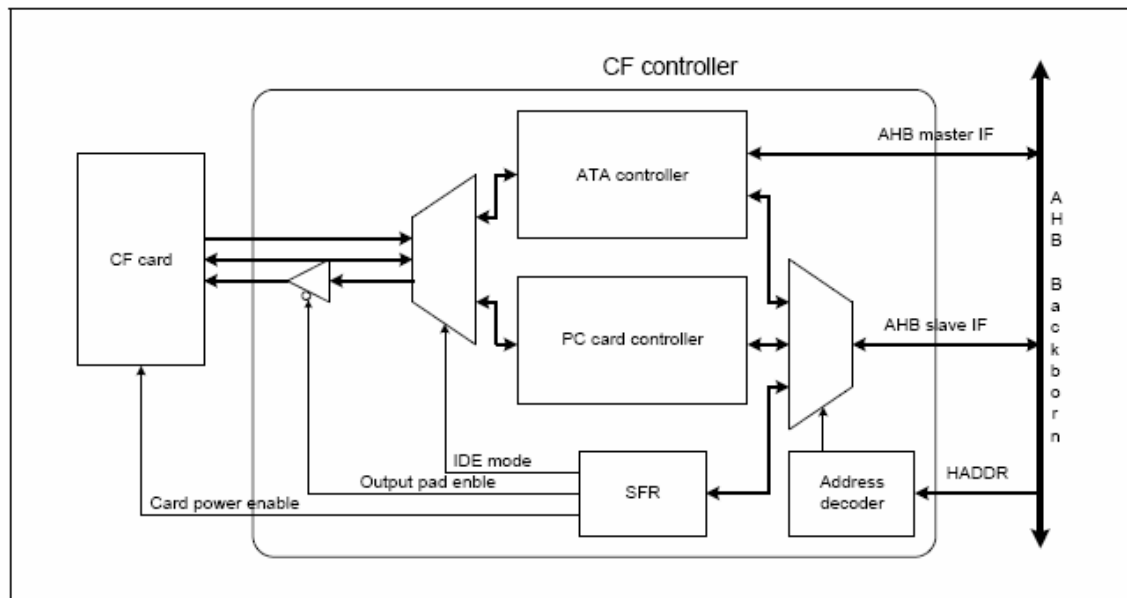


图 9-1 CF 控制器模块图

## 9.5.时序图

### 9.5.1.PC 卡模式

PC 卡模式时序图如图 9-2 所示。

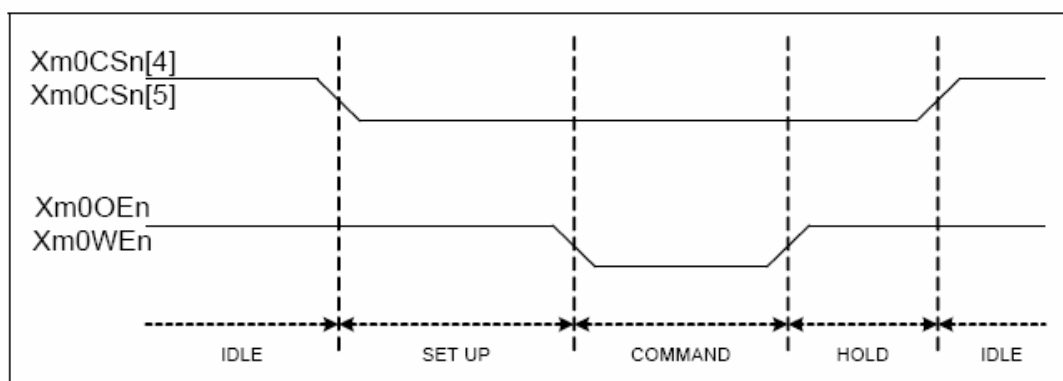


图 9-2 PC 卡模式状态定义

PC 卡模式的时序参数见表 9-2

表 9-2 PC 卡模式的时序参数

Area	Attribute memory	I/O interface	Common memory
	(min, Max) nS		
Set up	(30, --)	(70, --)	(30, --)
Command	(150, --)	(165, --)	(150, --)
Hold	(30, --)	(20, --)	(20, --)
S + C + H	(300, --)	(290, --)	(-, --)

9.5.2. Ture-IDE 模式

PIO 模式波形如图 9-3 所示

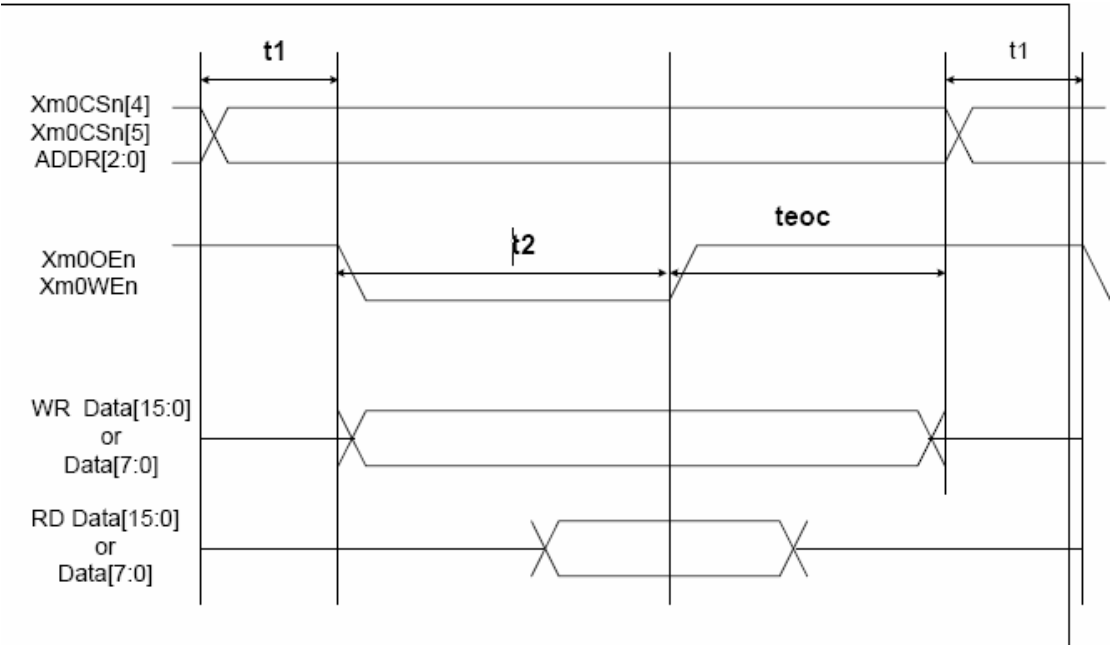


图 9-3 PIO 模式波形

PIO 模式时序参数见表 9-3。

表 9-3 PIO 模式时序参数

PIO mode	PIO 0	PIO 1	PIO 2	PIO 3	PIO 4
T1	(70, --)	(50, --)	(30, --)	(30, --)	(25, --)
T2 (16-bit)	(165, --)	(125, --)	(100, --)	(80, --)	(70, --)
T2 Register (8-bit)	(290, --)	(290, --)	(290, --)	(80, --)	(70, --)
TEOC	(20, --)	(15, --)	(10, --)	(10, --)	(10, --)
T1 + T2 + TEOC	(600, --)	(383, --)	(240, --)	(180, --)	(120, --)

9.5.3. UDMA 模式

直接模式和间接模式

主机可以通过 EBI 控制设备处于间接模式。如果外部存储器 IO 电压不是 3.3v，外部设备的接口信号将需要借用电平转换器完成。数据总线的电平转换器需要一个方向控制位，因为总线数据是双向信号。XhiIRQn 和 XirSDBW 两个管脚可以用来作为方向控制位（这两个管脚作为控制位时只用在 PC-CARD 模式和 PIO 模式下，不能用在 UDMA 模式下）。CF 卡或微驱动器可以直接与 S3C6410X 相连，不需要通过直接模式的存储器端口 0。

UDMA-In Transfer（通过设备终止）

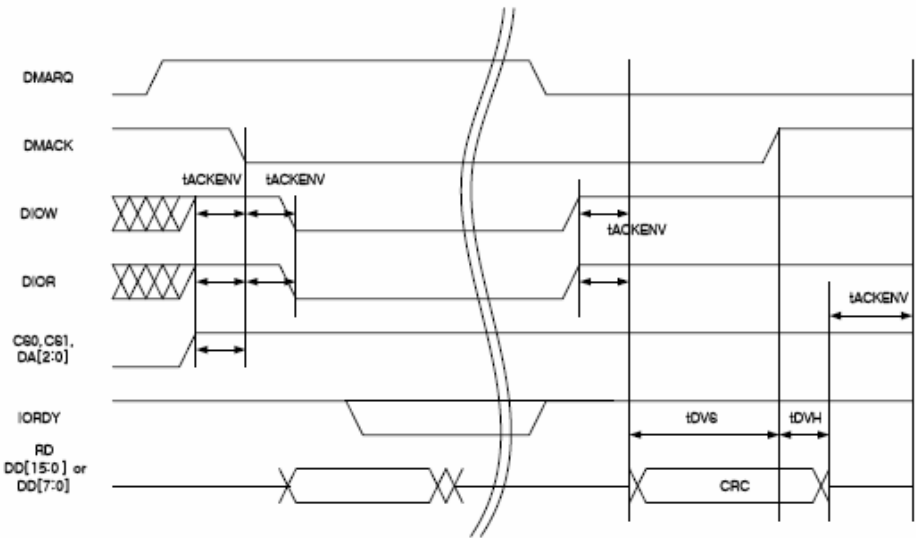


图 9-4 UDMA 输入运行（通过设备终止）

### UDMA-In Transfer (通过主机终止)

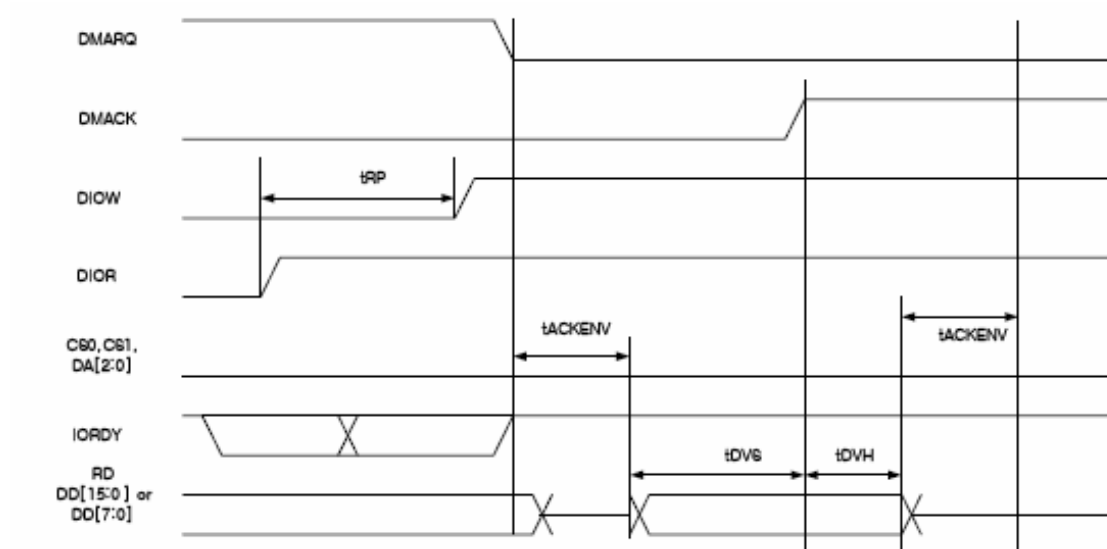


图 9-5 UDMA 输入运行 (通过主机终止)

### UDMA-Out Transfer (通过设备终止)

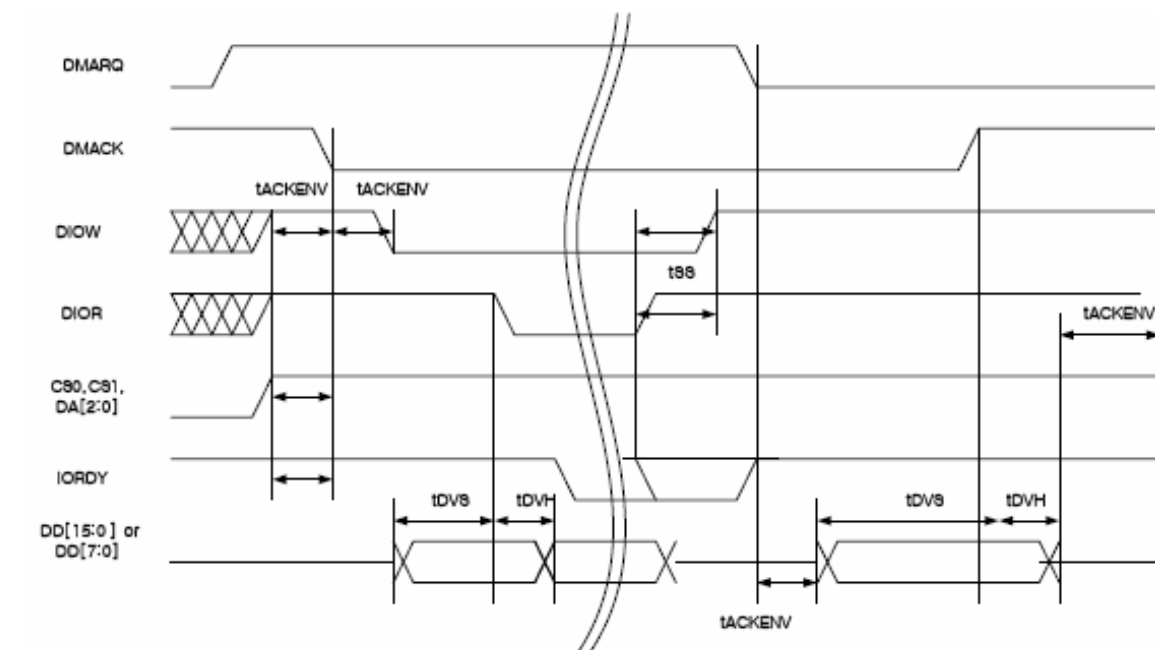


图 9-6 UDMA 输出运行 (通过)

UDMA-Out Transfer（通过主机终止）

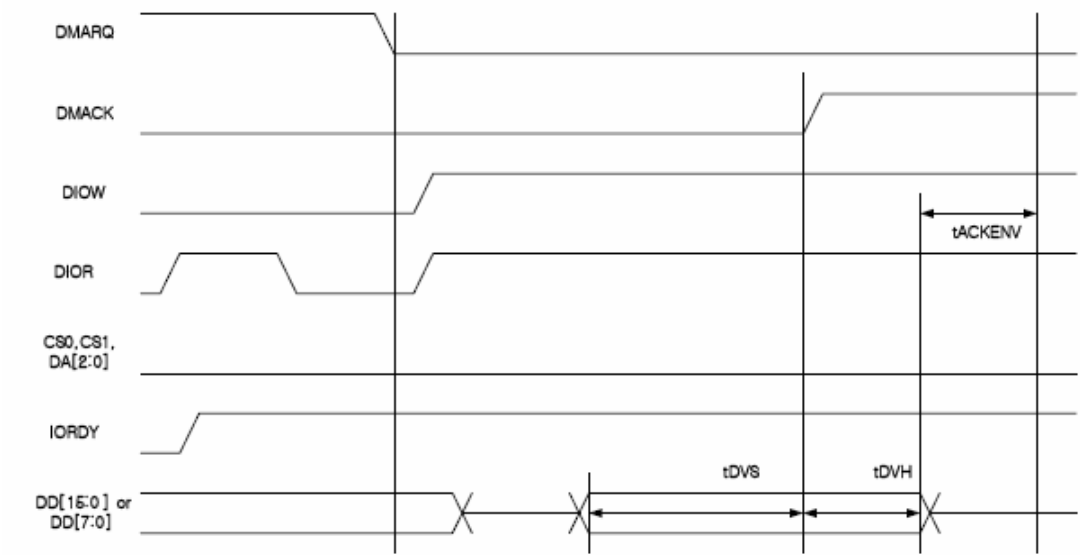


图 9-7 UDMA 输出运行（通过主机终止）

UDMA 模式时序参数见表 9-4

表 9-4 UDMA 模式的时序参数

UDMA mode	UDMA 0	UDMA 1	UDMA 2	UDMA 3	UDMA 4
tACKENV	(20, 70)	(20, 70)	(20, 70)	(20, 55)	(20, 55)
tRP	(160, --)	(125, --)	(100, --)	(100, --)	(100, --)
tSS	(50, --)	(50, --)	(50, --)	(50, --)	(50, --)
tDVS	(70, --)	(48, --)	(31, --)	(20, --)	(6.7, --)
tDVH	(6.2, --)	(6.2, --)	(6.2, --)	(6.2, --)	(6.2, --)
tDVS+tDVH	(120, --)	(80, --)	(60, --)	(45, --)	(30, --)

9.6. 特殊功能寄存器

9.6.1.内存映射

内存映射如图 9-8 所示（CFCON\_Base=0x7030\_0000）

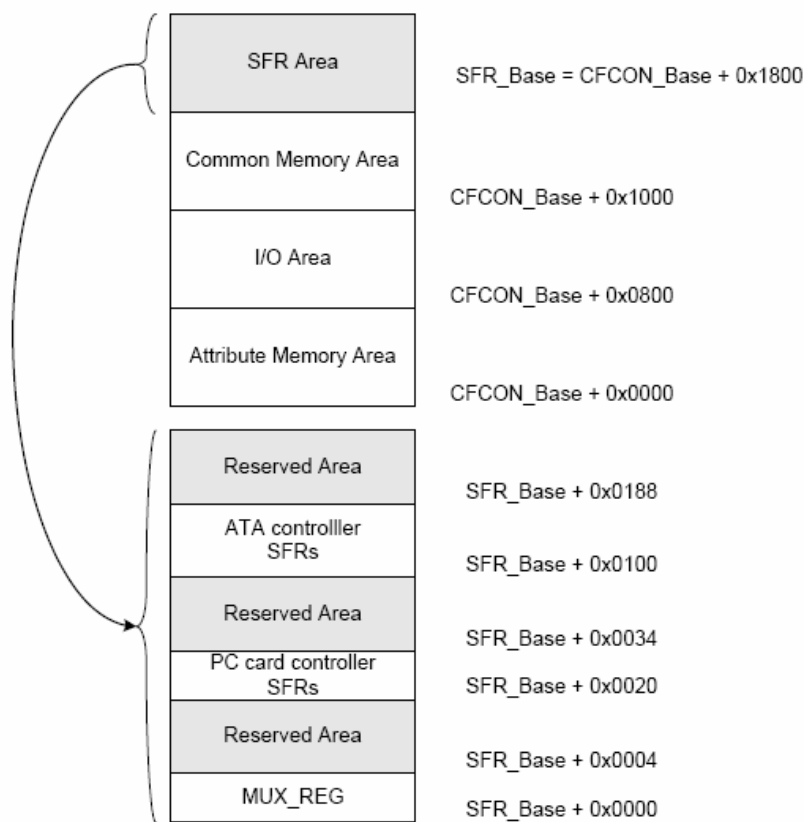


图 9-8 内存映射

9.6.2. 寄存器地址表

寄存器 SFR_BASE	地址	描述	复位值
	0x70301800	CF 卡主机控制基本地址	



MUX_REG	0x70301800	顶层控制和配置寄存器	0x00000006
Reserved	~0x001C	保留地址	
PCCARD_BASE	0X70301820	PC 卡控制基础地址	
PCCARD_ CNFG&STATUS	0x70301820	PC 卡控制和状态寄存器	0x00000F07
PCCARD_ INTMSK&SRC	0x70301824	PC 卡中断屏蔽和源寄存器	0x00000700
PCCARD_ATTR	0x70301828	PC 卡属性存储器  操作时间配置寄存器	0x00031909
PCCARD_I/O	0x7030182c	PC 卡 I/O 空间操作时间配置寄存器	0x00031909
PCCARD_COMM	0x70301830	PC 卡通用寄存器空间操作时间配置寄存器	0x00031909
Reserved	~0x00FC	保留空间	
ATA_BASE	0x70301900	ATA 控制器基础地址	
ATA_CONTROL	0x70301900	ATA 使能和时钟下降状态	0x00000002
ATA_STATUS	0x70301904	ATA 状态	0x00000000
ATA_COMMAND	0x70301908	ATA 命令	0x00000000
ATA_SWRST	0x7030190c	ATA 软件复位	0x00000000
ATA_IRQ	0x70301910	ATA 中断源	0x00000000
ATA_IRQ_MASK	0x70301914	ATA 中断屏蔽	0x0000001F
ATA_CFG	0x7030118	ATA 配置	0x00000000
Reserved	0x70301c~ 0x70301928	保留	
ATA_PIO_TIME	0x7030192c	ATA PIO 时序	0x001C238
ATA_UDMA_TIME	0x70301930	ATA UDMA 时序	0x020b1362
ATA_XFR_NUM	0x70301934	ATA 转换数字	0x00000000
ATA_XFR_CNT	0x70301938	ATA 电路转换	0x00000000
ATA_TBUF_START	0x7030193c	ATA 跟踪缓冲区 开始地址	0x00000000
ATA_TBUF_SIZE	0x70301940	ATA 跟中缓冲区尺寸	0x00000000
ATA_SBUF_START	0x70301944	ATA 源缓冲区开始地址	0x00000000

ATA_SBUF_SIZE	0x70301948	ATA 源缓冲区尺寸	0x00000000
ATA_CADR_TBUF	0x7030194C	ATA 跟踪缓冲区写地址	0x00000000
ATA_CADR_SBUF	0x70301950	ATA 源缓冲区读地址	0x00000000
ATA_PIO_DTR	0x70301954	ATA PIO 设备数据寄存器	0x00000000
ATA_PIO_FED	0x70301958	ATA PIO 设备性能/错误寄存器	0x00000000
ATA_PIO_SCR	0x7030195C	ATA PIO 设备计数寄存器	0x00000000
ATA_PIO_LLR	0x70301960	ATA PIO 设备 LBA 低寄存器	0x00000000
ATA_PIO_LMR	0x70301964	ATA PIO 设备 LBA 寄存器	0x00000000
ATA_PIO_LHR	0x70301968	ATA PIO 设备 LBA 高寄存器	0x00000000
ATA_PIO_DVR	0x7030196C	ATA PIO 设备寄存器	0x00000000
ATA_PIO_CSD	0x70301970	ATA PIO 设备命令/状态寄存器	0x00000000
ATA_PIO_DAD	0x70301974	ATA PIO 设备控制/补充状态寄存器	0x00000000
ATA_PIO_RDATA	0x7030197C	ATA PIO 读数据寄存器	0x00000000
ATA_FIFO_STATUS	0x70301994	ATA 内部 FIFO 状态	0x00000000

### 9.6.3. 寄存器说明

#### 1. MUX\_REG

寄存器	地址	描述	复位值
MUX_REG	0x70301800	MUX_REG 用来设置内部模式、输出端口使能和卡电源使能	0x0000_0006

MUX_REG	位	描述	读/写	复位值
Reserved	[31:1]	保留位		0x3
IDE_MODE	[0]	内部操作模式选择 0: PC 卡模式 1: True-IDE 模式	读/写	0x0

## 2. PCCARD\_CNFG&STATUS

寄存器	地址	描述	复位值
PCCARD_CNFG&STATUS	0x70301820	PCCARD_CNFG&STATUS 用于设置卡的配置以及读取卡的状态	0x0000-0F07

PCCARD_CNFG&STATUS	位	描述	读/写	复位值
Reserved	[31:14]	保留位	读	0x0
CARD_RESET	[13]	在 PC 卡模式下 的 CF 卡复位 0: 不复位 1: 复位	读/写	0x0
INT_SEL	[12]	选择卡中断需求类型（设备）0: 边缘触发 1: 水平触发	读/写	0x0
nWAIT_EN	[11]	nWAIT 使能 0: 禁止 1: 使能	读/写	0x1
DEVICE_ATT	[10]	设备类型是 16 位或者 8 位（性能 存储器空间） 0: 8 位 1: 16 位	读/写	0x1
DEVICE_COMM	[9]	设备类型是 16 位或者 8 位（共用 存储器空间） 0: 8 位 1: 16 位	读/写	0x1
DEVICE_IO	[8]	设备类型是 16 位或者 8 位(I/O 空 间)	读/写	0x1

		0: 8 位 1: 16 位		
Reserved	[7: 4]	保留位	读	0x0
nWAIT	[2]	CF 卡的 nWAIT 0: 等待 1: 准备就绪	读	0x1
nIREQ	[1]	CD 卡的中断请求 0: 中断请求 1: 没有中断请求	读	0x1
nCD	[0]	卡选择 0: 卡选择 1: 卡不选择	读	0x1

### 3. PCCARD\_INTMSK&SRC

寄存器	地址	描述	复位值
PCCARD_INTMSK &SRC	0x70301824	PCCARD_INTMSK&SRC 是中断源或 中断屏蔽寄存器	0x0000_0700

PCCARD_INTMSK& SRC	位	描述	读/写	复位值
Reserved	[31:11]	保留位	读	0x0
INTMSK_ERR_N	[10]	没有卡错误的中断屏蔽位 0: 不屏蔽 1: 屏蔽	读/写	0x1
INTMSK_IREQ	[9]	CF 卡中断请求的终端屏蔽位 0: 不屏蔽 1: 屏蔽	读/写	0x1
INTMSK_CD	[8]	CF 卡检测中断屏蔽位 0: 不屏蔽	读/写	0x1

		1: 屏蔽		
Reserved	[7:3]	保留位	读	0x0
INTMSK_ERR_N	[2]	主机访问时没有卡插槽 CPU 可以通过写入 “1” 清除中断	读/写	0x0
INTMSK_IREQ	[1]	当 CF 卡发出中断请求时, CPU 可以通过 写入 “1” 清除 中断	读/写	0x0
INTMSK_CD	[0]	在插槽内检测 CF 卡时 CPU 可以通过写入 “1” 清除中断	读/写	0x0

#### 4. PCCARD\_ATTR

寄存器	地址	描述	复位值
PCCARD_ATTR	0x70301828	PCCARD_ATTR 用于设置卡访问时间	0x0003_1909

PCCARD_ATTR	位	描述	读/写	复位值
Reserved	[31:23]	保留位	读	0x0
HOLD_ATTR	[22:16]	属性存储器空间的保持时间 $\text{Hold time} = \text{HCLK period} * (\text{HOLD\_ATTR} + 1)$	读/写	0x03
Reserved	[15]	保留位	读	0x0
CMND_ATTR	[14:8]	性能存储器命令区域时间 $\text{Command time} = \text{HCLK priod} * (\text{CMND\_ATTR} + 1)$	读/写	0x19
Reserved	[7]	保留位	读	0x0
SETUP_ATTR	[6:0]	性能存储器的设置区域时间 $\text{Setup time} = \text{HCLK priod} * (\text{SETUP\_ATTR} + 1)$	读/写	0x19

#### 5. PCCARD\_I/O

寄存器	地址	描述	复位值
PCCARD_I/O	0x7030182C	PCCARD_I/O 用于设置卡访问时间	0x0003_1909

PCCARD_I/O	位	描述	读/写	复位值
Reserved	[31:23]	保留位	读	0x0
HOLD_IO	[22:16]	属性存储器空间的保持时间 Hold time=HCLK period*( HOLD_IO+1)	读/写	0x03
Reserved	[15]	保留位	读	0x0
CMND_IO	[14:8]	性能存储器命令区域时间 Command time =HCLK priod*( CMND_IO+1)	读/写	0x19
Reserved	[7]	保留位	读	0x0
SETUP_IO	[6:0]	性能存储器的设置区域时间 Setup time= HCLK priod*( SETUP_IO +1)	读/写	0x19

## 6. PCCARD\_COMM

寄存器	地址	描述	复位值
PCCARD_COMM	0x70301830	PCCARD_COMM 用于设置卡访问时间	0x0003_1909

PCCARD_COMM	位	描述	读/写	复位值
Reserved	[31:23]	保留位	读	0x0
HOLD_ COMM	[22:16]	属性存储器空间的保持时间 Hold time=HCLK period*( HOLD_ COMM +1)	读/写	0x03
Reserved	[15]	保留位	读	0x0
CMND_ COMM	[14:8]	性能存储器命令区域时间 Command time =HCLK priod*( CMND_ COMM +1)	读/写	0x19
Reserved	[7]	保留位	读	0x0
SETUP_ COMM	[6:0]	性能存储器的设置区域时间 Setup time= HCLK priod*( SETUP_ COMM +1)	读/写	0x19

## 7. ATA\_CONTROL

寄存器	地址	描述	复位值
ATA_CONTROL	0x70301900	ATA 使能和时钟下降状态	0x0003_0002

ATA_CONTROL	位	描述	读/写	复位值
Reserved	[31:2]	保留位	读	0x0
CLK_DOWN_READY	[1]	时钟下降状态 当 ATA_CONTROL 位[0]值为 0 时，这个位 位闲置状态 0: 未准备好时钟下降 1: 准备好时钟下降	读/写	0x1
ATA_ENABLE	[0]	ATA 使能 0: ATA 禁止，为时钟下降做准备 1: ATA 使能	读/写	0x0

## 8. ATA\_STATUS

寄存器	地址	描述	复位值
ATA_STATUS	0x70301904	ATA 控制器状态	0x0000_0000

ATA_STATUS	位	描述	读/写	复位值
Reserved	[31:5]	保留位	读	0x0
ATADEV_IRQ	[4]	ATA 中断信号	读	0x0
ATADEV_IORDY	[3]	ATA iordy 信号	读	0x0
ATADEV_DMAREQ	[2]	ATA dmareq 信号	读	0x0
XFR_STATE	[1:0]	转换状态 2'b00:闲置状态 2'b01:转换状态 2'b11:等在状态	读	0x0

## 9. ATA\_COMMAND

寄存器	地址	描述	复位值
ATA_COMMAND	0x70301908	ATA 命令	0x0000_0000

ATA_COMMAND	位	描述	读/写	复位值
Reserved	[31:2]	保留位	读	0x0
XFR-COMMAND	[1:0]	ATA 转换命令 有四种命令类型（START, STOP, ABORT 和 CONTINUE）支持数据转换控制。 “START”命令用来开始数据转换。 “STOP”命令可以暂时停止转换。 “CONTINUW”命令可以继续数据进行数据转换。 “ABORT”命令可以终止当前的数据传输序列，主控制器进行数据转移到空闲状态 00: Stop 命令 01: start 命令 10: abort 命令 11 : continue 命令	读/写	0x0

## 10. ATA\_SWRST

寄存器	地址	描述	复位值
ATA_SWRST	0x7030190c	ATA 软件复位	0x0000_0000

ATA_SWRST	位	描述	读/写	复位值
Reserved	[31:1]	保留位	读	0x0
ATA_SWRSTN	[0]	ATA 主机软件复位 0: 不复位	读/写	0x0



		1: 所有的主机模块软件复位 软件复位之后，继续转换。可以对所有的主控制器和设备寄存器进行设置		
--	--	--	--	--

## 11. ATA\_IRQ

寄存器	地址	描述	复位值
ATA_IRQ	0x70301910	ATA 中断源	0x0000_0000

ATA_IRQ	位	描述	读/写	复位值
Reserved	[31:5]	保留位	读	0x0
SBUF_EMPTY_INT	[4]	当源缓冲区空时，CPU 可以通过写“1”进行终端清除	读/写	0x0
TBUF_FULL_INT	[3]	当跟踪缓冲区有一半空间时，CPU 可以通过写“1”清除中断	读/写	0x0
ATADEV_IRQ_INT	[2]	当 ATA 设备产生中断，CPU 可以通过写“1”清除中断	读/写	0x0
UDMA_HOLE_INT	[1]	当 UDMA 集内的设备屏蔽终止过早，CPU 可以通过写“1”清除中断	读/写	0x0
XFR_DONE_INT	[0]	当所有转换结束，CPU 可以通过写“1”清除中断	读/写	0x0

## 12. ATA\_IRQ\_MASK

寄存器	地址	描述	复位值
ATA_IRQ_MASK	0x70301914	ATA 中断源	0x0000_001F

ATA_IRQ_MASK	位	描述	读/写	复位值
Reserved	[31:2]	保留位	读	0x0
MASK_SBUF_EMPTY_INT	[4]	源缓冲区空时中断屏蔽位 0: 不屏蔽	读/写	0x1

		1: 屏蔽		
MASK_TBUF_ FULL_INT	[3]	目标缓冲区满时中断屏蔽位  0: 不屏蔽 1: 屏蔽	读/写	0x1
MASK_ATADEV_ IRQ_INT	[2]	ATA 设备中断请求时中断屏蔽位  0: 不屏蔽 1: 屏蔽	读/写	0x1
MASK_UDMA_ HOLE_INT	[1]	UDMA 中断屏蔽位  0: 不屏蔽 1: 屏蔽	读/写	0x1
MASK_XFR_ DONE_INT	[0]	Xfr 中断屏蔽位  0: 不屏蔽 1: 屏蔽	读/写	0x0

### 13. ATA\_CFG

寄存器	地址	描述	复位值
ATA_CFG	0x70301918	ATA 接口配置	0x0000_0000

AATA_CFG	位	描述	读/写	复位值
Reserved	[31:10]	保留位	读	0x0
UDMA_AUTO _MODE	[9]	确定在 UDMA 模式下决定是否自动继续提前终止  0: 保持在暂停状态，等在 CPU 活动 1: 继续自动运行	读/写	0x0
SBUF_EMPTY_MODE	[8]	确定当源缓冲区为空时是否自动继续  0: 用新的源缓冲区地址自动继续 1: 保持在暂停状态，等待 CUP 的活动	读/写	0x0
TBUF_FULL_MODE	[7]	确定当跟踪缓冲区满时是否自动继续  0: 用新的跟踪缓冲区地址自动继续	读/写	0x0

		1: 保持在暂停状态，等待 CUP 的活动		
BYTE_SWAP	[6]	确定数据字节很少或大的 16 位数据 0: 很少 1: 大	读/写	0x0
ATADEV_IRQ-AL	[5]	设备中断信号电平 0: 高电平有效 1: 地电平有效	读/写	0x0
DMA_DIR	[4]	DMA 转换方向 0: 主机从设备内读取数据 1: 主机向设备写入数据	读/写	0x0
ATA_CLASS	[3:2]	ATA 转换集选择 2'b00:PIO 转换集 2'b01:PIO DMA 转换集 2'b1x:UDMA 转换集	读/写	0x0
ATA_IORDY_EN	[1]	确定 IORDY 输入是否可以扩展数据转换 0: 禁止扩展数据 1: 可以扩展数据	读/写	0x0
ATA_RST	[0]	ATA 设备通过主机复位 0: 不复位 1: 复位	读/写	0x0

#### 14. ATA\_PIO\_TIME

寄存器	地址	描述	复位值
ATA_PIO_TIME	0x7030192c	ATA PIO 时序	0x0001_C238

ATA_PIO_TIME	位	描述	读/写	复位值
Reserved	[31:20]	保留位	读	0x0
PIO_TEOC	[19:12]	PIO 时间参数，teoc，结束和周期，值不 可以为 0.	读/写	0x1C

		Teoc=HCLK period* (pio_teoc +1)		
PIO_T2	[11:14]	PIO 时间参数, t2, DIOR/Wn 脉冲宽度。 值不可以为 0。 T2= HCLK period* (PIO_T2 +1)	读/写	0x23
PIO_T1	[3: 0]	PIO 时间参数, t1,DIOR/Wn 有效地址 t1= HCLK period* (PIO_T1 +1)	读/写	0x8

## 15. ATA-UDMA\_TIME

寄存器	地址	描述	复位值
ATA-UDMA_TIME	0x70301930	ATA UDMA 时序	0x020B_1362

ATA-UDMA_TIME	位	描述	读/写	复位值
Reserved	[31:28]	保留位	读	0x0
UDMA_TDVH	[27:24]	UDMA 时间参数 TDVH, TDVH=HCLK period* (UDMA_TDVH +1)	读/写	0x2
UDMA_TDVS	[23:16]	UDMA 时间参数 tDVS, 值不可以为零 tDVS= HCLK period* (UDMA_TDVS +1)	读/写	0x0B
UDMA_TRP	[15:8]	UDMA 时间参数 tRP, tRP= HCLK period* (UDMA_TRP +1)	读/写	0x13
UDMA_TSS	[7:4]	UDMA 时间参数 tSS, Tss= HCLK period* (UDMA_TSS +1)	读/写	0x6
UDMA_TACKENV	[3:0]	UDMA 时间参数 tENV, tENV= HCLK period* (UDMA_TACKENV +1)	读/写	0x2

## 16. ATA\_XFR\_NUM

寄存器	地址	描述	复位值
ATA_XFR_NUM	0x70301934	ATA 转换数	0x0000_0000

ATA_XFR_NUM	位	描述	读/写	复位值
XFR_NUM	[31:1]	数据转换数	读/写	0x0000_0000
Reserved	[0]	保留位	读	0x0

#### 17. ATA\_XFR\_CNT

寄存器	地址	描述	复位值
ATA_XFR_CNT	0x70301938	ATA 当前转换数量	0x0000_0000

ATA_XFR_CNT	位	描述	读/写	复位值
XFR_CNT	[31:1]	目前余下的转换数量。当所有数据提前转换过，这个值变为零。	读/写	0x0000_0000
Reserved	[0]	保留位	读	0x0

#### 18. ATA\_TBUF\_START

寄存器	地址	描述	复位值
ATA_TBUF_START	0x7030193C	ATA 跟踪缓冲区的开始地址	0x0000_0000

ATA_TBUF_START	位	描述	读/写	复位值
TRACK_BUFFER_START	[31:0]	跟踪缓冲区的开始地址(4 字节单元地址)	读/写	0x0000_0000

#### 19. ATA\_TBUF\_SIZE

寄存器	地址	描述	复位值
ATA_TBUF_SIZE	0x70301940	ATA 跟踪缓冲区的尺寸	0x0000_0000

ATA_TBUF_SIZE	位	描述	读/写	复位值
TRACK_BUFFER_SIZE	[31:0]	跟踪缓冲区的尺寸。只能设置为	读/写	0x0000_0000

		“Size_of_data_in_bytes-1”.例如 转换 1-sector(512-byte, 32'h200), 必须设置为 32'h1FF(=32'h200-1)		
--	--	--	--	--

## 20. ATA\_SBUF\_START

寄存器	地址	描述	复位值
ATA_SBUF_START	0x70301944	ATA 源缓冲区的开始地址	0x0000_0000

名称	位	描述	读/写	复位值
ATA_SBUF_START	[31:0]	源缓冲区的开始地址	读/写	0x0000_0000

## 21. ATA\_SBUF\_SIZE

寄存器	地址	描述	复位值
ATA_SBUF_SIZE	0x70301948	ATA 源缓冲区的尺寸	0x0000_0000

ATA_SBUF_SIZE	位	描述	读/写	复位值
SRC_BUFFER_SIZE	[31:0]	源缓冲区的尺寸。只能设置为 “Size_of_data_in_bytes-1”.例如 转换 1-sector(512-byte, 32'h200), 必须设置为 32'h1FF(=32'h200-1)	读/写	0x0000_0000

## 22. ATA\_CADDR\_TBUR

寄存器	地址	描述	复位值
ATA_CADDR_TBUR	0x7030194C	ATA 跟踪缓冲区的当前写地址	0x0000_0000

ATA_CADDR_TBUR	位	描述	读/写	复位值
Track_BUF_CUR_ADR	[31:0]	跟踪缓冲区的当前地址	读	0x0000_0000

### 23. ATA\_CADDR\_SBUF

寄存器	地址	描述	复位值
ATA_CADDR_SBUF	0x70301950	ATA 源缓冲区的当前读地址	0x0000_0000

ATA_CADDR_SBUF	位	描述	读/写	复位值
SCR_BUF_CUR_ADR	[31:0]	源缓冲区的当前地址	读	0x0000_0000

### 24. ATA\_PIO\_DTR

寄存器	地址	描述	复位值
ATA_PIO_DTR	0x70301954	ATA 设备数据寄存器	0x0000_0000

ATA_PIO_DTR	位	描述	读/写	复位值
Reserved	[31:16]	保留位	读	0x0
PIO_DEV_DTR*	[15:0]	16 位 PIO 数据寄存器	写	0x0000

注：可以通过访问寄存器 ATA\_PIO\_REATA 读取 PIO\_DEV\_DTR。

### 25. ATA\_PIO\_FED

寄存器	地址	描述	复位值
ATA_PIO_FED	0x70301958	ATA PIO 设备性能/错误寄存器	0x0000_0000

ATA_PIO_FED	位	描述	读/写	复位值
Reserved	[31:8]	保留位	读	0x0
PIO_DEV_FED	[7:0]	8 位 PIO 设备性能/错误寄存器	写	0x00

注：可以通过访问寄存器 ATA\_PIO\_REATA 读取 PIO\_DEV\_FED。

## 26. ATA\_PIO\_SCR

寄存器	地址	描述	复位值
ATA_PIO_SCR	0x7030195C	ATA PIO 部门计数寄存器	0x0000_0000

ATA_PIO_SCR	位	描述	读/写	复位值
Reserved	[31:8]	保留位	读	0x0
PIO_DEV_SCR	[7:0]	8 位 PIO 设备部门计数寄存器	写	0x00

注：可以通过访问寄存器 ATA\_PIO\_REATA 读取 PIO\_DEV\_SCR。

## 27. ATA\_PIO\_LLR

寄存器	地址	描述	复位值
ATA_PIO_LLR	0x7030195C	ATA PIO 设备 LBA 低位寄存器	0x0000_0000

ATA_PIO_LLR	位	描述	读/写	复位值
Reserved	[31:8]	保留位	读	0x0
PIO_DEV_LLR	[7:0]	8 位 PIO 设备 LBA 低位寄存器	写	0x00

注：可以通过访问寄存器 ATA\_PIO\_REATA 读取 PIO\_DEV\_LLR。

## 28. ATA\_PIO\_LMR

地址=0x70301964

寄存器	地址	描述	复位值
ATA_PIO_LMR	0x70301964	ATA PIO 设备 LBA 寄存器	0x0000_0000

ATA_PIO_LMR	位	描述	读/写	复位值
Reserved	[31:8]	保留位	读	0x0
PIO_DEV_LMR	[7:0]	8 位 PIO 设备 LBA 寄存器	写	0x00



注：可以通过访问寄存器 ATA\_PIO\_REATA 读取 PIO\_DEV\_LMR。

29. ATA\_PIO\_LHR

寄存器	地址	描述	复位值
ATA_PIO_LHR	0x70301968	ATA PIO 设备 LBA 高位寄存器	0x0000_0000

ATA_PIO_LMR	位	描述	读/写	复位值
Reserved	[31:8]	保留位	读	0x0
PIO_DEV_LHR	[7:0]	8 位 PIO 设备 LBA 高位寄存器	写	0x00

注：可以通过访问寄存器 ATA\_PIO\_REATA 读取 PIO\_DEV\_LHR。

30. ATA\_PIO\_DVR

寄存器	地址	描述	复位值
ATA_PIO_DVR	0x7030196C	ATA PIO 设备寄存器	0x0000_0000

ATA_PIO_DVR	位	描述	读/写	复位值
Reserved	[31:8]	保留位	读	0x0
PIO_DEV_DVR	[7:0]	8 位 PIO 设备寄存器	写	0x00

注：可以通过访问寄存器 ATA\_PIO\_REATA 读取 PIO\_DEV\_DVR。

31. ATA\_PIO\_CSD

寄存器	地址	描述	复位值
ATA_PIO_CSD	0x70301970	ATA PIO 设备命令/状态寄存器	0x0000_0000

ATA_PIO_CSD	位	描述	读/写	复位值
Reserved	[31:8]	保留位	读	0x0
PIO_DEV_CSD	[7:0]	8 位 PIO 设备命令/状态寄存器	写	0x00

注：可以通过访问寄存器 ATA\_PIO\_REATA 读取 PIO\_DEV\_CSD。

### 32. ATA\_PIO\_DAD

寄存器	地址	描述	复位值
ATA_PIO_DAD	0x70301974	ATA PIO 设备控制/补充寄存器	0x0000_0000

ATA_PIO_DAD	位	描述	读/写	复位值
Reserved	[31:8]	保留位	读	0x0
PIO_DEV_DAD	[7:0]	8 位 PIO 设备控制/补充寄存器	写	0x00

注：可以通过访问寄存器 ATA\_PIO\_REATA 读取 PIO\_DEV\_DAD。

### 33. ATA\_PIO\_RDATA

寄存器	地址	描述	复位值
ATA_PIO_RDATA	0x7030197C	ATA PIO 从设备寄存器读取数据	0x0000_0000

ATA_PIO_RDATA	位	描述	读/写	复位值
Reserved	[31:16]	保留位	读	0x0
PIO_RDATA	[15:0]	PIO 读数据寄存器，主机从 ATA 设个 寄存器内读取数据	读	0x0000

### 34. ATA\_FIFO\_STATUS

寄存器	地址	描述	复位值
ATA_FIFO_STATUS	0x70301994	ATA 内部 ATA FIFO 状态	0x0000_0000

ATA_FIFO_STATUS	位	描述	读/写	复位值
Reserved	[31]	保留位	读	0x0
ATA_STATE	[30:28]	3”B000:IDLE 其他：保留	读	0x0000

PIO_STATE	[27:26]	2'B00:IDLE 2'b01: T1 2'b10:T2 2'b11:TEOC	读	0x0
PDMA_STATE	[25:24]	2'B00:IDLE 2'b01: T1 2'b10:T2 2'b11:TEOC	读	0x0
Reserved	[23:21]	保留位	读	0x0
UDMA_STATE	[20:16]	5'b00000: IDLE 5'b00100:END	读	0x00
Reserved	[15:0]	保留位	读	0x0

## 10 GPIO

S3C6410 包含了 187 个多功能 输入/输出端口管脚。下表列出了 S3C6410 的 17 个端口。

端口名称	管脚数	混合引脚	电压
GPA port	8	UART/EINT	1.8~3.3V
GPB port	7	UART/IrDA/I2C/CF/Ext. DMA/EINT	1.8~3.3V
GPC port	8	SPI/SDMMC/I2S_V40/EINT	1.8~3.3V
GPD port	5	PCM/I2S/AC97/EINT	1.8~3.3V
GPE port	5	PCM/I2S/AC97	1.8~3.3V
GPF port	16	CAMIF/PWM /EINT	1.8~3.3V
GPG port	7	SDMMC/EINT	1.8~3.3V
GPH port	10	SDMMC/KEYPAD/CF/I2S_V40/EINT	1.8~3.3V
GPI port	16	LCD	1.8~3.3V
GPJ port	12	LCD	1.8~3.3V
GPK port	16	HostIF/HIS/KEYPAD/CF	1.8~3.3V
GPL port	15	HostIF/KEYPAD/CF/OTG/EINT	1.8~3.3V
GPMport	6	HostIF /CF/ EINT	1.8~3.3V
GPN port	16	EINT/KEYPADEINT	1.8~3.3V
GP0 port	16	MemoryPort0/EINT	1.8~3.3V
GP0 port	15	MemoryPort0/EINT	1.8~3.3V
GPQ port	16	MemoryPort0/EINT	1.8~3.3V

GPIO 有以下特性：

- （1）可以控制 127 个外部中断
- （2）有 187 个多功能输入/输出端口
- （3）控制管脚的睡眠模式状态，除了 GPK、GPL、GPM、和 GPN 管脚以外。

GPIO 包含两部分，分别是 alive 部分和 off 部分。Alive 部分的电源由睡眠模式提供，off 部分与它不

同。因此，寄存器可以在睡眠模式下保持原值。图 10-1 为 GPIO 的模块图。

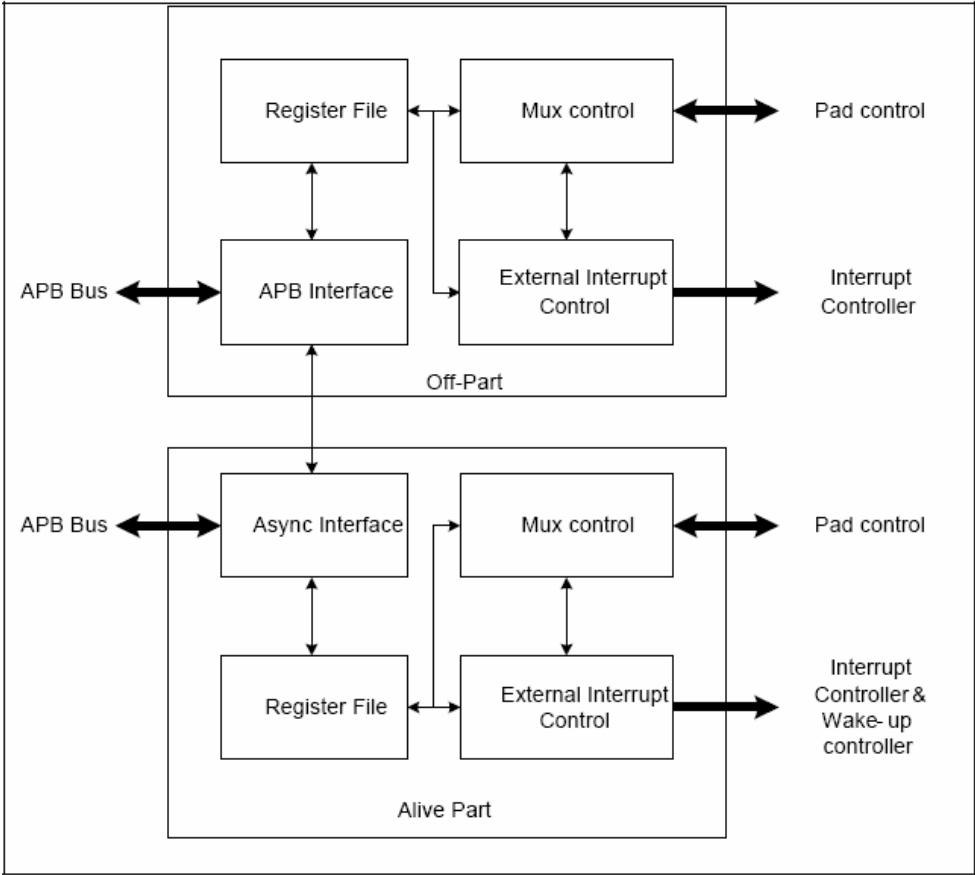


图 10-1 GPIO 模块图

## 10.1.寄存器描述

存储器映射

寄存器	地址	读/写	描述	复位值
GPACON	0x7F008000	读/写	端口 A 配置寄存器	0x0
GPADAT	0x7F008004	读/写	端口 A 数据寄存器	未定义
GPAPUD	0x7F008008	读/写	端口 A 上拉/下拉寄存器	0x0000555
GPACONSLP	0x7F00800C	读/写	端口 A 睡眠模式配置寄存器	0x0
GPAPUDSLP	0x7F008010	读/写	端口 A 睡眠模式拉/下拉寄存器	0x0

GPBCON	0x7F008020	读/写	端口 B 配置寄存器	0x40000
GPBDAT	0x7F008024	读/写	端口 B 数据寄存器	未定义
GPBPUD	0x7F008028	读/写	端口 B 上拉/下拉寄存器	0x00001555
GPBCONSLP	0x7F00802C	读/写	端口 B 睡眠模式配置寄存器	0x0
GPBPUDSLP	0x7F008030	读/写	端口 B 睡眠模式拉/下拉寄存器	0x0
GPCCON	0x7F008040	读/写	端口 C 配置寄存器	0x0
GPCDAT	0x7F008044	读/写	端口 C 数据寄存器	未定义
GPCPUD	0x7F008048	读/写	端口 C 上拉/下拉寄存器	0x00005555
GPCCONSLP	0x7F00804C	读/写	端口 C 睡眠模式配置寄存器	0x0
GPCPUDSLP	0x7F008050	读/写	端口 C 睡眠模式拉/下拉寄存器	0x0
GPDCON	0x7F008060	读/写	端口 D 配置寄存器	0x0
GPDDAT	0x7F008064	读/写	端口 D 数据寄存器	未定义
GPDPUD	0x7F008068	读/写	端口 D 上拉/下拉寄存器	0x00000155
GPDCONSLP	0x7F00806C	读/写	端口 D 睡眠模式配置寄存器	0x0
GPDPUDSLP	0x7F008070	读/写	端口 D 睡眠模式拉/下拉寄存器	0x0
GPECON	0x7F008080	读/写	端口 E 配置寄存器	0x0
GPEDAT	0x7F008084	读/写	端口 E 数据寄存器	未定义
GPEPUD	0x7F008088	读/写	端口 E 上拉/下拉寄存器	0x00000155
GPECONSLP	0x7F00808C	读/写	端口 E 睡眠模式配置寄存器	0x0
GPEPUDSLP	0x7F008090	读/写	端口 E 睡眠模式拉/下拉寄存器	0x0
GPFCON	0x7F0080A0	读/写	端口 F 配置寄存器	0x0
GPFDAT	0x7F0080A4	读/写	端口 F 数据寄存器	未定义
GPFPUD	0x7F0080A8	读/写	端口 F 上拉/下拉寄存器	0x55555555
GPFCONSLP	0x7F0080AC	读/写	端口 F 睡眠模式配置寄存器	0x0
GPFPUDSLP	0x7F0080B0	读/写	端口 F 睡眠模式拉/下拉寄存器	0x0
GPGCON	0x7F0080C0	读/写	端口 G 配置寄存器	0x0
GPGDAT	0x7F0080C4	读/写	端口 G 数据寄存器	未定义
GPGPUD	0x7F0080C8	读/写	端口 G 上拉/下拉寄存器	0x00001555

GPGCONSLP	0x7F0080CC	读/写	端口 G 睡眠模式配置寄存器	0x0
GPGPUDSLP	0x7F0080D0	读/写	端口 G 睡眠模式拉/下拉寄存器	0x0
GPHCON	0x7F0080E0	读/写	端口 H 配置寄存器	0x0
GPHDAT	0x7F0080E4	读/写	端口 H 数据寄存器	未定义
GPHPUD	0x7F0080E8	读/写	端口 H 上拉/下拉寄存器	0x00055555
GPHCONSLP	0x7F0080EC	读/写	端口 H 睡眠模式配置寄存器	0x0
GPHPUDSLP	0x7F0080F0	读/写	端口 H 睡眠模式拉/下拉寄存器	0x0
GPICON	0x7F008100	读/写	端口 I 配置寄存器	0x0
GPIDAT	0x7F008104	读/写	端口 I 数据寄存器	未定义
GPIPUD	0x7F008108	读/写	端口 I 上拉/下拉寄存器	0x55555555
GPICONSLP	0x7F00810C	读/写	端口 I 睡眠配置寄存器	0x0
GPIPUDSLP	0x7F008110	读/写	端口 I 睡眠拉/下拉寄存器	0x0
GPJCON	0x7F008120	读/写	端口 J 配置寄存器	0x0
GPJDAT	0x7F008124	读/写	端口 J 数据寄存器	未定义
GPJPUD	0x7F008128	读/写	端口 J 上拉/下拉寄存器	0x00555555
GPJCONSLP	0x7F00812C	读/写	端口 J 睡眠模式配置寄存器	0x0
GPJPUDSLP	0x7F008130	读/写	端口 J 睡眠模式拉/下拉寄存器	0x0
GPKCON0	0x7F008800	读/写	端口 K 配置寄存器 0	0x22222222
GPKCON1	0x7F008804	读/写	端口 K 配置寄存器 1	0x22222222
GPKDAT	0x7F008808	读/写	端口 K 数据寄存器	未定义
GPKPUD	0x7F00880C	读/写	端口 K 上拉/下拉寄存器	0x55555555
GPLCON0	0x7F008810	读/写	端口 L 配置寄存器 0	0x22222222
GPLCON1	0x7F008814	读/写	端口 L 配置寄存器 1	0x22222222
GPLDAT	0x7F008818	读/写	端口 L 数据寄存器	未定义
GPLPUD	0x7F00881C	读/写	端口 L 上拉/下拉寄存器	0x55555555
GPMCON	0x7F008820	读/写	端口 M 配置寄存器	0x00222222
GPMDAT	0x7F008824	读/写	端口 M 数据寄存器	未定义
GPMPUD	0x7F008828	读/写	端口 M 上拉/下拉寄存器	0x000002AA

GPNCON	0x7F008830	读/写	端口 N 配置寄存器	0x0
GPNDAT	0x7F008834	读/写	端口 N 数据寄存器	未定义
GPNPUD	0x7F008838	读/写	端口 N 上拉/下拉寄存器	0x55555555
GPOCON	0x7F008140	读/写	端口 O 配置寄存器	0xAAAAAAAA
GPODAT	0x7F008144	读/写	端口 O 数据寄存器	未定义
GPOPUD	0x7F008148	读/写	端口 O 上拉/下拉寄存器	0x0
GPOCONSLP	0x7F00814C	读/写	端口 O 睡眠模式配置寄存器	0x0
GPOPUDSLP	0x7F008150	读/写	端口 O 睡眠模式拉/下拉寄存器	0x0
GPPCON	0x7F008160	读/写	端口 P 配置寄存器	0x2AAAAAAAA
GPPDAT	0x7F008164	读/写	端口 P 数据寄存器	未定义
GPPPUD	0x7F008168	读/写	端口 P 上拉/下拉寄存器	0x1011AAA0
GPPCONSLP	0x7F00816C	读/写	端口 P 睡眠模式配置寄存器	0x0
GPPPUDSLP	0x7F008170	读/写	端口 P 睡眠模式拉/下拉寄存器	0x0
GPQCON	0x7F008180	读/写	端口 Q 配置寄存器	0x0002AAAA
GPQDAT	0x7F008184	读/写	端口 Q 数据寄存器	未定义
GPQPUD	0x7F008188	读/写	端口 Q 上拉/下拉寄存器	0x00001555
GPQCONSLP	0x7F00818C	读/写	端口 Q 睡眠模式配置寄存器	0x0
GPQPUDSLP	0x7F008180	读/写	端口 Q 睡眠模式拉/下拉寄存器	0x0
SPCON	0x7F0081A0	读/写	特殊端口配置寄存器	0Xbfc11500
MEM0CONSLP0	0x7F0081C0	读/写	存储器端口 0 睡眠 模式配置 0	0x0
MEM0CONSLP1	0x7F0081C4	读/写	存储器端口 0 睡眠 模式配置 1	0x0
MEM0CONSLP	0x7F0081C8	读/写	存储器端口 0 睡眠模式配置	0x0
MEM0DRVCON	0x7F0081D0	读/写	存储器端口 0 驱动控制寄存器	0x10555551
MEM1DRVCON	0x7F0081D4	读/写	存储器端口 0 驱动控制寄存器	0x0
EINT0CON0	0x7F008900	读/写	外部中断配置寄存器 0	0x0
EINT0CON1	0x7F008904	读/写	外部中断配置寄存器 1	0x0
EINT0FLTCON0	0x7F008910	读/写	外部中断过滤控制寄存器 0	0x0
EINT0FLTCON1	0x7F008914	读/写	外部中断过滤控制寄存器 1	0x0



EINT0FLTCON2	0x7F008918	读/写	外部中断过滤控制寄存器 2	0x0
EINT0FLTCON3	0x7F00891C	读/写	外部中断过滤控制寄存器 3	0x0
EINT0MASK	0x7F008920	读/写	外部中断屏蔽寄存器	0x0FFFFFFF
EINT0PEND	0x7F008924	读/写	外部中断悬挂寄存器 0	0x0
SPCONSLP	0x7F008880	读/写	特殊端口睡眠模式配置寄存器	0x00000010
SLPEN	0x7F008930	读/写	睡眠模式垫配置寄存器	0x0
EINT12CON	0x7F008200	读/写	外部中断 1, 2 配置寄存器	0x0
EINT34CON	0x7F008204	读/写	外部中断 3, 4 配置寄存器	0x0
EINT56CON	0x7F008208	读/写	外部中断 5, 6 配置寄存器	0x0
EINT78CON	0x7F00820C	读/写	外部中断 7, 8 配置寄存器	0x0
EINT9CON	0x7F008210	读/写	外部中断 9 配置寄存器	0x0
EINT12FLTCON	0x7F008220	读/写	外部中断 1, 2 控制寄存器	0x0
EINT34FLTCON	0x7F008224	读/写	外部中断 3, 4 控制寄存器	0x0
EINT56FLTCON	0x7F008228	读/写	外部中断 5, 6 控制寄存器	0x0
EINT78FLTCON	0x7F00822C	读/写	外部中断 7, 8 控制寄存器	0x0
EINT9FLTCON	0x7F008230	读/写	外部中断 9 控制寄存器	0x0
EINT12MASK	0x7F008240	读/写	外部中断 1, 2 屏蔽寄存器	0x00FF7FFF
EINT34 MASK	0x7F008244	读/写	外部中断 3, 4 屏蔽寄存器	0x03FFF03FF
EINT56 MASK	0x7F008248	读/写	外部中断 5, 6 屏蔽寄存器	0x03FF007F
EINT78 MASK	0x7F00824C	读/写	外部中断 7, 8 屏蔽寄存器	0x7FFFFFFF
EINT9 MASK	0x7F008250	读/写	外部中断 9 屏蔽寄存器	0x0000001FF
EINT12PEND	0x7F008260	读/写	外部中断 1, 2 悬挂寄存器	0x0
EINT34 PEND	0x7F008264	读/写	外部中断 3, 4 悬挂寄存器	0x0
EINT56 PEND	0x7F008268	读/写	外部中断 5, 6 悬挂寄存器	0x0
EINT78 PEND	0x7F00826C	读/写	外部中断 7, 8 悬挂寄存器	0x0
EINT9 PEND	0x7F008270	读/写	外部中断 9 悬挂寄存器	0x0
PRIORITY	0x7F008280	0x7F008	优先控制寄存器	0x000003FF
SERVICE	0x7F008284	读	当前服务寄存器	0x0

SERVICEPEND	0x7F008288	读	当前服务悬挂寄存器	0x0
-------------	------------	---	-----------	-----

注：不要访问上面没有定义的地址空间。

## 10.2. Individual 寄存器描述

### 10.2.1 端口 A 控制寄存器

端口 A 控制寄存器包括五个控制寄存器，分别是 GPACON、 GPADAT、 GPAPUD 、GPACONSLP、GPAPUDSLP。

寄存器	地址	读/写	描述	复位值
GPACON	0x7F008000	读/写	端口 A 配置寄存器	0x0000
GPADAT	0x7F008004	读/写	端口 A 数据寄存器	未定义
GPAPUD	0x7F008008	读/写	端口 A 上拉寄存器	0x0005555
GPACONSLP	0x7F00800C	读/写	端口 A 睡眠模式配置寄存器	0x0
GPAPUDSLP	0x7F008010	读/写	端口 A 睡眠模式上拉/下拉寄存器	0x0

GPACON	位	描述		初始状态
GPA0	[3:0]	0000=输入 0010=UART RXD[0] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 1[0]	0000
GPA1	[7:4]	0000=输入 0010=UART TXD[0] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 1[1]	0000
GPA2	[11:8]	0000=输入 0010=UART CTS <sub>n</sub> [0]	0001=输出 0011=保留	0000

		0100=保留 0110=保留	0101=保留 0111=外部中断组 1[2]	
GPA3	[15:12]	0000=输入 0010=UART RTSn [0] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 1[3]	0000
GPA4	[19:16]	0000=输入 0010=UART RXD[1] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 1[4]	0000
GPA5	[23:20]	0000=输入 0010=UART RTXD[1] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 1[5]	0000
GPA6	[27:24]	0000=输入 0010=UART CTSn [1] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 1[6]	0000
GPA7	[31:28]	0000=输入 0010=UART RTSn [1] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 1[7]	0000

GPADAT	位	描述
GPA[7:0]	[7:0]	当端口作为输入端口时，相应的位管脚状态，当端口作为输出段实时，管脚状态等同于相应的位。当端口作为功能管脚是，读取未被定义的值

GPAPUD	位	描述
GPA[n]	[2n+1: 2n]	00=禁止上拉/下拉

	n=0~7	01=下拉使能 10=上拉使能 11=保留
--	-------	-----------------------------

GPACONSLP	位	描述	初始状态
GPA[n]	[2n+1: 2n] n=0~7	00=输出 0 01=输出 1 10=输入 11=与先前状态相同	00

GPAPUDSLP	位	描述
GPA[n]	[2n+1: 2n] n=0~7	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

### 10.2.2 端口 B 控制寄存器

端口 B 控制寄存器包括五个控制寄存器，分别是 GPBCON、GPBDAT、GPBPUD、GPBCONSLP、GPBPUDSLP。

寄存器	地址	读/写	描述	复位值
GPBCON	0x7F008020	读/写	端口 B 配置寄存器	0x40000
GPBDAT	0x7F008024	读/写	端口 B 数据寄存器	未定义
GPBPUD	0x7F008028	读/写	端口 B 上拉寄存器	0x00005555
GPBCONSLP	0x7F00802C	读/写	端口 B 睡眠模式配置寄存器	0x0
GPBPUDSLP	0x7F008030	读/写	端口 B 睡眠模式上拉/下拉寄存器	0x0

GPBCON	位	描述		初始状态
GPB0	[3:0]	0000=输入 0010=UART RXD[2] 0100=IrDA RXD 0110=保留	0001=输出 0011=Ext.DMA 请求 0101=ADDR_CF[0] 0111=外部中断组 1[8]	0000
GPB1	[7:4]	0000=输入 0010=UART TXD[2] 0100=IrDA TXD 0110=保留	0001=输出 0011=Ext.DMA Ack 0101=ADDR_CF[1] 0111=外部中断组 1[9]	0000
GPB2	[11:8]	0000=输入 0010= UART RXD[3] 0100= Ext.DMA Req 0110=I2C SCL[1]	0001=输出 0011= IrDA RXD 0101= ADDR_CF[2] 0111=外部中断组 1[10]	0000
GPB3	[15:12]	0000=输入 0010= UART TXD[3] 0100= Ext.DMA Ack 0110= I2C SDA[1]	0001=输出 0011= IrDA TXD 0101=保留 0111=外部中断组 1[11]	0000
GPB4	[19:16]	0000=输入 0010= IrDA SNBW 0100=CF Data DIR 0110=保留	0001=输出 0011=CAM FIELD 0101=保留 0111=外部中断组 1[12]	0010
GPB5	[23:20]	0000=输入 0010=I2C SCL[0] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 1[13]	0000
GPB6	[27:24]	0000=输入 0010= I2C SDA[0] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 1[14]	0000

GPBDAT	位	描述
GPB[6:0]	[6:0]	当端口作为输入端口时，相应的位处于管脚状态。当端口作为输出端口时，管脚状态于相应的位相同。当端口作为功能管脚时，读取未被定义的值。

GPBPUD	位	描述
GPB[n]	[2n+1: 2n] n=0~6	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

GPBCONSLP	位	描述	初始状态
GPB[n]	[2n+1: 2n] n=0~6	00=输出 0 01=输出 1 10=输入 11=与先前状态相同	00

GPBPUDSLP	位	描述
GPB[n]	[2n+1: 2n] n=0~6	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

### 10.2.3 端口 C 控制寄存器

端口 C 控制寄存器包括五个控制寄存器，分别是 GPCCON、 GPCDAT、 GPCPUD 、GPCCONSLP、GPCPUDSLP。

寄存器	地址	读/写	描述	复位值
GPCCON	0x7F008040	读/写	端口 C 配置寄存器	0x0000
GPCDAT	0x7F008044	读/写	端口 C 数据寄存器	未定义
GPCPUD	0x7F008048	读/写	端口 C 上拉寄存器	0x00005555
GPCCONSLP	0x7F00804C	读/写	端口 C 睡眠模式配置寄存器	0x0
GPCPUDSLP	0x7F008050	读/写	端口 C 睡眠模式上拉/下拉寄存器	0x0

GPCCON	位	描述		初始状态
GPC0	[3:0]	0000=输入 0010=SPI MISO[0] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 2[0]	0000
GPC1	[7:4]	0000=输入 0010=SPI CLK [0] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 2[1]	0000
GPC2	[11:8]	0000=输入 0010=SPI MOSI [0] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 2[2]	0000
GPC3	[15:12]	0000=输入 0010=SPI CSn[0] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 2[3]	0000
GPC4	[19:16]	0000=输入 0010=SPI MISO[1] 0100=保留 0110=保留	0001=输出 0011=MMC CMD2 0101=I2S_V40 DO[0] 0111=外部中断组 2[4]	0000
GPC5	[23:20]	0000=输入	0001=输出	0000

		0010=SPI CLK[1] 0100=保留 0110=保留	0011=MMC CLK2 0101= I2S_V40 DO[1] 0111=外部中断组 2[5]	
GPC6	[27:24]	0000=输入 0010=SPI MOSI [1] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 2[6]	0000
GPC7	[31:28]	0000=输入 0010=SPI CSn [1] 0100=保留 0110=保留	0001=输出 0011=保留 0101= I2S_V40 DO[2] 0111=外部中断组 2[7]	0000

GPCDAT	位	描述
GPC[7:0]	[7:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应的位状态相同。当端口作为功能管脚时，读取未被定义的值。

GPCPUD	位	描述
GPC[n]	[2n+1: 2n] n=0~7	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

GPCCONSLP	位	描述	初始状态
GPC[n]	[2n+1: 2n] n=0~7	00=输出 0 01=输出 1 10=输入 11=与先前状态相同	00



GPCPUDSLP	位	描述
GPC[n]	[2n+1: 2n] n=0~7	00=禁止上拉/下拉  01=下拉使能  10=上拉使能  11=保留

### 10.2.4 端口 D 控制寄存器

端口 D 控制寄存器包括五个控制寄存器，分别是 GPDCON、GPDDAT、GPDPU D 、GPDCONSLP、GPDPU DSLP。

寄存器	地址	读/写	描述	复位值
GPDCON	0x7F008060	读/写	端口 D 配置寄存器	0x00
GPDDAT	0x7F008064	读/写	端口 D 数据寄存器	未定义
GPDPU D	0x7F008068	读/写	端口 D 上拉寄存器	0x00000155
GPDCONSLP	0x7F00806C	读/写	端口 D 睡眠模式配置寄存器	0x0
GPDPU DSLP	0x7F008070	读/写	端口 D 睡眠模式上拉/下拉寄存器	0x0

GPDCON	位	描述		初始状态
GPD0	[3:0]	0000=输入 0010=PCM SCLK[0] 0100=AC97 BITCLK 0110=保留	0001=输出 0011=I2S CLK[0] 0101=保留 0111=外部中断组 3[0]	0000
GPD1	[7:4]	0000=输入 0010=PCM EXTCLK [0] 0100=AC97 RESETn 0110=保留	0001=输出 0011= I2S CDCLK[0] 0101=保留 0111=外部中断组 3[1]	0000
GPD2	[11:8]	0000=输入 0010=PCM FSYNC [0]	0001=输出 0011= I2S LRCLK[0]	0000

		0100=AC97 SYNC 0110=保留	0101=保留 0111=外部中断组 3[2]	
GPD3	[15:12]	0000=输入 0010=PCM SIN[0] 0100=AC97 SDI 0110=保留	0001=输出 0011= I2S DI[0] 0101=保留 0111=外部中断组 3[3]	0000
GPD4	[19:16]	0000=输入 0010=PCM SOUT[0] 0100=AC97 SDO 0110=保留	0001=输出 0011= I2S DO[0] 0101=保留 0111=外部中断组 3[4]	0000

GPDDAT	位	描述
GPD[4:0]	[4:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应位的状态相同。当端口作为功能管脚时，读取未被定义的值。

GPDPUD	位	描述
GPD[n]	[2n+1: 2n] n=0~4	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

GPDCONSLP	位	描述	初始状态
GPD[n]	[2n+1: 2n] n=0~4	00=输出 0 01=输出 1 10=输入 11=与先前状态相同	00

GPDPUDSLP	位	描述
-----------	---	----

GPD[n]	[2n+1: 2n] n=0~4	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留
--------	---------------------	---

### 10.2.5 端口 E 控制寄存器

端口 E 控制寄存器包括五个控制寄存器，分别是 GPECON、 GPEDAT、 GPEPUD 、GPECONSLP、GPEPUDSLP。

寄存器	地址	读/写	描述	复位值
GPECON	0x7F008080	读/写	端口 E 配置寄存器	0x00
GPEDAT	0x7F008084	读/写	端口 E 数据寄存器	未定义
GPEPUD	0x7F008088	读/写	端口 E 上拉寄存器	0x00000155
GPECONSLP	0x7F00808C	读/写	端口 E 睡眠模式配置寄存器	0x0
GPEPUDSLP	0x7F008090	读/写	端口 E 睡眠模式上拉/下拉寄存器	0x0

GPECON	位	描述		初始状态
GPE0	[3:0]	0000=输入 0010=PCM SCLK[1] 0100=AC97 BITCLK 0110=保留	0001=输出 0011=I2S CLK[1] 0101=保留 0111=保留	0000
GPE1	[7:4]	0000=输入 0010=PCM EXTCLK [1] 0100=AC97 RESETn 0110=保留	0001=输出 0011= I2S CDCLK[1] 0101=保留 0111=保留	0000
GPE2	[11:8]	0000=输入 0010=PCM FSYNC [1] 0100=AC97 SYNC	0001=输出 0011= I2S LRCLK[1] 0101=保留	0000

		0110=保留 E	0111=保留	
GPE3	[15:12]	0000=输入 0010=PCM SIN[1] 0100=AC97 SDI 0110=保留	0001=输出 0011= I2S DI[1] 0101=保留 0111=保留	0000
GPE4	[19:16]	0000=输入 0010=PCM SOUT[1] 0100=AC97 SDO 0110=保留	0001=输出 0011= I2S DO[1] 0101=保留 0111=保留	0000

GPEDAT	位	描述
GPE[4:0]	[4:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应位的状态相同。当端口作为功能管脚时，读取未被定义的值。

GPEPUD	位	描述
GPE[n]	[2n+1: 2n] n=0~4	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

GPECONSLP	位	描述	初始状态
GPE[n]	[2n+1: 2n] n=0~4	00=输出 0 01=输出 1 10=输入 11=与先前状态相同	00

GPEPUDSLP	位	描述
GPE[n]	[2n+1: 2n]	00=禁止上拉/下拉

	n=0~4	01=下拉使能 10=上拉使能 11=保留
--	-------	-----------------------------

### 10.2.6 端口 F 控制寄存器

端口 E 控制寄存器包括五个控制寄存器，分别是 GPFCON、 GPFDAT、 GPFPU D 、GPFCONSLP、GPFPU DSLP。

寄存器	地址	读/写	描述	复位值
GPFCON	0x7F0080A0	读/写	端口 F 配置寄存器	0x00
GPFDAT	0x7F0080A4	读/写	端口 F 数据寄存器	未定义
GPFPU D	0x7F0080A8	读/写	端口 F 上拉寄存器	0x55555555
GPFCONSLP	0x7F0080AC	读/写	端口 F 睡眠模式配置寄存器	0x0
GPFPU DSLP	0x7F0080B0	读/写	端口 F 睡眠模式上拉/下拉寄存器	0x0

GPFCON	位	描述	初始状态
GPF0	[1:0]	00=输入                      01=输出 10=CAMIF CLK              11=外部中断组 4[0]	00
GPF1	[3:2]	00=输入                      01=输出 10=CAMIF HREF            11=外部中断组 4[1]	00
GPF2	[5:4]	00=输入                      01=输出 10=CAMIF PCLK            11=外部中断组 4[2]	00
GPF3	[7:6]	00=输入                      01=输出 10=CAMIF RSTn            11=外部中断组 4[3]	00
GPF4	[9:8]	00=输入                      01=输出 10=CAMIF VSYNC           11=外部中断组 4[4]	00
GPF5	[11:10]	00=输入                      01=输出 10=CAMIF YDATA [0]      11=外部中断组 4[5]	00

GPF6	[13:12]	00=输入 10=CAMIF YDATA [1]	01=输出 11=外部中断组 4[6]	00
GPF7	[15:14]	00=输入 10=CAMIF YDATA [2]	01=输出 11=外部中断组 4[7]	00
GPF8	[17:16]	00=输入 10=CAMIF YDATA [3]	01=输出 11=外部中断组 4[8]	00
GPF9	[19:18]	00=输入 10=CAMIF YDATA [4]	01=输出 11=外部中断组 4[9]	00
GPF10	[21:20]	00=输入 10=CAMIF YDATA [5]	01=输出 11=外部中断组 4[10]	00
GPF11	[23:22]	00=输入 10=CAMIF YDATA [6]	01=输出 11=外部中断组 4[11]	00
GPF12	[25:24]	00=输入 10=CAMIF YDATA [7]	01=输出 11=外部中断组 4[12]	00
GPF13	[27:26]	00=输入 10=PWM ECLK	01=输出 11=外部中断组 4[13]	00
GPF14	[29:28]	00=输入 10=PWM TOUT[0]	01=输出 11=CLKOUT[0]	00
GPF15	[31:30]	00=输入 10=PWM TOUT[1]	01=输出 11=保留	00

GPFDAT	位	描述
GPF[15:0]	[15:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应位的状态相同。当端口作为功能管脚时，读取未被定义的值。

GPFPU	位	描述
GPF[n]	[2n+1: 2n] n=0~15	00=禁止上拉/下拉 01=下拉使能

		10=上拉使能 11=保留
--	--	------------------

GPFCONSLP	位	描述	初始状态
GPF[n]	[2n+1: 2n] n=0~15	00=输出 0 01=输出 1 10=输入 11=与先前状态相同	00

GPFPUDSL	位	描述
GPF[n]	[2n+1: 2n] n=0~15	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

### 10.2.7 端口 G 控制寄存器

端口 G 控制寄存器包括五个控制寄存器，分别是 GPGCON 、GPGDAT 、GPGPUD、 GPGCONSLP 、GPGPUDSL。

寄存器	地址	读/写	描述	复位值
GPGCON	0x7F0080C0	读/写	端口 G 配置寄存器	0x00
GPGDAT	0x7F0080C4	读/写	端口 G 数据寄存器	未定义
GPGPUD	0x7F0080C8	读/写	端口 G 上拉寄存器	0x00001555
GPGCONSLP	0x7F0080CC	读/写	端口 G 睡眠模式配置寄存器	0x0
GPGPUDSL	0x7F0080D0	读/写	端口 G 睡眠模式上拉/下拉寄存器	0x0

GPGCON	位	描述	初始状态
--------	---	----	------

GPG0	[3:0]	0000=输入 0010=MMC CLK0 0100=IrDA RXD 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 5[0]	0000
GPG1	[7:4]	0000=输入 0010=MMC CMD0 0100=保留 0110=保留	0001=输出 0011=保留 0101=ADDR_CF[1] 0111=外部中断组 5[1]	0000
GPG2	[11:8]	0000=输入 0010= MMC DATA[0] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 5[2]	0000
GPG3	[15:12]	0000=输入 0010= MMC DATA[1] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 5[3]	0000
GPG4	[19:16]	0000=输入 0010= MMC DATA0[2] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 5[4]	0000
GPG5	[23:20]	0000=输入 0010=MMC DATA0[3] 0100=保留 0110=保留	0001=输出 0011=保留 0101=保留 0111=外部中断组 5[5]	0000
GPG6	[27:24]	0000=输入 0010= MMC CDn0 0100=保留 0110=保留	0001=输出 0011=MMC CDn1 0101=保留 0111=外部中断组 5[6]	0000



GPGDAT	位	描述
GPG[6:0]	[6:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状同于相应位的状态相同。当端口作为功能管脚时，读取未被定义的值

GPGPUD	位	描述
GPG[n]	[2n+1: 2n] n=0~6	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

GPGCONSLP	位	描述	初始状态
GPG[n]	[2n+1: 2n] n=0~6	00=输出 0 01=输出 1 10=输入 11=与先前状态相同	00

GPGPUDSLP	位	描述
GPG[n]	[2n+1: 2n] n=0~6	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

### 10.2.8 端口 H 控制寄存器

端口 H 控制寄存器包括六个控制寄存器，分别是 GPHCON0、GPHCON1、 GPHDAT 、GPHPUD、GPHCONSLP 、GPHPUDSLP。

寄存器	地址	读/写	描述	复位值
-----	----	-----	----	-----

GPHCON0	0x7F0080E0	读/写	端口 H 配置寄存器	0x00
GPHCON1	0x7F0080E4	读/写	端口 H 配置寄存器	0x00
GPHDAT	0x7F0080E8	读/写	端口 H 数据寄存器	未定义
GPHPUD	0x7F0080EC	读/写	端口 H 上拉寄存器	0x00055555
GPHCONSLP	0x7F0080F0	读/写	端口 H 睡眠模式配置寄存器	0x0
GPHPUDSLP	0x7F0080F4	读/写	端口 H 睡眠模式上拉/下拉寄存器	0x0

GPHCON0	位	描述	初始状态
GPH0	[3:0]	0000=输入                      0001=输出 0010=MMC CLK1                0011=保留 0100=Key pad COL[0]        0101=保留 0110=保留                      0111=外部中断组 6[0]	0000
GPH1	[7:4]	0000=输入                      0001=输出 0010=MMC CMD1               0011=保留 0100= Key pad COL[1]        0101=保留 0110=保留                      0111=外部中断组 6[1]	0000
GPH2	[11:8]	0000=输入                      0001=输出 0010= MMC DATA1[0]        0011=保留 0100= Key pad COL[2]        0101=保留 0110=保留                      0111=外部中断组 6[2]	0000
GPH3	[15:12]	0000=输入                      0001=输出 0010= MMC DATA1[1]        0011=保留 0100= Key pad COL[3]        0101=保留 0110=保留                      0111=外部中断组 6[3]	0000
GPH4	[19:16]	0000=输入                      0001=输出 0010= MMC DATA1[2]        0011=保留 0100= Key pad COL[4]        0101=保留 0110=保留                      0111=外部中断组 6[4]	0000
GPH5	[23:20]	0000=输入                      0001=输出	0000

		0010=MMC DATA1[3]    0011=保留 0100= Key pad COL[5]    0101=保留 0110=保留    0111=外部中断组 6[5]	
GPH6	[27:24]	0000=输入    0001=输出 0010= MMC DATA1[4 ]    0011= MMC DATA2[0] 0100= Key pad COL[6]    0101=I2S_V40 BCLK 0110=保留    0111=外部中断组 6[6]	0000
GPH7	[31:28]	0000=输入    0001=输出 0010= MMC DATA1[5 ]    0011= MMC DATA2[1] 0100= Key pad COL[7]    0101=I2S_V40 BCLK 0110=ADDR_CF[1]    0111=外部中断组 6[7]	0000

GPHCON1	位	描述	初始状态
GPH8	[3:0]	0000=输入    0001=输出 0010= MMC DATA1[6 ]    0011= MMC DATA2[2] 0100=保留    0101=I2S_V40 LRCLK 0110=ADDR_CF[2]    0111=外部中断组 6[8]	0000
GPH9	[7:4]	0000=输入    0001=输出 0010= MMC DATA1[7 ]    0011= MMC DATA2[3] 0100=保留    0101=I2S_V40 DI 0110=保留    0111=外部中断组 6[9]	0000

GPHDAT	位	描述
GPH[9:0]	[9:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应位的状态相同。当端口作为功能管脚时，读取未被定义的值

GPHPUD	位	描述
GPH[n]	[2n+1: 2n] n=0~9	00=禁止上拉/下拉 01=下拉使能

		10=上拉使能 11=保留
--	--	------------------

GPHCONSLP	位	描述	初始状态
GPH[n]	[2n+1: 2n] n=0~9	00=输出 0 01=输出 1 10=输入 11=与先前状态相同	00

GPHPUDSLP	位	描述
GPH[n]	[2n+1: 2n] n=0~9	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

### 10.2.9 端口 I 控制寄存器

端口 I 控制寄存器包括五个控制寄存器，分别是 GPICON、 GPIDAT、 GPIPUD 、GPICONSLP、GPIPUDSLP。

寄存器	地址	读/写	描述	复位值
GPICON	0x7F008100	读/写	端口 I 配置寄存器	0x00
GPIDAT	0x7F008104	读/写	端口 I 数据寄存器	未定义
GPIPUD	0x7F008108	读/写	端口 I 上拉寄存器	0x55555555
GPICONSLP	0x7F00810C	读/写	端口 I 睡眠模式配置寄存器	0x0
GPIPUDSLP	0x7F008110	读/写	端口 I 睡眠模式上拉/下拉寄存器	0x0

GPICON	位	描述	初始状态
--------	---	----	------

GPI0	[1:0]	00=输入 10=LCD VD[0]	01=输出 11=保留	00
GPI1	[3:2]	00=输入 10=LCD VD[1]	01=输出 11=保留	00
GPI2	[5:4]	00=输入 10=LCD VD[2]	01=输出 11=保留	00
GPI3	[7:6]	00=输入 10=LCD VD[3]	01=输出 11=保留	00
GPI4	[9:8]	00=输入 10=LCD VD[4]	01=输出 11=保留	00
GPI5	[11:10]	00=输入 10=LCD VD[5]	01=输出 11=保留	00
GPI6	[13:12]	00=输入 10=LCD VD[6]	01=输出 11=保留	00
GPI7	[15:14]	00=输入 10=LCD VD[7]	01=输出 11=保留	00
GPI8	[17:16]	00=输入 10=LCD VD[8]	01=输出 11=保留	00
GPI9	[19:18]	00=输入 10=LCD VD[9]	01=输出 11=保留	00
GPI10	[21:20]	00=输入 10=LCD VD[10]	01=输出 11=保留	00
GPI11	[23:22]	00=输入 10=LCD VD[11]	01=输出 11=保留	00
GPI12	[25:24]	00=输入 10=LCD VD[12]	01=输出 11=保留	00
GPI13	[27:26]	00=输入 10=LCD VD[13]	01=输出 11=保留	00
GPI14	[29:28]	00=输入	01=输出	00

		10=LCD VD[14]                      11=保留	
GPI15	[31:30]	00=输入                      01=输出 10=LCD VD[15]                      11=保留	00

GPI DAT	位	描述
GPI[15:0]	[15:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应位的状态相同。当端口作为功能管脚时，读取未被定义的值。

GPI PUD	位	描述
GPI[n]	[2n+1: 2n] n=0~15	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

GPI SLPCON	位	描述	初始状态
GPI[n]	[2n+1: 2n] n=0~15	00=输出 0 01=输出 1 10=输入 11=与先前状态相同	00

GPI PUDSLP	位	描述
GPF[n]	[2n+1: 2n] n=0~15	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

### 10.2.10 端口 J 控制寄存器

端口 J 控制寄存器包括五个控制寄存器，分别是 GPJCON、 GPJDAT、 GPJPUD 、GPJCONSLP、GPJPUDSLP。

寄存器	地址	读/写	描述	复位值
GPJCON	0x7F008120	读/写	端口 J 配置寄存器	0x00
GPJDAT	0x7F008124	读/写	端口 J 数据寄存器	未定义
GPJPUD	0x7F008128	读/写	端口 J 上拉寄存器	0x05555555
GPJCONSLP	0x7F00812C	读/写	端口 J 睡眠模式配置寄存器	0x0
GPJPUDSLP	0x7F008130	读/写	端口 J 睡眠模式上拉/下拉寄存器	0x0

GPJCON	位	描述		初始状态
GPJ0	[1:0]	00=输入 10=LCD VD[16]	01=输出 11=保留	00
GPJ1	[3:2]	00=输入 10=LCD VD[17]	01=输出 11=保留	00
GPJ2	[5:4]	00=输入 10=LCD VD[18]	01=输出 11=保留	00
GPJ3	[7:6]	00=输入 10=LCD VD[19]	01=输出 11=保留	00
GPJ4	[9:8]	00=输入 10=LCD VD[20]	01=输出 11=保留	00
GPJ5	[11:10]	00=输入 10=LCD VD[21]	01=输出 11=保留	00
GPJ6	[13:12]	00=输入 10=LCD VD[22]	01=输出 11=保留	00
GPJ7	[15:14]	00=输入 10=LCD VD[23]	01=输出 11=保留	00

GPJ8	[17:16]	00=输入 10=LCD HSYNC	01=输出 11=保留	00
GPJ9	[19:18]	00=输入 10=LCD VSYNC	01=输出 11=保留	00
GPJ10	[21:20]	00=输入 10=LCD VDEN	01=输出 11=保留	00
GPJ11	[23:22]	00=输入 10=LCD VCLK	01=输出 11=保留	00

GPJDAT	位	描述
GPJ[11:0]	[11:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应的位的状态相同。当端口作为功能管脚时，读取未被定义的值。

GPJPUD	位	描述
GPJ[n]	[2n+1: 2n] n=0~11	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

GPJSLPCON	位	描述	初始状态
GPJ[n]	[2n+1: 2n] n=0~11	00=输出 0 01=输出 1 1*=输入	00

GPJPUDSLP	位	描述
GPJ[n]	[2n+1: 2n] n=0~11	00=禁止上拉/下拉 01=下拉使能 10=上拉使能



		11=保留
--	--	-------

注意：

当在睡眠模式下设置 LCD Bypass 模式时，GPJSLPCON 和 GPJPUDSLP 不能控制 J 端口。因为此情况下的 J 端口输入单元由主机 I/F 模块控制，信号由 K，L，M 端口单元发出。

10.2.11 端口 K 控制寄存器

端口 K 控制寄存器包括四个控制寄存器，分别是 GPKCON0 、GPKCON1、 GPKDAT 、GPKPUD。

寄存器	地址	读/写	描述	复位值
GPKCON0	0x7F008800	读/写	端口 K 配置寄存器 0	0x22222222
GPKCON1	0x7F008804	读/写	端口 K 配置寄存器 1	0x22222222
GPKDAT	0x7F008808	读/写	端口 K 数据寄存器	未定义
GPKPUD	0x7F00880C	读/写	端口 K 上拉/下拉寄存器	0x55555555

GPKCON0	位	描述		初始状态
GPK0	[3:0]	0000=输入	0001=输出	0010
		0010=Host I/F DATA[0]	0011=HIS RX READY	
		0100=保留	0101=DATA_CF[0]	
		0110=保留	0111=保留	
GPK1	[7:4]	0000=输入	0001=输出	0010
		0010=Host I/F DATA[1]	0011=HIS RX WAKE	
		0100=保留	0101=DATA_CF[1]	
		0110=保留	0111=保留	
GPK2	[11:8]	0000=输入	0001=输出	0010

		0010=Host I/F DATA[2] 0100=保留 0110=保留	0011=HIS RX FLAG 0101=DATA_CF[2] 0111=保留	
GPK3	[15:12]	0000=输入 0010=Host I/F DATA[3] 0100=保留 0110=保留	0001=输出 0011=HIS RX DATA 0101=DATA_CF[3] 0111=保留	0010
GPK4	[19:16]	0000=输入 0010=Host I/F DATA[4] 0100=保留 0110=保留	0001=输出 0011=HIS TX READY 0101=DATA_CF[4] 0111=保留	0010
GPK5	[23:20]	0000=输入 0010=Host I/F DATA[5] 0100=保留 0110=保留	0001=输出 0011=HIS TX WAKE 0101=DATA_CF[5] 0111=保留	0010
GPK6	[27:24]	0000=输入 0010=Host I/F DATA[6] 0100=保留 0110=保留	0001=输出 0011=HIS TX FLAG 0101=DATA_CF[6] 0111=保留	0010
GPK7	[31:28]	0000=输入 0010=Host I/F DATA[7] 0100=保留 0110=保留	0001=输出 0011=HIS TX DATA 0101=DATA_CF[7] 0111=保留	0010

GPKCON1	位	描述		初始状态
GPK8	[3:0]	0000=输入 0010=Host I/F DATA[8] 0100=保留 0110=保留	0001=输出 0011=Key pad ROW[0] 0101=DATA_CF[8] 0111=保留	0010

GPK9	[7:4]	0000=输入 0010=Host I/F DATA[9] 0100=保留 0110=保留	0001=输出 0011= Key pad ROW[1] 0101=DATA_CF[9] 0111=保留	0010
GPK10	[11:8]	0000=输入 0010=Host I/F DATA[10] 0100=保留 0110=保留	0001=输出 0011= Key pad ROW[2] 0101=DATA_CF[10] 0111=保留	0010
GPK11	[15:12]	0000=输入 0010=Host I/F DATA[11] 0100=保留 0110=保留	0001=输出 0011= Key pad ROW[3] 0101=DATA_CF[11] 0111=保留	0010
GPK12	[19:16]	0000=输入 0010=Host I/F DATA[12] 0100=保留 0110=保留	0001=输出 0011= Key pad ROW[4] 0101=DATA_CF[12] 0111=保留	0010
GPK13	[23:20]	0000=输入 0010=Host I/F DATA[13] 0100=保留 0110=保留	0001=输出 0011= Key pad ROW[5] 0101=DATA_CF[13] 0111=保留	0010
GPK14	[27:24]	0000=输入 0010=Host I/F DATA[14] 0100=保留 0110=保留	0001=输出 0011= Key pad ROW[6] 0101=DATA_CF[14] 0111=保留	0010
GPK15	[31:28]	0000=输入 0010=Host I/F DATA[15] 0100=保留 0110=保留	0001=输出 0011= Key pad ROW[7] 0101=DATA_CF[15] 0111=保留	0010

GPKDAT	位	描述
GPK[15:0]	[15:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应位的状态相同。当端口作为功能管脚时，读取未被定义的值

GPKPUD	位	描述
GPK[n]	[2n+1: 2n] n=0~15	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

### 10.2.12 端口 L 控制寄存器

端口 L 控制寄存器包括四个控制寄存器，分别是 GPLCON0、GPLCON1、GPLDAT、GPLPUD。

寄存器	地址	读/写	描述	复位值
GPLCON0	0x7F008810	读/写	端口 L 配置寄存器 0	0x22222222
GPLCON1	0x7F008814	读/写	端口 L 配置寄存器 1	0x22222222
GPLDAT	0x7F008818	读/写	端口 L 数据寄存器	未定义
GPLPUD	0x7F00881C	读/写	端口 L 上拉/下拉寄存器	0x15555555

GPLCON0	位	描述	初始状态
GPL0	[3:0]	0000=输入                      0001=输出 0010=Host I/F ADDR[0]      0011=Key pad COL[0] 0100=保留                      0101=保留 0110= ADDR_CF[0]            0111=保留	0010
GPL1	[7:4]	0000=输入                      0001=输出 0010=Host I/F ADDR[1]      0011=Key pad COL[1] 0100=保留                      0101=保留 0110= ADDR_CF[1]            0111=保留	0010

GPL2	[11:8]	0000=输入 0010=Host I/F ADDR[2] 0100=保留 0110= ADDR_CF[2]	0001=输出 0011=Key pad COL[2] 0101=保留 0111=保留	0010
GPL3	[15:12]	0000=输入 0010=Host I/F ADDR[3] 0100=保留 0110= MEM0_INTata	0001=输出 0011=Key pad COL[3] 0101=保留 0111=保留	0010
GPL4	[19:16]	0000=输入 0010=Host I/F ADDR[4] 0100=保留 0110= MEM0_RESEata	0001=输出 0011=Key pad COL[4] 0101=保留 0111=保留	0010
GPL5	[23:20]	0000=输入 0010=Host I/F ADDR[5] 0100=保留 0110= MEM0_INPACKata	0001=输出 0011=Key pad COL[5] 0101=保留 0111=保留	0010
GPL6	[27:24]	0000=输入 0010=Host I/F ADDR[6] 0100=保留 0110= MEM0_REGata	0001=输出 0011=Key pad COL[6] 0101=保留 0111=保留	0010
GPL7	[31:28]	0000=输入 0010=Host I/F ADDR[7] 0100=保留 0110= MEM0_CData	0001=输出 0011=Key pad COL[7] 0101=保留 0111=保留	0010

GPLCON1	位	描述		初始状态
GPL8	[3:0]	0000=输入 0010=Host I/F ADDR[8] 0100=保留	0001=输出 0011=Ext.Interrupt[16] 0101=CE_CF[0]	0010

		0110=保留	0111=保留	
GPL9	[7:4]	0000=输入 0010=Host I/F ADDR[9] 0100=保留 0110=保留	0001=输出 0011=Ext.Interrupt[17] 0101=CE_CF[1] 0111=保留	0010
GPL10	[11:8]	0000=输入 0010=Host I/F ADDR[10] 0100=保留 0110=保留	0001=输出 0011=Ext.Interrupt[18] 0101=IORD_CF 0111=保留	0010
GPL11	[15:12]	0000=输入 0010=Host I/F ADDR[11] 0100=保留 0110=保留	0001=输出 0011=Ext.Interrupt[19] 0101=LOWR_CF 0111=保留	0010
GPL12	[19:16]	0000=输入 0010=Host I/F ADDR [12] 0100=保留 0110=保留	0001=输出 0011= Ext.Interrupt[20] 0101=LORDY_CF 0111=保留	0010
GPL13	[23:20]	0000=输入 0010=Host I/F DATA[16] 0100=保留 0110=保留	0001=输出 0011= Ext.Interrupt[21] 0101=保留 0111=保留	0010
GPL14	[27:24]	0000=输入 0010=Host I/F DATA[17] 0100=保留 0110=保留	0001=输出 0011= Ext.Interrupt[22] 0101=保留 0111=保留	0010

GPKDAT	位	描述
GPK[14:0]	[14:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应位的状态相同。当端口作为功能管脚时，读取未被定义的值

GPKPUD	位	描述
GPK[n]	[2n+1: 2n] n=0~14	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

### 10.2.13 端口 M 控制寄存器

端口 M 控制寄存器包括三个控制寄存器，分别是 GPMCON、GPMDAT、GMPUD。

寄存器	地址	读/写	描述	复位值
GPMCON	0x7F008820	读/写	端口 M 配置寄存器	0x22222222
GPMDAT	0x7F008824	读/写	端口 M 数据寄存器	未定义
GMPUD	0x7F008828	读/写	端口 M 上拉/下拉寄存器	0x000002AA

GPMCON	位	描述	初始状态
GPM0	[3:0]	0000=输入                      0001=输出 0010=Host I/F CSn            0011=Ext.Interrupt[23] 0100=保留                      0101=保留 0110= CE_CF[1]                0111=保留	0010
GPM1	[7:4]	0000=输入                      0001=输出 0010=Host I/F CSn_main       0011=Ext.Interrupt[24] 0100=保留                      0101=保留 0110= CE_CF[0]                0111=保留	0010
GPM2	[11:8]	0000=输入                      0001=输出 0010=Host I/F CSn_sub        0011=Ext.Interrupt[25] 0100= Host I/F MDP_VSYNC    0101=保留	0010

		0110= IORD_CF	0111=保留	
GPM3	[15:12]	0000=输入 0010=Host I/F WEn 0100=保留 0110= LOWR_CF	0001=输出 0011= Ext.Interrupt[26] 0101=保留 0111=保留	0010
GPM4	[19:16]	0000=输入 0010=Host I/F OEn 0100=保留 0110= IORDY_CF	0001=输出 0011= Ext.Interrupt[27] 0101=保留 0111=保留	0010
GPM5	[23:20]	0000=输入 0010=Host I/F INTRn 0100=保留 0110=保留	0001=输出 0011=CF Data Dir 0101=保留 0111=保留	0010

GPMDAT	位	描述
GPM[5:0]	[5:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应的位的状态相同。当端口作为功能管脚时，读取未被定义的值

GPMPUD	位	描述
GPM[n]	[2n+1: 2n] n=0~5	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

### 10.2.14 端口 N 控制寄存器

端口 N 控制寄存器包括三个控制寄存器，分别是 GPNCON 、 GPNDAT 、 GPNPUD。

GPNCON 、 GPNDAT 、 GPNPUD 都是 alive 部分。



寄存器	地址	读/写	描述	复位值
GPNCON	0x7F008830	读/写	端口 N 配置寄存器	0x00
GPNDAT	0x7F008834	读/写	端口 N 数据寄存器	未定义
GPNPUD	0x7F008838	读/写	端口 N 上拉/下拉寄存器	0x55555555

GPNCON	位	描述		初始状态
GPN0	[1:0]	00=输入 10=Ext.Interrupt[0]	01=输出 11= Key pad ROW[0]	00
GPN1	[3:2]	00=输入 10=Ext.Interrupt[1]	01=输出 11= Key pad ROW[1]	00
GPN2	[5:4]	00=输入 10=Ext.Interrupt[2]	01=输出 11= Key pad ROW[2]	00
GPN3	[7:6]	00=输入 10=Ext.Interrupt[3]	01=输出 11= Key pad ROW[3]	00
GPN4	[9:8]	00=输入 10=Ext.Interrupt[4]	01=输出 11= Key pad ROW[4]	00
GPN5	[11:10]	00=输入 10=Ext.Interrupt[5]	01=输出 11= Key pad ROW[5]	00
GPN6	[13:12]	00=输入 10=Ext.Interrupt[6]	01=输出 11= Key pad ROW[6]	00
GPN7	[15:14]	00=输入 10=Ext.Interrupt[7]	01=输出 11= Key pad ROW[7]	00
GPN8	[17:16]	00=输入 10= Ext.Interrupt[8]	01=输出 11=保留	00
GPN9	[19:18]	00=输入 10= Ext.Interrupt[9]	01=输出 11=保留	00
GPN10	[21:20]	00=输入 10= Ext.Interrupt[10]	01=输出 11=保留	00

GPN11	[23:22]	00=输入 10= Ext.Interrupt[11]	01=输出 11=保留	00
GPN12	[25:24]	00=输入 10= Ext.Interrupt[12]	01=输出 11=保留	00
GPN13	[27:26]	00=输入 10= Ext.Interrupt[13]	01=输出 11=保留	00
GPN14	[29:28]	00=输入 10= Ext.Interrupt[14]	01=输出 11=保留	00
GPN15	[31:30]	00=输入 10= Ext.Interrupt[15]	01=输出 11=保留	00

GPNDAT	位	描述
GPN[15:0]	[15:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应位的状态相同。当端口作为功能管脚时，读取未被定义的值

GPNPUD	位	描述
GPN[n]	[2n+1: 2n] n=0~15	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

### 10.2.15 端口 O 控制寄存器

端口 O 控制寄存器包括五个控制寄存器，分别是 GPOCON 、 GPODAT 、 GPOPUD、 GPOCONSLP、 GPOPUDSLP。

寄存器	地址	读/写	描述	复位值
GPOCON	0x7F008140	读/写	端口 O 配置寄存器	0xAAAAAAAA

GPODAT	0x7F008144	读/写	端口 O 数据寄存器	未定义
GPOPUD	0x7F008148	读/写	端口 O 上拉寄存器	0x0
GPOCONSLP	0x7F00814C	读/写	端口 O 睡眠模式配置寄存器	0x0
GPOPUDSLP	0x7F008150	读/写	端口 O 睡眠模式上拉/下拉寄存器	0x0

GPOCON	位	描述		初始状态
GPO0	[1:0]	00=输入 10=MEM0_nCS[2]	01=输出 11= Ext.Interrupt Group7[0]	10
GPO1	[3:2]	00=输入 10=MEM0_nCS[3]	01=输出 11= Ext.Interrupt Group7[1]	10
GPO2	[5:4]	00=输入 10=MEM0_nCS[4]	01=输出 11= Ext.Interrupt Group7[2]	10
GPO3	[7:6]	00=输入 10=MEM0_nCS[5]	01=输出 11= Ext.Interrupt Group7[3]	10
GPO4	[9:8]	00=输入 10=保留	01=输出 11= Ext.Interrupt Group7[4]	10
GPO5	[11:10]	00=输入 10=保留	01=输出 11= Ext.Interrupt Group7 [5]	10
GPO6	[13:12]	00=输入 10=MEM0_ADDR[6]	01=输出 11= Ext.Interrupt Group7[6]	10
GPO7	[15:14]	00=输入 10=MEM0_ADDR[7]	01=输出 11= Ext.Interrupt Group7[7]	10
GPO8	[17:16]	00=输入 10=MEM0_ADDR[8]	01=输出 11= Ext.Interrupt Group7[8]	10
GPO9	[19:18]	00=输入 10=MEM0_ADDR[9]	01=输出 11= Ext.Interrupt Group7[9]	10
GPO10	[21:20]	00=输入 10=MEM0_ADDR[10]	01=输出 11= Ext.Interrupt Group7[10]	10
GPO11	[23:22]	00=输入	01=输出	10

		10=MEM0_ADDR[11]    11= Ext.Interrupt Group7[11]	
GPO12	[25:24]	00=输入                      01=输出 10=MEM0_ADDR[12]    11= Ext.Interrupt Group7[12]	10
GPO13	[27:26]	00=输入                      01=输出 10=MEM0_ADDR[13]    11= Ext.Interrupt Group7[13]	10
GPO14	[29:28]	00=输入                      01=输出 10=MEM0_ADDR[14]    11= Ext.Interrupt Group7[14]	10
GPO15	[31:30]	00=输入                      01=输出 10=MEM0_ADDR[15]    11= Ext.Interrupt Group7[15]	10

GPODAT	位	描述
GPO[15:0]	[15:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应位的状态相同。当端口作为功能管脚时，读取未被定义的值

GPOPUD	位	描述
GPO[n]	[2n+1: 2n] n=0~15	00=禁止 上拉/下拉 01=下拉使能 10=上拉使能 11=保留

GPOCONSLP	位	描述	初始状态
GPF[n]	[2n+1: 2n] n=0~15	00=输出 0 01=输出 1 10=输入 11=与先前状态相同	00

GPOPUDSLP	位	描述
GPF[n]	[2n+1: 2n]	00=禁止上拉/下拉

	n=0~15	01=下拉使能 10=上拉使能 11=保留
--	--------	-----------------------------

注意：

- (1) 当端口用于接收存储器接口信号时，不可以进行上拉/下拉。
- (2) 当端口用于接收存储器接口信号时，端口状态由停止模式的 MEM0CONSTOP 控制，MEM0CONSTOP0 处于睡眠模式。
- (3) 在停止模式和睡眠模式，GPO/GPP/GPQ 端口均设置为存储器功能。

### 10.2.16 端口 P 控制寄存器

端口 P 控制寄存器包括五个控制寄存器，分别是 GPPCON 、GPPDAT 、GPPPUD、GPPCONSLP、GPPPUDSLP。

寄存器	地址	读/写	描述	复位值
GPPCON	0x7F008160	读/写	端口 P 配置寄存器	0x2AAAAAAAA
GPPDAT	0x7F008164	读/写	端口 P 数据寄存器	未定义
GPPPUD	0x7F008168	读/写	端口 P 上拉寄存器	0x1011AAA0
GPPCONSLP	0x7F00816C	读/写	端口 P 睡眠模式配置寄存器	0x0
GPPPUDSLP	0x7F008170	读/写	端口 P 睡眠模式上拉/下拉寄存器	0x0

GPPCON	位	描述	初始状态
GPP0	[1:0]	00=输入                      01=输出 10=MEM0_ADDRV            11= Ext.Interrupt Group8[0]	10
GPP1	[3:2]	00=输入                      01=输出 10=MEM0_SMCKL            11= Ext.Interrupt Group8[1]	10
GPP2	[5:4]	00=输入                      01=输出 10=MEM0_nWAIT            11= Ext.Interrupt Group8[2]	10
GPP3	[7:6]	00=输入                      01=输出	10

		10= MEM0_RDY0_ALE    11= Ext.Interrupt Group8[3]	
GPP4	[9:8]	00=输入                      01=输出 10= MEM0_RDY1_CLE    11= Ext.Interrupt Group8[4]	10
GPP5	[11:10]	00=输入                      01=输出 10= MEM0_INTsm0-FEW    11= Ext.Interrupt Group8[5]	10
GPP6	[13:12]	00=输入                      01=输出 10=MEM0_ INTsm0-FRE    11= Ext.Interrupt Group8[6]	10
GPP7	[15:14]	00=输入                      01=输出 10=MEM0_ RPN_RnB        11= Ext.Interrupt Group8[7]	10
GPP8	[17:16]	00=输入                      01=输出 10=MEM0_INTata          11= Ext.Interrupt Group8[8]	10
GPP9	[19:18]	00=输入                      01=输出 10=MEM0_ RESETata       11= Ext.Interrupt Group8[9]	10
GPP10	[21:20]	00=输入                      01=输出 10=MEM0_INPACKata       11= Ext.Interrupt Group8[10]	10
GPP11	[23:22]	00=输入                      01=输出 10=MEM0_REGata          11= Ext.Interrupt Group8[11]	10
GPP12	[25:24]	00=输入                      01=输出 10=MEM0_WEata            11= Ext.Interrupt Group8[12]	10
GPP13	[27:26]	00=输入                      01=输出 10=MEM0_Oeata            11= Ext.Interrupt Group8[13]	10
GPP14	[29:28]	00=输入                      01=输出 10=MEM0_CDData          11= Ext.Interrupt Group8[14]	10

GPPDAT	位	描述
GPP[14:0]	[14:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应位的状态相同。当端口作为功能管脚时，读取未被定义的值

GPPPUD	位	描述
GPP[n]	[2n+1: 2n] n=0~14	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

GPPCONSLP	位	描述	初始状态
GPP[n]	[2n+1: 2n] n=0~14	00=输出 0 01=输出 1 10=输入 11=与先前状态相同	00

GPPPUDSLP	位	描述
GPP[n]	[2n+1: 2n] n=0~14	00=禁止上拉/下拉 01=下拉使能 10=上拉使能 11=保留

注意：

- （1）当端口被设置为存储器接口信号时，它们的状态由停止模式的 MEM0CONSTOP 控制，MEM0CONSTOP1 处于睡眠模式。
- （2）在停止模式和睡眠模式下，GPO/GPP/GPQ 端口均设置为存储器功能。

### 10.2.17 端口 Q 控制寄存器

端口 Q 控制寄存器包括五个控制寄存器，分别是 GPQCON 、 GPQDAT 、 GPQPUD、GPQCONSLP、GPQPUDSLP。

寄存器	地址	读/写	描述	复位值
GPQCON	0x7F008180	读/写	端口 Q 配置寄存器	0x0002AAAA

GPQDAT	0x7F008184	读/写	端口 Q 数据寄存器	未定义
GPQPUD	0x7F008188	读/写	端口 Q 上拉寄存器	0x0
GPQCONSLP	0x7F00818C	读/写	端口 Q 睡眠模式配置寄存器	0x0
GPQPUDSLP	0x7F008190	读/写	端口 Q 睡眠模式上拉/下拉寄存器	0x0

GPQCON	位	描述	初始状态
GPQ0	[1:0]	00=输入 10=MEM0_ADDRV18_RAS 01=输出 11= Ext.Interrupt Group9[0]	10
GPQ1	[3:2]	00=输入 10=MEM0_ADDR19_RAS 01=输出 11= Ext.Interrupt Group9[1]	10
GPQ2	[5:4]	00=输入 10=保留 01=输出 11= Ext.Interrupt Group9[2]	10
GPQ3	[7:6]	00=输入 10= 保留 01=输出 11= Ext.Interrupt Group9[3]	10
GPQ4	[9:8]	00=输入 10=保留 01=输出 11= Ext.Interrupt Group9[4]	10
GPQ5	[11:10]	00=输入 10=保留 01=输出 11= Ext.Interrupt Group9[5]	10
GPQ6	[13:12]	00=输入 10=保留 01=输出 11= Ext.Interrupt Group9[6]	10
GPQ7	[15:14]	00=输入 10=MEM0_ADDR17_WEndmc 01=输出 11= Ext.Interrupt Group9[7]	10
GPQ8	[17:16]	00=输入 10=MEM0_ADDR16_APdmc 01=输出 11= Ext.Interrupt Group9[8]	10

GPQDAT	位	描述
GPQ[8:0]	[8:0]	当端口作为输入端口时，相应的位处于管脚状态，当端口作为输出端口时，管脚状态于相应位的状态相同。当端口作为功能管脚时，读取未被定义的值



GPQPUD	位	描述
GPQ[n]	[2n+1: 2n] n=0~8	00=禁止上拉/下拉  01=下拉使能  10=上拉使能  11=保留

GPQCONSLP	位	描述	初始状态
GPQ[n]	[2n+1: 2n] n=0~8	00=输出 0  01=输出 1  10=输入  11=与先前状态相同	00

GPQPUDSLP	位	描述
GPQ[n]	[2n+1: 2n] n=0~8	00=禁止上拉/下拉  01=下拉使能  10=上拉使能  11=保留

注意：

- (1) 当端口被设置为内存储器接口信号时，它们的状态由停止模式的 MEM0CONSTOP 控制，MEM0CONSTOP0 处于睡眠模式。
- (2) 当端口 GPQ[4:0]和端口 GPQ[8:7]被设置为存储器接口信号时，上拉/下拉失效。
- (3) 当单口 GPQ[6:5]被设置为存储器接口信号时，上拉、下拉由 SPCON[11:10]控制。

### 10.2.18 特殊端口控制寄存器

寄存器	地址	读/写	描述	复位值
SPCON	0x7F0081A0	读/写	特殊端口控制寄存器	0xBFC11500

SPCON	位	描述	初始状态
DRVCON_CAM	[31:30]	CAMERA 端口驱动 00=2mA 01=4 mA 10=7 mA 11=9 mA	10
DRVCON_HSSPI	[29:28]	HSSPI 端口驱动 00=2mA 01=4 mA 10=7 mA 11=9 mA	11
DRVCON_HSMMC	[27:26]	HSMMCI 端口驱动 00=2mA 01=4 mA 10=7 mA 11=9 mA	11
DRVCON_LCD	[25:24]	LCDI 端口驱动 00=2mA 01=4 mA 10=7 mA 11=9 mA	11
DRVCON_MODEM	[23:22]	MODEM 端口驱动 00=2mA 01=4 mA 10=7 mA 11=9 mA	11
Reserved	[21]	保留	0
nRSTOUT_OEM	[20]	复位输出管脚（XnRSTOUT）输出使能 0=使能 1=不能	0
DRVCON_SPICK1	[19:18]	SPICK1[1]端口驱动 00=2mA 01=4 mA 10=7 mA 11=9 mA	00
MEM1_DQS_PUD1	[17:16]	存储器端口 1DWA 管脚上拉/下拉控制 00=不能 01=下拉 10=上拉 11=保留	01
MEM1_DQS_PUD0	[13:12]	存储器端口 1 数据管脚[31:16]上拉/下拉控制 00=不能 01=下拉 10=上拉 11=保留	01
Reserved	[11: 10]	保留	-
MEM1_DQS_PUD	[9:8]	存储器端口 0 数据管脚[31:16]上拉/下拉控制 00=不能 01=下拉 10=上拉 11=保留	01
USBH_DMPD	[7]	USB_Host DP 下拉控制 0=disable 1=enable	0
USBH_DPPD	[6]	USB_Host DM 下拉控制	0

		0=disable      1=enable	
USBH_PUSW2	[5]	当 USBH_PUSW1 处于 ON 状态是 USB_Host Pull-up 开关 2 是可控制的。 0=off      1=on	0
USBH_PUSW1	[4]	USB_Host Pull-up 开关 2 控制 0=off      1=on	0
USBH_SUSPND	[3]	使 USB 收发接口 PAD 进入暂停模式	0
Reserved	[2]	保留	0
LCD_SEL	[1:0]	选择 LCD I/F 管脚组态 00=Host I/F 形态      01=RGB I/F 形态 10=601/656 形态      11=保留	00

注意：

上拉电阻是 1.2K $\Omega$ ，0.5K $\Omega$  ,下拉电阻是 20K $\Omega$  .参见图 10-2。

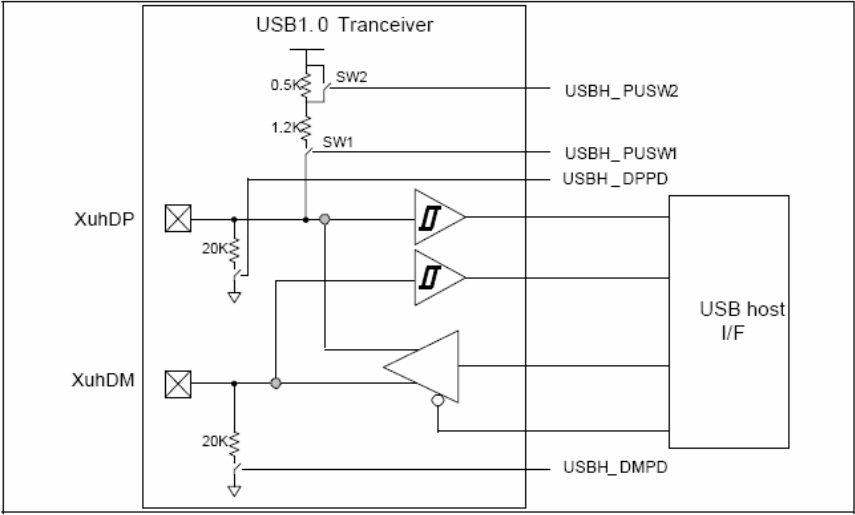


图 10-2 USB 收发器模块图

管脚	LCD_SEL[1:0]			
	00	01	10	11
XvVD[6:0]	XvSYS_VD[6:0]	XvRGBVD[6:0]	VEN_DATA[6:0]	

XvVD[7]	XvSYS_VD[7]	XvRGBVD[7]	VEN_DATA[7]	保留
XvVD[13:8]	XvSYS_VD[13:8]	XvRGBVD[13:8]	V656_DATA[5:0]	
XvVD[15:14]	XvSYS_VD[15:14]	XvRGBVD[15:14]	V656_DATA[7:6]	
XvVD[17:16]	XvSYS_VD[17:16]	XvRGBVD[17:16]		
XvVD[20:18]		XvRGBVD[20:18]		
XvVD[21]		XvRGBVD[21]		
XvVD[22]	XvSYS_VSYNC_ldi	XvRGBVD[22]	V656_CLK	
XvVD[23]	XvSYS_OEn	XvRGBVD[23]	VEN_FIELD	
XvHSYNC	XvSYS_CSn_main	XvHSYNC	VEN_HSYNC	
XvVSYNC	XvSYS_CSn_sub	XvVSYNC	VEN_VSYNC	
XvVDEN	XvSYS_RS	XvVDEN	VEN_HREF	
XvVCLK	XvSYS_WEn	XvVCLK	V601_CLK	

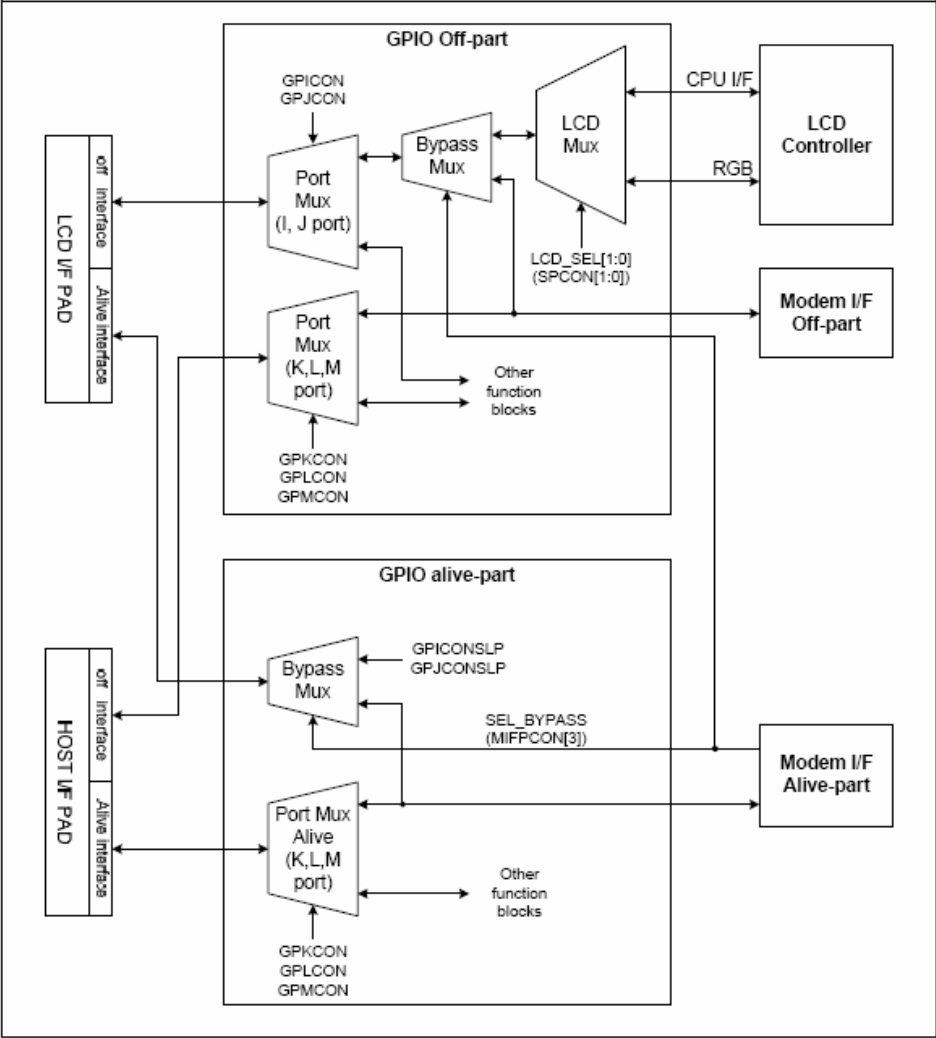


图 4-8 LCD 旁路逻辑图

10.2.19 停止模式的存储器接口管脚配置寄存器

寄存器	地址	读/写	描述	复位值
MEM0CONSTOP	0x7F0081B0	读/写	存储器端口 0 配置寄存器	0x0
MEM1CONSTOP	0x7F0081B4	读/写	存储器端口 1 配置寄存器	0x0

MEM0CONSTOP	位	描述	初始状态
-------------	---	----	------

Reserved	[31:29]	保留	000
MEM0_RESET	[28]	存储器端口 0 RESET 管脚 (xm0RESET) 配置 0=先前状态 1=Hi-Z	0
MEM0_RP	[27]	存储器端口 0 RP 管脚 (xm0RP) 配置 0=先前状态 1=Hi-Z	0
MEM0_ADDRVLD	[26]	存储器端口 0 ADDRVLD 管脚 (Xm0ADDRVLD) 配置 0=先前状态 1=Hi-Z	0
MEM0_FREn	[25]	存储器端口 0 FREn 管脚 (Xm0 FREn) 配置 0=先前状态 1=Hi-Z	0
MEM0_FWEn	[24]	存储器端口 0 FWEn 管脚 (Xm0 FWEn) 配置 0=先前状态 1=Hi-Z	0
MEM0_CLE	[23]	存储器端口 0 CLE 管脚 (Xm0CLE) 配置 0=先前状态 1=Hi-Z	0
MEM0_ALE	[22]	存储器端口 0 ALE 管脚 (Xm0ALE) 配置 0=先前状态 1=Hi-Z	0
MEM0_OEn	[28]	存储器端口 0 输出使能管脚 (Xm0OEn) 配置 0=先前状态 1=Hi-Z	0
Reserved	[20:17]	保留	-
MEM0_A	[16]	存储器端口 0 地址管脚 (Xm0ADDR) 配置 0=先前状态 1=Hi-Z	0
Reserved	[15:14]	保留	-
MEM0_WEn	[13]	存储器端口 0 读使能管脚 (Xm0WEn) 配置 0=先前状态 1=Hi-Z	0
MEM0_CSn	[13]	存储器端口 0 ChipSelet 能管脚 (Xm0CSEn) 配置 0=先前状态 1=Hi-Z	0
Reserved	[12:5]	保留	-
MEM0_SMCLK	[4]	存储器端口 0 SSMCS 时钟管脚 (Xm0SMCLK) 配置 0=先前状态 1=Hi-Z	0
Reserved	[3:0]	保留	-

MEM1CONSTOP	位	描述	初始状态
Reserved	[31:21]	保留	0x0000
MEM1_SLCKn	[20]	存储器端口 1 SLCKn 管脚 (Xm1 SLCKn) 配置 0=先前状态 1=Hi-Z	0
MEM1_SLCK	[19]	存储器端口 1 SLCK 管脚 (Xm1SLCK) 配置 0=先前状态 1=Hi-Z	0
MEM1_CKE	[18]	存储器端口 0 CKE 管脚 (Xm1 CKE) 配置 0=先前状态 1=Hi-Z	0
MEM1_DQM	[17]	存储器端口 1 DQM 管脚 (Xm1DQM) 配置 0=先前状态 1=Hi-Z	0
MEM1_A	[16]	存储器端口 1 地址管脚 (Xm1 ADDR) 配置 0=先前状态 1=Hi-Z	0
MEM1_CASn	[15]	存储器端口 1 CAS 管脚 (Xm1 CASn) 配置 0=先前状态 1=Hi-Z	0
MEM1_RASn	[14]	存储器端口 1 RAS 管脚 (Xm1RASn) 配置 0=先前状态 1=Hi-Z	0
MEM1_WEn	[13]	存储器端口 1 读使能管脚 (Xm1 WEn) 配置 0=先前状态 1=Hi-Z	0
MEM1_CSn	[12]	存储器端口 1 Chip Select 管脚 (Xm1CSn) 配置 0=先前状态 1=Hi-Z	0
Reserved	[11:0]	保留	-

## 10.2.20 睡眠模式的存储器接口管脚配置寄存器

寄存器	地址	读/写	描述	复位值
MEM0CONSLP0	0x7F0081C0	读/写	存储器端口 0 管脚配置寄存器 0	0x0

MEM1CONSLP1	0x7F0081C4	读/写	存储器端口 0 管脚配置寄存器 1	0x0
MEM1CONSLP	0x7F0081C4	读/写	存储器端口 1 管脚配置寄存器	0x0

MEM0CONSLP0	位	描述	初始状态
Reserved	[31:22]	保留	00
MEM0_A	[21:20]	存储器端口 0 地址管脚 (Xm0ADDR) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
Reserved	[19:16]	保留	00
MEM0_WEn	[15:14]	存储器端口 0 写使能管脚 (Xm0WEn) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_CS <sub>n</sub>	[13:12]	存储器端口 0 芯片选择管脚 (Xm0CS <sub>n</sub> ) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_DQS	[11:8]	存储器端口 0 DQS 管脚 (Xm0DQS) 配置  0000=输出 0                      0001=输出 1 0100=输入(hi-Z)                  0101=输入下拉电阻使能 0110=输入上拉电阻使能        0111=未用 10xx=先前状态	0000
Reserved	[7: 6]	保留	00
MEM0_AP	[5:4]	存储器端口 0 AP 管脚 (Xm0AP) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_D	[11:8]	存储器端口 0 数据管脚 (Xm0DATA) 配置  0000=输出 0                      0001=输出 1 0100=输入(hi-Z)                  0101=输入下拉电阻使能 0110=输入上拉电阻使能        0111=未用 10xx=先前状态	0000

MEM0CONSLP1	位	描述	初始状态
Reserved	[31:26]	保留	00



MEM0_nOEata	[25:24]	ATA I/F 输出使能管脚 (Xm0 nOEata) 配置 00=输出 0    01=输出 1    1x=输入(pull-up)	00
MEM0_nWEata	[23:22]	ATA I/F 写使能管脚 (Xm0 nWEata) 配置 00=输出 0    01=输出 1    1x=输入(pull-up)	00
MEM0_SMCLK	[21:20]	ROM 时钟管脚 (Xm0SMCLK) 配置 00=输出 0    01=输出 1    1x=输入(pull-up)	00
MEM0_WAIT	[19:18]	ROM 时钟管脚 (Xm0 WAITn) 配置 00=输出 0    01=输出 1    1x=输入(pull-up)	00
MEM0_REGata	[17:16]	Nand Flash RnB 管脚 (Xf REGata) 配置 00=输出 0    01=输出 1    1x=输入(hi-Z)	00
MEM0_RESEata	[15:14]	存储器端口 0 RESET 管脚 (Xm0 RESEata) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_RP_RnB	[13:12]	存储器端口 0 RP 管脚 (Xm0 RP) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_ADDRVLD	[11:10]	存储器端口 0 ADDRVLD 管脚 (Xm0 ADDRVLD) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_INTsm1_FREn	[9:8]	存储器端口 0 FREn 管脚 (Xm0 INTsm1_FREn) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_INTsm0_FWEEn	[7:6]	存储器端口 0 FWEEn 管脚 (Xm0INTsm0_FWEEn) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_RDY0_CLE	[5:4]	存储器端口 0 CLE 管脚 (Xm0RDY0_CLE) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_RDY1_ALE	[3:2]	存储器端口 0 ALE 管脚 (Xm0RDY1_ALE) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_OEn	[1:0]	存储器端口 0 输出使能管脚 (Xm0OEn) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00

MEM1CONSLP	位	描述	初始状态
Reserved	[31:30]	保留	00

MEM0_SCLKn	[29:28]	存储器端口 1 SCLKn 管脚 (Xm1 SCLKn) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_SCLK	[27:26]	存储器端口 1 SCLK 管脚 (Xm1 SCLK) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_CKE	[25:24]	存储器端口 1 CKE 管脚 (Xm1CKE) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_DQM	[23:22]	存储器端口 1 DQM 管脚 (Xm1DQM) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_A	[21:20]	存储器端口 1 地址管脚 (Xm1ADDR) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_CASn	[19:18]	存储器端口 1 CAS 管脚 (Xm1 CASn) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_RASn	[17:16]	存储器端口 1 RAS 管脚 (Xm1 RASn) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_WEn	[15:14]	存储器端口 1 写使能管脚 (Xm1 WEn) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_CSn	[13:12]	存储器端口 1 Chip 选择管脚 (Xm1 SCLKn) 配置 00=输出 0    01=输出 1    1x=输出禁止(hi-Z)	00
MEM0_DQS	[29:28]	存储器端口 1 DQS 管脚 (Xm1DQS) 配置  0000=输出 0                      0001=输出 1 0100=输入(hi-Z)                  0101=输入下拉电阻使能 0110=输入上拉电阻使能        0111=未使用 10xx=先前状态	0000

10.2.21 存储器接口驱动控制寄存器

寄存器	地址	读/写	描述	复位值
MEM0DRVCON	0x7F0081D0	读/写	存储器端口 0 驱动控制寄存器	0x10555551
MEM1DRVCON	0x7F0081D4	读/写	存储器端口 0 驱动控制寄存器	0x0

MEM0DRVCON	位	描述	初始状态
MEM0_CF	[31:30]	存储器端口 0 CF 管脚（Xm0INTata, Xm0RESETata, Xm0REGTata, Xm0Weata, Xm0Oeata, Xm0Cdata）配置	00
MEM0_ADDRVLD_RP	[29:28]	存储器端口 0 ADDRVLDRP 管脚（Xm0 ADDRVLDRP）配置	01
MEM0_FWE_FRE	[27:26]	存储器端口 0 FWEn、FREn 管脚（Xm0 FWEn, Xm0FREn）配置	00
MEM0_AWE_CLE	[25:24]	存储器端口 0 CLE、ALE 管脚（Xm0CLE, Xm0ALE）配置	00
Reserved	[23:16]	保留	-
MEM0_A	[15:14]	存储器端口 0 地址管脚（Xm0ADDR）配置	01
MEM0_BEn	[13:12]	存储器端口 0 BEn 管脚（Xm0 BEn）配置	01
MEM0_WEn_OEn	[11:10]	存储器端口 0 写使能、输出使能管脚（Xm0WEn, Xm0OEn）配置	01
Reserved	[9:8]	保留	01
MEM0_CSn4_5	[7:6]	存储器端口 0 芯片选择管脚（Xm0CSn[5:4]）配置	01
MEM0_CSn0_3	[5:4]	存储器端口 0 芯片选择管脚（Xm0CSn[3:0]）配置	01
MEM0_NAND	[3:2]	存储器端口 0 NAND 管脚（Xm0SMCLK, Xm0DRY0_ALE, Xm0DRY1_CLE, Xm0INTsm0_FWEn, Xm0INTsm1_FREn, Xm0RPn_RnB）配置	00
MEM0_D15_0	[1:0]	存储器端口 0 数据管脚（Xm0DATA[15:0]）配置	00

MEM0DRVCON	位	描述
	[2n+1:2n] N=0~11	在 VDDmem0=1.8v 的情况下 00=3mA 01=7 mA 10=10 mA 11=14 mA 在 VDDmem0=2.5v 的情况下 00=5mA 01=10 mA 10=15 mA 11=20 mA

MEM1DRVCON	位	描述	初始状态
Reserved	[31: 24]	保留	0x00
MEM1_SCLKn	[23:22]	存储器端口 1 SCLKn 管脚 (Xm1SCLKn) 配置	00
MEM1_DQS	[21:20]	存储器端口 1 DQS 管脚 (Xm1DQS) 配置	00
MEM1_CKE	[19:18]	存储器端口 1 CKE 管脚 (Xm1CKE) 配置	00
MEM1_SCLK	[17:16]	存储器端口 1 SCLK 管脚 (Xm1SCLK) 配置	00
MEM1_A	[15:14]	存储器端口 1 地址管脚 (Xm1ADDR) 配置	00
MEM1_DQM	[13:12]	存储器端口 1 DQM 管脚 (Xm1DQM) 配置	00
MEM1_WEn	[11:10]	存储器端口 1 写使能管脚 (Xm1WEn) 配置	00
MEM1_RASn_CASn	[9:8]	存储器端口 1RAS, CAS 管脚 (Xm1 RASn, Xm1 RASn) 配置	00
MEM1_CSn1	[7:6]	存储器端口 1 芯片选择管脚 (Xm1 CSn[1]) 配置	00
MEM1_CSn0	[5:4]	存储器端口 1 芯片选择管脚 (Xm1 CSn[0]) 配置	00
MEM1_D31_16	[3:2]	存储器端口 1 数据管脚 (Xm1DATA[31:16]) 配置	00
MEM1_D15_0	[1:0]	存储器端口 1 数据管脚 (Xm1DATA[15:0]) 配置	00

MEM1DRVCON	位	描述
	[2n+1:2n] N=0~11	在 VDDmem1=1.8v 的情况下  00=3mA  01=7 mA  10=10 mA  11=14 mA  在 VDDmem1=2.5v 的情况下  00=5mA  01=10 mA  10=15 mA  11=20 mA

10.2.22 外部中断控制寄存器

外部中断由 0-9 十个群组组成。只有外部中断组 0 用于停止模式和睡眠模式下的唤醒源。在空闲模式，所有的中断都可以作为唤醒源。

下面的表格是外部中断控制寄存器的清单。0 组有专用的管脚，0 组的中断可以比其它组的中断控制更多的内容。S3C6410 显示出 0 组的多个寄存器以及每对中断信号所占的位。其他组同样有多个寄存器，每个寄存器可以控制 2 个组或多个组。0 组数字滤波器计数脉冲源是 FIN，其它组的数字滤波器计数脉冲源是 PCLK.

寄存器	地址	读/写	描述	复位值
EINT0CON0	0x7F008900	读/写	外部中断 0（0 组）配置寄存器 0	0x0
EINT0CON1	0x7F008904	读/写	外部中断 0（0 组）配置寄存器 1	0x0
EINT0FLTCON0	0x7F008910	读/写	外部中断 0（0 组）过滤控制寄存器 0	0x0
EINT0FLTCON1	0x7F008914	读/写	外部中断 0（0 组）过滤控制寄存器 1	0x0
EINT0FLTCON2	0x7F008918	读/写	外部中断 0（0 组）过滤控制寄存器 2	0x0
EINT0FLTCON3	0x7F00891C	读/写	外部中断 0（0 组）过滤控制寄存器 3	0x0

EINT0MASK	0x7F008920	读/写	外部中断 0（0 组）屏蔽寄存器	0x0FFFFFFF
EINT0PEND	0x7F008924	读/写	外部中断 0（0 组）悬挂寄存器	0x0
EINT12CON	0x7F008200	读/写	外部中断 1、2（1、2 组）配置寄存器	0x0
EINT34CON	0x7F008204	读/写	外部中断 3、4（3、4 组）配置寄存器	0x0
EINT56CON	0x7F008208	读/写	外部中断 5、6（5、6 组）配置寄存器	0x0
EINT78CON	0x7F00820C	读/写	外部中断 7、8（7、8 组）配置寄存器	0x0
EINT9CON	0x7F008210	读/写	外部中断 9（9 组）配置寄存器	0x0
EINT12FLTCON	0x7F008220	读/写	外部中断 1、2（1、2 组）过滤控制寄存器	0x0
EINT34FLTCON	0x7F008224	读/写	外部中断 3、4（3、4 组）过滤控制寄存器	0x0
EINT56FLTCON	0x7F008228	读/写	外部中断 5、6（5、6 组）过滤控制寄存器	0x0
EINT78FLTCON	0x7F00822C	读/写	外部中断 7、8（7、8 组）过滤控制寄存器	0x0
EINT9FLTCON	0x7F008230	读/写	外部中断 9（9 组）过滤控制寄存器 0	0x0
EINT12MASK	0x7F008240	读/写	外部中断 1、2（1、2 组）屏蔽寄存器	0x00FF7FFF
EINT34 MASK	0x7F008244	读/写	外部中断 3、4（3、4 组）屏蔽寄存器	0x3FFF03FF
EINT56 MASK	0x7F008248	读/写	外部中断 5、6（5、6 组）屏蔽寄存器	0x03FF007F
EINT78 MASK	0x7F00824C	读/写	外部中断 7、8（7、8 组）屏蔽寄存器	0x7FFFFFFF
EINT9 MASK	0x7F008250	读/写	外部中断 9（9 组）屏蔽寄存器	0x000001FF
EINT12PEND	0x7F008260	读/写	外部中断 1、2（1、2 组）悬挂寄存器	0x0
EINT34 PEND	0x7F008264	读/写	外部中断 3、4（3、4 组）悬挂寄存器	0x0
EINT56 PEND	0x7F008268	读/写	外部中断 5、6（5、6 组）悬挂寄存器	0x0
EINT78 PEND	0x7F00826C	读/写	外部中断 7、8（7、8 组）悬挂寄存器	0x0
EINT9 PEND	0x7F008270	读/写	外部中断 9（9 组）悬挂寄存器	0x0
PRIORITY	0x7F008280	读/写	优先控制寄存器	0x3FF
SERVICE	0x7F008284	读	当前服务寄存器	0x0

SERVICEPEND	0x7F008288	读/写	当前服务悬挂寄存器	0x0
-------------	------------	-----	-----------	-----

EINT0CON0	位	描述	初始状态
Reserved	[31]	保留	0
EINT15、14	[30:28]	设置 EINT15 和 EINT14 的信令方法 000=低电平            001=高电平 01x=边沿下降触发    10x=边沿上升触发 11x=边沿触发	0000
Reserved	[27]	保留	0
EINT13、12	[26:24]	设置 EINT13 和 EINT12 的信令方法 000=低电平            001=高电平 01x=边沿下降触发    10x=边沿上升触发 11x=边沿触发	0000
Reserved	[23]	保留	0
EINT11、10	[22:20]	设置 EINT11 和 EINT10 的信令方法 000=低电平            001=高电平 01x=边沿下降触发    10x=边沿上升触发 11x=边沿触发	0000
Reserved	[19]	保留	0
EINT9、8	[18:16]	设置 EINT9 和 EINT8 的信令方法 000=低电平            001=高电平 01x=边沿下降触发    10x=边沿上升触发 11x=边沿触发	0000
Reserved	[15]	保留	0
EINT7、6	[14:12]	设置 EINT7 和 EINT6 的信令方法 000=低电平            001=高电平 01x=边沿下降触发    10x=边沿上升触发 11x=边沿触发	0000
Reserved	[11]	保留	0

EINT5、4	[10:8]	设置 EINT5 和 EINT4 的信令方法 000=低电平            001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[7]	保留	0
EINT3、2	[6:4]	设置 EINT3 和 EINT2 的信令方法 000=低电平            001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[3]	保留	0
EINT1、0	[2:0]	设置 EINT1 和 EINT0 的信令方法 000=低电平            001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000

EINT0CON1	位	描述	初始状态
Reserved	[31:23]	保留	0
EINT27、26	[22:20]	设置 EINT27 和 EINT26 的信令方法 000=低电平            001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[19]	保留	0
EINT25、24	[18:16]	设置 EINT13 和 EINT12 的信令方法 000=低电平            001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[15]	保留	0
EINT23、22	[14:12]	设置 EINT23 和 EINT22 的信令方法 000=低电平            001=高电平	0000



		01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	
Reserved	[11]	保留	0
EINT21、20	[10:8]	设置 EINT21 和 EINT20 的信令方法  000=低电平      001=高电平  01x=边沿下降触发      10x=边沿上升触发  11x=边沿触发	0000
Reserved	[7]	保留	0
EINT19、18	[6:4]	设置 EINT19 和 EINT18 的信令方法  000=低电平      001=高电平  01x=边沿下降触发      10x=边沿上升触发  11x=边沿触发	0000
Reserved	[3]	保留	0
EINT17、16	[2:0]	设置 EINT17 和 EINT16 的信令方法  000=低电平      001=高电平  01x=边沿下降触发      10x=边沿上升触发  11x=边沿触发	0000

EINT0FLTCON0	位	描述	初始状态
FLTEN	[31]	EINT6、7 过滤器使能  0=禁止      1=使能	0
FLTSEL	[30]	EINT6、7 滤波器选择  0=延迟滤波器      1=数字滤波器	0
EINT6、7	[29:24]	EINT6、7 过滤  当 FLTSEL 值为 1 时有效。	000
FLTEN	[23]	EINT4、5 过滤器使能  0=禁止      1=使能	0
FLTSEL	[22]	EINT4、5 滤波器选择  0=延迟滤波器      1=数字滤波器	0

EINT4、5	[21:16]	EINT4、5 过滤 当 FLTSEL 值为 1 时有效。	000
FLTEN	[15]	EINT2、3 过滤器使能 0=禁止 1=使能	0
FLTSEL	[14]	EINT2、3 滤波器选择 0=延迟滤波器 1=数字滤波器	0
EINT2、3	[13:8]	EINT2、3 过滤 当 FLTSEL 值为 1 时有效。	000
FLTEN	[7]	EINT0、1 过滤器使能 0=禁止 1=使能	0
FLTSEL	[6]	EINT0、1 滤波器选择 0=延迟滤波器 1=数字滤波器	0
EINT0、1	[5:0]	EINT0、1 过滤 当 FLTSEL 值为 1 时有效。	000

EINT0FLTCON1	位	描述	初始状态
FLTEN	[31]	EINT14、15 过滤器使能 0=禁止 1=使能	0
FLTSEL	[30]	EINT14、15 滤波器选择 0=延迟滤波器 1=数字滤波器	0
EINT14、15	[29:24]	EINT14、15 过滤 当 FLTSEL 值为 1 时有效。	000
FLTEN	[23]	EINT12、13 过滤器使能 0=禁止 1=使能	0
FLTSEL	[22]	EINT12、13 滤波器选择 0=延迟滤波器 1=数字滤波器	0
EINT12、13	[21:16]	EINT12、13 过滤 当 FLTSEL 值为 1 时有效。	000
FLTEN	[15]	EINT10、11 过滤器使能	0

		0=禁止    1=使能	
FLTSEL	[14]	EINT10、11 滤波器选择 0=延迟滤波器                      1=数字滤波器	0
EINT10、11	[13:8]	EINT10、11 过滤 当 FLTSEL 值为 1 时有效。	000
FLTEN	[7]	EINT8、9 过滤器使能 0=禁止    1=使能	0
FLTSEL	[6]	EINT8、9 滤波器选择 0=延迟滤波器                      1=数字滤波器	0
EINT8、9	[5:0]	EINT8、9 过滤 当 FLTSEL 值为 1 时有效。	000

EINT0FLTCON2	位	描述	初始状态
FLTEN	[31]	EINT22、23 过滤器使能 0=禁止    1=使能	0
FLTSEL	[30]	EINT22、23 滤波器选择 0=延迟滤波器                      1=数字滤波器	0
EINT22、23	[29:24]	EINT22、23 过滤 当 FLTSEL 值为 1 时有效。	000
FLTEN	[23]	EINT20、21 过滤器使能 0=禁止    1=使能	0
FLTSEL	[22]	EINT20、21 滤波器选择 0=延迟滤波器                      1=数字滤波器	0
EINT20、21	[21:16]	EINT20、21 过滤 当 FLTSEL 值为 1 时有效。	000
FLTEN	[15]	EINT18、19 过滤器使能 0=禁止    1=使能	0
FLTSEL	[14]	EINT18、19 滤波器选择 0=延迟滤波器                      1=数字滤波器	0



EINT22	[22]	0=使中断	1=屏蔽	1
EINT21	[21]	0=使中断	1=屏蔽	1
EINT20	[20]	0=使中断	1=屏蔽	1
EINT19	[19]	0=使中断	1=屏蔽	1
EINT18	[18]	0=使中断	1=屏蔽	1
EINT17	[17]	0=使中断	1=屏蔽	1
EINT16	[16]]	0=使中断	1=屏蔽	1
EINT15	[15]	0=使中断	1=屏蔽	1
EINT14	[14]	0=使中断	1=屏蔽	1
EINT13	[13]	0=使中断	1=屏蔽	1
EINT12	[12]	0=使中断	1=屏蔽	1
EINT11	[11]	0=使中断	1=屏蔽	1
EINT10	[10]	0=使中断	1=屏蔽	1
EINT9	[9]	0=使中断	1=屏蔽	1
EINT8	[8]	0=使中断	1=屏蔽	1
EINT7	[7]	0=使中断	1=屏蔽	1
EINT6	[6]]	0=使中断	1=屏蔽	1
EINT5	[5]	0=使中断	1=屏蔽	1
EINT4	[4]	0=使中断	1=屏蔽	1
EINT3	[3]	0=使中断	1=屏蔽	1
EINT2	[2]	0=使中断	1=屏蔽	1
EINT1	[1]	0=使中断	1=屏蔽	1
EINT0	[0]	0=使中断	1=屏蔽	1

EINT0PEND	位	描述		初始状态
EINT27	[27]	0=不发生中断	1=发生中断	1
EINT26	[26]]	0=不发生中断	1=发生中断	1
EINT25	[25]	0=不发生中断	1=发生中断	1

EINT24	[24]	0=不发生中断	1=发生中断	1
EINT23	[23]	0=不发生中断	1=发生中断	1
EINT22	[22]	0=不发生中断	1=发生中断	1
EINT21	[21]	0=不发生中断	1=发生中断	1
EINT20	[20]	0=不发生中断	1=发生中断	1
EINT19	[19]	0=不发生中断	1=发生中断	1
EINT18	[18]	0=不发生中断	1=发生中断	1
EINT17	[17]	0=不发生中断	1=发生中断	1
EINT16	[16]]	0=不发生中断	1=发生中断	1
EINT15	[15]	0=不发生中断	1=发生中断	1
EINT14	[14]	0=不发生中断	1=发生中断	1
EINT13	[13]	0=不发生中断	1=发生中断	1
EINT12	[12]	0=不发生中断	1=发生中断	1
EINT11	[11]	0=不发生中断	1=发生中断	1
EINT10	[10]	0=不发生中断	1=发生中断	1
EINT9	[9]	0=不发生中断	1=发生中断	1
EINT8	[8]	0=不发生中断	1=发生中断	1
EINT7	[7]	0=不发生中断	1=发生中断	1
EINT6	[6]]	0=不发生中断	1=发生中断	1
EINT5	[5]	0=不发生中断	1=发生中断	1
EINT4	[4]	0=不发生中断	1=发生中断	1
EINT3	[3]	0=不发生中断	1=发生中断	1
EINT2	[2]	0=不发生中断	1=发生中断	1
EINT1	[1]	0=不发生中断	1=发生中断	1
EINT0	[0]	0=不发生中断	1=发生中断	1

EINT12CON	位	描述	初始状态
Reserved	[31:23]	保留	0

EINT2[7:4]	[22:20]	设置 EINT2[7:4]的信令方法 000=低电平            001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[19]	保留	0
EINT2[3:0]	[18:16]	设置 EINT2[3:0]的信令方法 000=低电平            001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[15]	保留	0
EINT1[14:12]	[14:12]	设置 EINT1[14:12]的信令方法 000=低电平            001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[11]	保留	0
EINT1[11:8]	[10:8]	设置 EINT1[11:8]的信令方法 000=低电平            001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[7]	保留	0
EINT1[7:4]	[6:4]	设置 EINT1[7:4]的信令方法 000=低电平            001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[3]	保留	0
EINT1[3:0]	[2:0]	设置 EINT1[3:0]的信令方法 000=低电平            001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000

EINT34CON	位	描述	初始状态
Reserved	[31]	保留	0
EINT4[13:12]	[30:28]	设置 EINT4[13:12]的信令方法 000=低电平            001=高电平 01x=边沿下降触发    10x=边沿上升触发 11x=边沿触发	0000
Reserved	[27]	保留	0
EINT4[11:8]	[26:24]	设置 EINT4[11:8]的信令方法 000=低电平            001=高电平 01x=边沿下降触发    10x=边沿上升触发 11x=边沿触发	0000
Reserved	[23]	保留	0
EINT4[7:4]	[22:20]	设置 EINT4[7:4]的信令方法 000=低电平            001=高电平 01x=边沿下降触发    10x=边沿上升触发 11x=边沿触发	0000
Reserved	[19]	保留	0
EINT4[3:0]	[18:16]	设置 EINT4[3:0]的信令方法 000=低电平            001=高电平 01x=边沿下降触发    10x=边沿上升触发 11x=边沿触发	0000
Reserved	[15:8]	保留	0
Reserved	[7]	保留	0
EINT3[4]	[6:4]	设置 EINT3[4]的信令方法 000=低电平            001=高电平 01x=边沿下降触发    10x=边沿上升触发 11x=边沿触发	0000
Reserved	[3]	保留	0



EINT3[3:0]	[2:0]	设置 EINT3[3:0]的信令方法 000=低电平              001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
------------	-------	--	------

EINT56CON	位	描述	初始状态
Reserved	[31:27]	保留	0
EINT6[9:8]	[26:24]	设置 EINT6[9:8]的信令方法 000=低电平              001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[23]	保留	0
EINT6[7:4]	[22:20]	设置 EINT6[7:4]的信令方法 000=低电平              001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[19]	保留	0
EINT6[3:0]	[18:16]	设置 EINT6[3:0]的信令方法 000=低电平              001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[15:7]	保留	0
EINT5[6:4]	[6:4]	设置 EINT5[6:4]的信令方法 000=低电平              001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[3]	保留	0
EINT5[3:0]	[2:0]	设置 EINT5[3:0]的信令方法 000=低电平              001=高电平	0000

		01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	
--	--	--	--

EINT78CON	位	描述	初始状态
Reserved	[31]	保留	0
EINT8[14:12]	[30:28]	设置 EINT8[14:12]的信令方法  000=低电平      001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[27]	保留	0
EINT8[11:8]	[26:24]	设置 EINT8[11:8]的信令方法  000=低电平      001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[23]	保留	0
EINT8[7:4]	[22:20]	设置 EINT8[7:4]的信令方法  000=低电平      001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[19]	保留	0
EINT8[3:0]	[18:16]	设置 EINT8[3:0]的信令方法  000=低电平      001=高电平 01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	0000
Reserved	[15]	保留	0
EINT7[15:12]	[14:12]	设置 EINT7[15:12]的信令方法  000=低电平      001=高电平	0000

		01x=边沿下降触发      10x=边沿上升触发 11x=边沿触发	
Reserved	[11]	保留	0
EINT7[11:8]	[10:8]	设置 EINT7[11:8]的信令方法  000=低电平      001=高电平  01x=边沿下降触发      10x=边沿上升触发  11x=边沿触发	0000
Reserved	[7]	保留	0
EINT7[7:4]	[6:4]	设置 EINT7[7:4]的信令方法  000=低电平      001=高电平  01x=边沿下降触发      10x=边沿上升触发  11x=边沿触发	0000
Reserved	[3]	保留	0
EINT7[3:0]	[2:0]	设置 EINT7[3:0]的信令方法  000=低电平      001=高电平  01x=边沿下降触发      10x=边沿上升触发  11x=边沿触发	0000

EINT9CON	位	描述	初始状态
Reserved	[31:7]	保留	0
EINT9[8:4]	[6:4]	设置 EINT9[8:4]的信令方法  000=低电平      001=高电平  01x=边沿下降触发      10x=边沿上升触发  11x=边沿触发	0000
Reserved	[3]	保留	0
EINT9[3:0]	[2:0]	设置 EINT9[3:0]的信令方法  000=低电平      001=高电平  01x=边沿下降触发      10x=边沿上升触发  11x=边沿触发	0000

EINT12FLTCON	位	描述	初始状态
Reserved	[31:24]	保留	0x00
FLTEN2[7:0]	[23]	EINT2[7:0]滤波器使能 0=禁止            1=使能	0
EINT2[7:0]	[22:16]	EINT2[7:0]滤波宽度	000
FLTEN1[14:8]	[15]	EINT1[14:8]滤波器使能 0=禁止            1=使能	0
EINT1[14:8]	[14:8]	EINT1[14:8]滤波宽度	000
FLTEN1[7:0]	[7]	EINT1[7:0]滤波器使能 0=禁止            1=使能	0
EINT1[7:0]	[6:0]	EINT1[7:0]滤波宽度	000

EINT34FLTCON	位	描述	初始状态
FLTEN4[13:8]	[31]	EINT4[13:8]滤波器使能 0=禁止            1=使能	0
EINT4[13:8]	[30:24]	EINT4[13:8]滤波宽度	000
FLTEN4[7:0]	[23]	EINT4[7:0]滤波器使能 0=禁止            1=使能	0
EINT4[7:0]	[22:16]	EINT4[7:0]滤波宽度	000
Reserved	[15:8]	保留	0x00
FLTEN3[4:0]	[7]	EINT3[4:0]滤波器使能 0=禁止            1=使能	0
EINT3[4:0]	[6:0]	EINT3[4:0]滤波宽度	000

EINT56FLTCON	位	描述	初始状态
FLTEN6[9:8]	[31]	EINT6[9:8]滤波器使能	0

		0=禁止                  1=使能	
EINT6[9:8]	[30:24]	EINT6[9:8]滤波宽度	000
FLTEN6[7:0]	[23]	EINT6[7:0]滤波器使能 0=禁止                  1=使能	0
EINT6[7:0]	[22:16]	EINT6[7:0]滤波宽度	000
Reserved	[15:8]	保留	0x00
FLTEN5[6:0]	[7]	EINT5[6:0]滤波器使能 0=禁止                  1=使能	0
EINT5[6:0]	[6:0]	EINT5[6:0]滤波宽度	000

EINT78FLTCON	位	描述	初始状态
FLTEN8[15:8]	[31]	EINT8[15:8]滤波器使能 0=禁止                  1=使能	0
EINT8[15:8]	[30:24]	EINT8[9:8]滤波宽度	000
FLTEN8[7:0]	[23]	EINT8[7:0]滤波器使能 0=禁止                  1=使能	0
EINT8[7:0]	[22:16]	EINT8[7:0]滤波宽度	000
FLTEN7[15:8]	[15]	EINT7[15:8]滤波器使能 0=禁止                  1=使能	0
EINT7[15:8]	[14:8]	EINT7[9:8]滤波宽度	000
FLTEN7[7:0]	[7]	EINT7[7:0]滤波器使能 0=禁止                  1=使能	0
EINT7[7:0]	[7:0]	EINT7[7:0]滤波宽度	000

EINT9FLTCON	位	描述	初始状态
Reserved	[15:8]	保留	0x00
FLTEN9[8:0]	[7]	EINT9[8:0]滤波器使能 0=禁止                  1=使能	0

EINT9[8:0]	[6:0]	EINT9[8:0]滤波宽度	000
------------	-------	----------------	-----

EINT12MASK	位	描述	初始状态
Reserved	[31:24]	保留	0
EINT2[m]	[16+m] m=0~7	0=使中断                  1=屏蔽	1
Reserved	[15]	保留	0
EINT1[n]	[n] n=0~14	0=使中断                  1=屏蔽	1

EINT34MASK	位	描述	初始状态
Reserved	[31:30]	保留	0
EINT4[m]	[16+m] m=0~13	0=使中断                  1=屏蔽	1
Reserved	[15:5]	保留	0
EINT3[n]	[n] n=0~4	0=使中断                  1=屏蔽	1

EINT56MASK	位	描述	初始状态
Reserved	[31:26]	保留	0
EINT6[m]	[16+m] m=0~9	0=使中断                  1=屏蔽	1
Reserved	[15:7]	保留	0
EINT5[n]	[n] n=0~6	0=使中断                  1=屏蔽	1

EINT78MASK	位	描述	初始状态
EINT8[m]	[16+m]	0=使中断                  1=屏蔽	1

	m=0~14		
EINT7[n]	[n] n=0~15	0=使中断                  1=屏蔽	1

EINT9MASK	位	描述	初始状态
Reserved	[31:9]	保留	0
EINT9[n]	[n] n=0~8	0=使中断                  1=屏蔽	1

EINT12PEND	位	描述	初始状态
Reserved	[31:24]	保留	0
EINT2[m]	[16+m] m=0~7	0=不发生中断                  1=发生中断	0
Reserved	[15]	保留	0
EINT1[n]	[n] n=0~14	0=不发生中断                  1=发生中断	1

EINT34PEND	位	描述	初始状态
Reserved	[31:30]	保留	0
EINT4[m]	[16+m] m=0~13	0=不发生中断                  1=发生中断	1
Reserved	[15:5]	保留	0
EINT3[n]	[n] n=0~4	0=不发生中断                  1=发生中断	1

EINT56MASK	位	描述	初始状态
Reserved	[31:26]	保留	0
EINT6[m]	[16+m]	0=不发生中断                  1=发生中断	1

	m=0~9		
Reserved	[15:7]	保留	0
EINT5[n]	[n] n=0~6	0=不发生中断 1=发生中断	1

EINT78MASK	位	描述	初始状态
EINT8[m]	[16+m] m=0~14	0=不发生中断 1=发生中断	1
EINT7[n]	[n] n=0~15	0=不发生中断 1=发生中断	1

EINT9MASK	位	描述	初始状态
Reserved	[31:9]	保留	0
EINT9[n]	[n] n=0~8	0=不发生中断 1=发生中断	1

优先控制寄存器（PRIORITY）

寄存器	地址	读/写	描述	复位值
PRIORITY	0x7F008280	读/写	外部中断优先控制寄存器 0	0x000003FF

PRIORITY	位	描述	初始状态
ARB9	[9]	外部中断 9 组优先旋转使能 0=不优先旋转 0=优先旋转使能	1
ARB8	[8]	外部中断 8 组优先旋转使能 0=不优先旋转 0=优先旋转使能	1
ARB7	[7]	外部中观 7 组优先旋转使能 0=不优先旋转	1



		0=优先旋转使能	
ARB6	[6]	外部中观 6 组优先旋转使能 0=不优先旋转 0=优先旋转使能	1
ARB5	[5]	外部中观 5 组优先旋转使能 0=不优先旋转 0=优先旋转使能	1
ARB4	[4]	外部中观 4 组优先旋转使能 0=不优先旋转 0=优先旋转使能	1
ARB3	[3]	外部中观 3 组优先旋转使能 0=不优先旋转 0=优先旋转使能	1
ARB2	[2]	外部中观 2 组优先旋转使能 0=不优先旋转 0=优先旋转使能	1
ARB1	[1]	外部中观 1 组优先旋转使能 0=不优先旋转 0=优先旋转使能	1
ARB0	[0]	外部中观 0 组优先旋转使能 0=不优先旋转 0=优先旋转使能	1

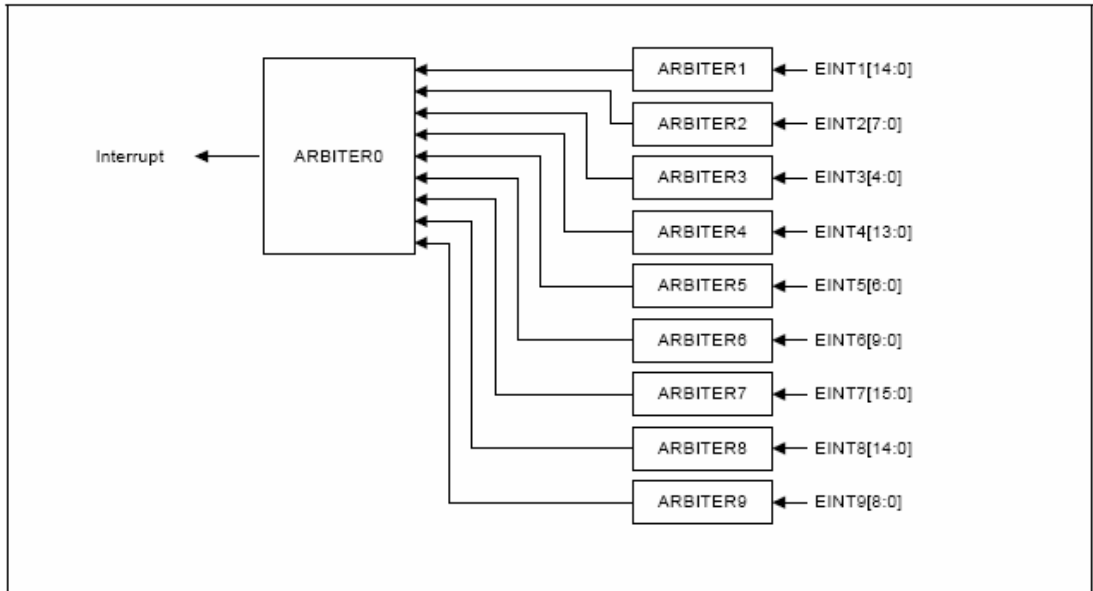


图 10-3 优先产生模块

### 10.3.当前服务寄存器（SERVICE）

当前服务寄存器将显示出服务于哪个中断。位值描述的是组的序号和终端的序号。通过 **PRIORITY** 寄存器决定位的值，当产生 nIRQ 时，位值无效。

当前服务寄存器将显示出需要清除哪个中断悬挂位。完成中断服务历程后，可以通过写入值清除中断悬挂寄存器内的中断悬挂位。如：如果当前服务寄存器的组序号是 4，可以通过向 **EINT34PEND** 内输入 **SERVICEPEND** 清除相应的中断悬挂位。

寄存器	地址	读/写	描述	复位值
SERVICE	0x7F008284	读	当前服务寄存器	0x00
SERVICEPEND	0x7F008288	读	当前服务悬挂寄存器	0x00

SERVICE	位	描述	初始状态
Group	[7:4]	组序号	0000
Interrupt No.	[3:0]	中断序号。当组位不为 00 时才有效	0000

睡眠模式下的外部管脚配置寄存器

此寄存器在睡眠模式下保持原来的值。

寄存器	地址	读/写	描述	复位值
SPCONSLP	0x7F008880	读/写	特殊端口睡眠模式配置寄存器	0x0000010
SLPEN	0x7F008930	读/写	睡眠模式 Pad 配置寄存器	0x00

SERVICE	位	描述	初始状态
Reserved	[31:15]	保留	0
TDOPULLDOWN	[14]	停止模式下的 XjTDO Pad 下拉控制 0=禁止 1=使下拉	0
RSTOUT	[13:12]	复位输出管脚（XnRSTOUT）配置 00=输出 0 01=输出 1 1x=输出禁止（hi-Z）	00
Reserved	[11: 6]	保留	00
CKE1_INIT	[5]	CKE 存储器端口 1 的初始值。只有当系统电源处于复位状态或者睡眠激活状态时有效。	0
Reserved	[4:2]	保留	00
KP_COL	[1:0]	键盘设定栏位 00=输出 0 01=输出 1 1x=输入	00

SLPEN	位	描述	初始状态
Reserved	[7:2]	保留	0
SLPEN_CFG	[1]	0: 通过睡眠模式自动进行 1: 通过 SLPEN 位	0
SLPEN	[0]	睡眠模式 Pad 状态使能寄存器 当此位设置为 1 时，外部管脚由睡眠模式控制，如	00

		<p>ACONSLP/MEM0COMSLP 等等。</p> <p>当系统进入睡眠模式，此位值自动设为 1，可以通过输入或冷启动进行清除。当苏醒时，此值仍保持 1 值。</p>	
--	--	---	--

# 11 DMA 控制器

这一节主要介绍用于 S3C6410 RSIC 微处理器的 DMA 控制器。S3C6410 包含四个的 DMA 控制器。每个 DMA 控制器是由八个传输的通道组成。DMA 控制器的每个通道能在 SPINE AXI 总线的设备和/或者 PERIPHERAL AXI 总线之间通过 AHB 到 AXI（没有其它限制）进行数据传输。换言之，每个通道可以处理以下四个案例，如：

- （1）源及目标在中心总线上；
- （2）当目标在外设总线上可用时，在中心总线上，源也可以用；
- （3）当目标在中心总线上可用时，在外设总线上，源也可以用；
- （4）源及目标可用在外设总线上。

ARM 的 PrimeCell DMA 控制器 PL080 用来作为 S3C6410 DMA 控制器。该 DMAC 是一个 AMBA AHB 模块，连接到先进，性能高的总线（AHB）。DMAC 是一个先进的微控制器总线体系（全名 Advanced Microcontroller Bus Architecture，即 AMBA），兼容的系统单晶片（全名 System-on-Chip，即 SoC），它的开发、测试，许可符合 ARM 的规范。

DMA 的主要优点是没有 CPU 的干预，同样可以传输数据。DMA 的操作可以通过 S / W 初始化，或通过内部外设做请求，或外部引脚做请求。

## 11.1 DMA 控制器的特性

### 1. DMA 控制器

DMA 控制器提供以下功能：

- （1）S3C6410 包含四个的 DMA 控制器。每个 DMA 控制器由八个传输的通道组成，每个通道支持单向传输。
- （2）每个 DMA 控制器提供了 16 个外设 DMA 请求。
- （3）每个外设连接到 DMAC，可以表明一个脉冲 DMA 请求或者一个单一的 DMA 请求。DMA 脉冲的大小通过执行 DMAC 来设置。
- （4）支持内存到内存，内存到外设，外设到内存以及外设向外设传输。

(5) 通过使用连接表，支持分散 DMA 或集合 DMA。

(6) 硬件 DMA 通道的优先权，每一个 DMA 通道有一个特定的硬件优先。DMA 通道 0 有最高优先级下降至通道 7，具有最低的优先级。如果请求在同一时间两个通道变为有效，则首先服务最高优先级。

(7) 该 DMAC 通过写入 DMA 控制寄存器来超过 AHB 接口。

(8) 两个 AXI 总线主要通过 AHB 和 AXI 桥传输数据，当 DMA 请求其作用时，这些接口将用于传输数据。

(9) 来源及目标的递增或非递增地址。

(10) 可编程的 DMA 的脉冲大小。DMA 的脉冲大小可以编程，以提高效率的传输数据。通常是脉冲大小，在外设，设置为 FIFO 的一半大小。

(11) 内部 4 字 FIFO 通道。

(12) 支持 8 位，16 位和 32 位宽度处理。

(13) 支持大端和小端模式。复位时，DMA 控制器默认的是小端模式。

(14) 单独的和组合的 DMA 错误和 DMA 计数中断请求。在一个 DMA 错误上或者当 DMA 计数读取 0（通常用于指示传输完成）时，处理器的中断产生。

三个中断请求信号是用来做到以下几点：

- 当传输已完成时，产生 DMACINTTC 信号。
- 当发生错误时，产生 DMACINERR 信号。
- DMACINTTC 和 DMACINERR 中断请求信号。DMACINTR 中断请求可以在系统中使用，其中有少数中断控制器的请求输入。

(15) 中断屏蔽。DMA 错误和 DMA 终端计数中断请求可能被屏蔽。

(16) 原始中断状态，DMA 错误和计数原始中断状态可以读取预屏蔽的信息。

## 2. DMA 控制器

DMA 控制器的结构框图，如图 11-1 所示。

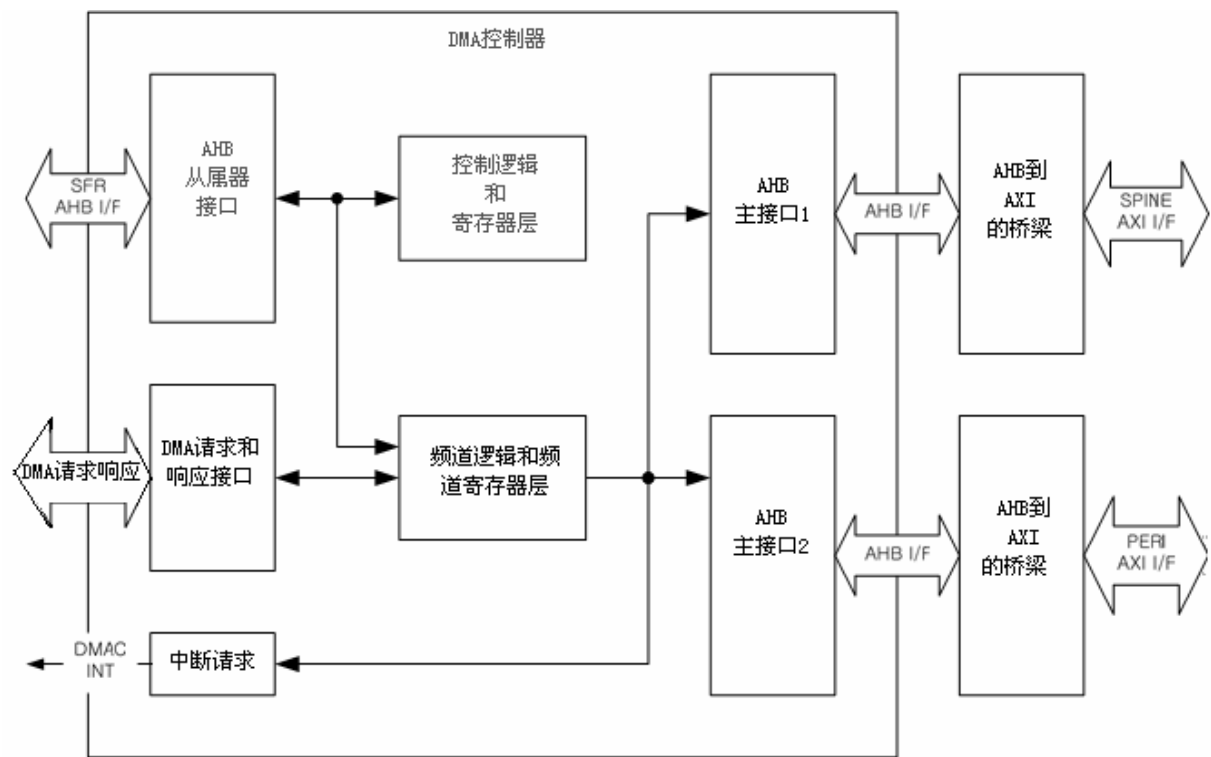


图 11-1 DMA 控制器的结构框图

## 11.2 DMA 源

该 S3C6410 支持 64 位 DMA 源，如表 11-1 所示。

表 11-1 DMA 源

组	DMA 编号	源	描述
DMA0, SDMA0	0	DMA_UART0[0]	UART0 DMA 源 0
DMA0, SDMA0	1	DMA_UART0[1]	UART0 DMA 源 1
DMA0, SDMA0	2	DMA_UART1[0]	UART1 DMA 源 0
DMA0, SDMA0	3	DMA_UART1[1]	UART1 DMA 源 1
DMA0, SDMA0	4	DMA_UART2[0]	UART2 DMA 源 0
DMA0, SDMA0	5	DMA_UART2[1]	UART2 DMA 源 1
DMA0, SDMA0	6	DMA_UART3[0]	UART3 DMA 源 0

DMA0, SDMA0	7	DMA_UART3[1]	UART3 DMA 源 1
DMA0, SDMA0	8	DMA_PCM0_TX	PCM0 DMA TX 源
DMA0, SDMA0	9	DMA_PCM0_RX	PCM0 DMA RX 源
DMA0, SDMA0	10	DMA_I2S0_TX	I2S0 TX DMA 源
DMA0, SDMA0	11	DMA_I2S0_RX	I2S0 RX DMA 源
DMA0, SDMA0	12	DMA_SPI0_TX	SPI0 TX DMA 源
DMA0, SDMA0	13	DMA_SPI0_RX	SPI0 RX DMA 源
DMA0, SDMA0	14	DMA_HSI_TX	MIPI HSI DMA TX 源
DMA0, SDMA0	15	DMA_HSI_RX	MIPI HSI DMA RX 源
DMA1, SDMA1	0	DMA_PCM1_TX	PCM1 DMA TX 源
DMA1, SDMA1	1	DMA_PCM1_RX	PCM1 DMA RX 源
DMA1, SDMA1	2	DMA_I2S1_TX	I2S1 TX DMA 源
DMA1, SDMA1	3	DMA_I2S1_RX	I2S1 RX DMA 源
DMA1, SDMA1	4	DMA_SPI1_TX	SPI1 TX DMA 源
DMA1, SDMA1	5	DMA_SPI1_RX	SPI1 RX DMA 源
DMA1, SDMA1	6	DMA_AC_PCMou	AC97 PCMout DMA 源
DMA1, SDMA1	7	DMA_AC_PCMin	AC97 PCMin DMA 源
DMA1, SDMA1	8	DMA_AC_MICin	AC97 MICin DMA 源
DMA1, SDMA1	9	DMA_PWM	PWM DMA 源
DMA1, SDMA1	10	DMA_IrDA	IrDA DMA 源
DMA1, SDMA1	11	Reserved	
DMA1, SDMA1	12	Reserved	
DMA1, SDMA1	13	Reserved	
DMA1, SDMA1	14	DMA_SECU_RX	安全 RX DMA 源
DMA1, SDMA1	15	DMA_SECU_TX	安全 TX DMA 源



## 11.3 DMA 接口

### 1. DMA 请求信号

DMA 请求信号是由外设要求的数据传输而被使用的。

该 DMA 请求信号表明：

- (1) 是否传输单个字或脉冲（多字）数据的传输需要。
- (2) 是否在数据包中，传输的是最后一次。

该 DMA 请求信号向 DMA 控制器为每个外设分列如下：

- (1) DMACxBREQ：脉冲请求信号。这个执行程序脉冲字的数目被转移。
- (2) DMACxSREQx：单传输请求信号。执行一个单个字被传输。该 DMA 控制器传输单个字到外设或来自外设。

注：如果外设只传输数据的脉冲，它不是强制地去连接单一传输请求信号。如果外设只传输单一的数据的字，则它不是强制地去连接脉冲请求信号。

### 2. DMA 的响应信号

该 DMA 响应信号表明是否传输由 DMA 请求信号完成。那个响应信号也可以被用来表明是否有一个完整的包已传输。

该 DMA 响应信号从 DMA 控制器为每个外设分列如下：

- (1) DMACxCLR<sub>x</sub>：DMA 的清除或确认信号。
- (2) DMACxTC：DMA 的终端计数信号。
- (3) DMA 使用 DMACxCLR<sub>x</sub> 信号来确认来自外设的 DMA 请求。
- (4) 该 DMACxTC 信号是以表明外设指出，DMA 传输完成所用的 DMA 控制器。

注：有些外设不需要连线的 DMA 终端计数信号。

### 3. 传输类型

该 DMA 控制器支持四种类型的传输：

- (1) 从内存到外设。
- (2) 从外设到内存。
- (3) 从内存到内存。

(4) 从外设到外设。

每一个传输类型可以转让任一外设，或 DMA 控制器作为流量控制器。因此，有四个可能的控制情况。

#### 4. 在 DMA 控制器的流控制下外设到内存的处理

对于不是脉冲大小倍数的处理，使用脉冲和单一请求信号，如图 11-2 所示。

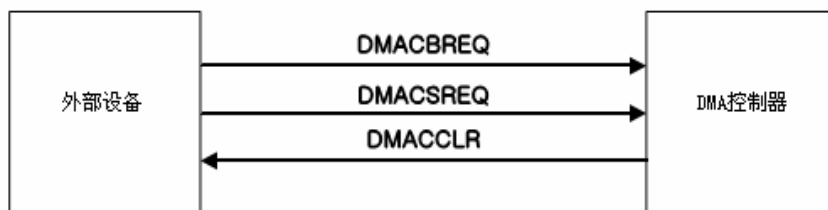


图 11-2 由脉冲和单一请求组成的外设到内存的处理

这两个请求信号并非互相排斥的，该 DMA 控制器监控器 DMACBREQ，数据的数据左传输大于脉冲大小时，并当发生请求时，开始一个脉冲传输（来自外设）。当数据的数据左传输小于脉冲大小时，DMA 控制器监控 DMACBREQ，并当发生请求时，开始单一传输。

#### 5. 在 DMA 控制器的流控制下内存到外设的处理

处理多种模块大小，只用于脉冲模块请求信号，如图 11-3 所示，由脉冲组成的内存到外设的处理。

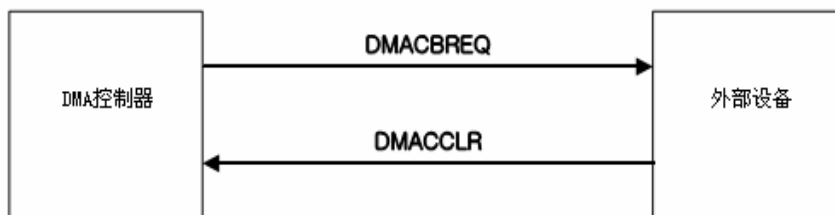


图 11-3 由脉冲组成的内存到外设的处理

只请求 DMACBREQ。当剩余的数据数量大于脉冲的大小，DMA 控制器发送数据的全脉冲。当剩余的数据数量小于脉冲的大小，DMAC 再次监控 DMACBREQ，并且传输当请求的时候，传输剩余的数据。

#### 6. 在 DMA 控制器的流控制下内存到内存的处理

软件程序从内存到内存传输的 DMA 通道。当它启用，DMA 通道没有 DMA 请求开始传输。它继续，直到发生下列情况中的一种：

- (1) 所有数据转移。
- (2) 通过软件禁止该通道。

注：必须执行内存到内存的传输与低通道优先，否则，其它 DMA 通道不能进入总线，直到内存到内存的传输已经完成，或其他 AHB 的控制无法执行任何处理。

内存到内存的处理，如图 11-4 所示。

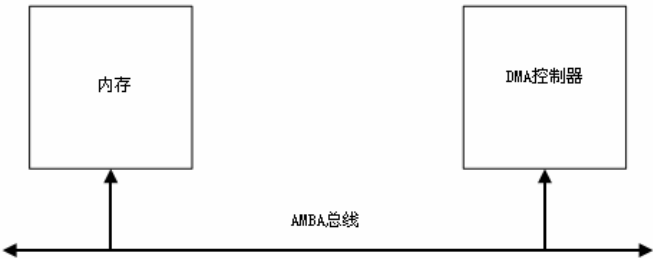


图 11-4 在 DMA 控制器的流控制下内存到内存的处理

7. 在 DMA 控制器的流控制下外设到外设的处理

当处理的不是脉冲大小的倍数时，用下面的信号：

- (1) 单一和脉冲请求（DMACBREQ and DMACSREQ）来源是外设信号。
- (2) 脉冲请求（DMACBREQ）目标是外设信号。

外设到外设的处理，如图 11-5 所示。

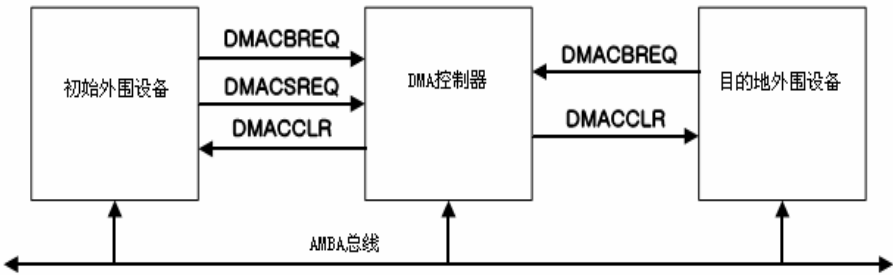


图 11-5 由脉冲和单一请求组成的外设到外设的处理

源外设遵循同样程序，当作描述外设到内存的 DMA 控制器的流量控制。目标外设遵循同样的程序，当作描述内存到外设的处理下外设的流量控制。下一个 LLI 装载时，所有读与写的传输是完整的。可以使用 DMACTC 的信号表明，过去的的数据已传输到外设上。如表 11-2 所示。

表 11-2 DMA 控制器的流量控制

传输方向	请求发生器	请求信号使用
外设到内存	外设	DMACBREQ
内存到外设	外设	DMACBREQ
内存到内存	DMA 控制器	None
外设到外设	外设	Src: DMACBREQ

## 11.4 信号时序

DMA 信号的时序行为描述如下：

(1) DMA 请求信号 DMAC{L}(B/S)REQ<sub>x</sub>

通知 DMA 控制器，该外设准备按指定的大小进行 DMA 传输。

高有效位。由 DMA 控制器取样，关于 HCLK 的实际优势。DMA 请求信号用于连接 DMACCLR 信号来实现握手。

(2) DMA 的承认或明确的 DMACCLR<sub>x</sub>

简单说明一个 DMA 传输的完成。高有效位。

(3) DMA 的终端计数的 DMACTC<sub>x</sub>

简单说明数据包的最后已经准备。高有效位。

注：如果 DMA 请求来源不使用相同的时钟作为 DMA 控制器，则在 DMACSync 寄存器中，必须通过设置相关的位请求同步。

## 11.5 功能时序图

外设表明，一个 DMA 请求保持有效状态。该 DMACCLR 信号声称，当结束数据项目已被传输时，DMA 控制器表明 DMACCLK 信号，如图 11-6 所示。

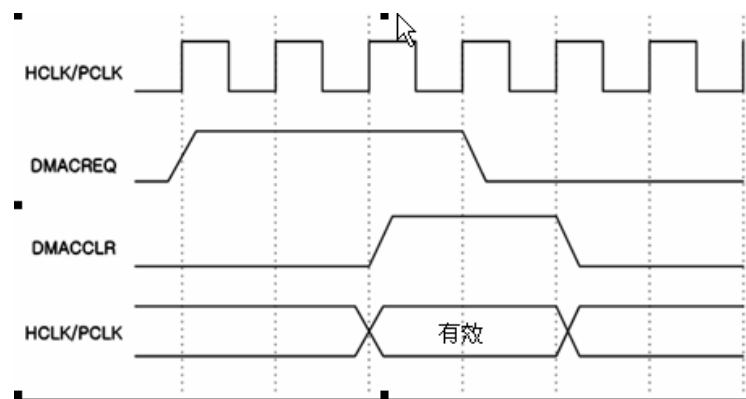


图 11-6 DMA 接口时序

## 11.6 程序员的模式

### 1. 设计 DMA 控制器

所有 AHB 从执行总线的处理必须是 32 位带宽。当编程 DMA 控制器时，消除尾端的问题。

### 2. 启动 DMA 控制器

在 DMA 配置寄存器，启动 DMA 控制器设置 DMA 的使能位。

### 3. 禁用 DMA 控制器

禁用 DMA 控制器采取下列步骤：

(1) 读取 DMACEnbldChns 寄存器，并确保所有 DMA 通道已被禁用。如果任何通道是有效的，提交禁用 DMA 通道。

(2) 通过在 DMA 配置寄存器写入 DMA 有效位，停用 DMA 控制器。

### 4. 启动 DMA 通道

在启动 DMA 通道设置，控制有关 DMA 通道配置寄存器。

注：通道必须在初始化之前，它是启用的。此外，DMA 控制器必须设置之前，启用通道。

### 5. 禁用 DMA 通道

DMA 通道可以被禁用是在以下三个方面：

- (1) 直接写入通道使能位。如果使用这个方法任何在 FIFO 中的突出的数据将丢失。
- (2) 使用有效和停止位连接通道使能位。
- (3) 直到传输完成。该通道是自动禁止。

禁用 DMA 通道，并丢失 FIFO 中数据：

在有关通道配置寄存器中，清楚有关通道使能位。当前的 AHB 传输完成和通道被禁用。在 FIFO 中，任何数据失去。

禁止 DMA 通道，并不失去的数据。

禁止 DMA 通道并不会丢失 FIFO 中数据的步骤如下：

(1) 在有关的通道配置寄存器中设置停止位。忽略进一步的 DMA 请求。

(2) 在有关的通道配置寄存器轮询有效位，直到它满足条件。该位指示是否通道中还有没传输的数据。

(3) 在有关通道配置寄存器中，清除有关通道使能位。

## 6. 建立一个新的 DMA 传输

建立一个新的 DMA 传输的步骤：

(1) 如果通道没有设置用于 DMA 处理的边：

- a. 读取 DMACEnbldChns 控制器寄存器，并找出哪些通道是无效的。
- b. 选择一个无效的通道，它有请求优先级。

(2) 执行 DMA 控制器。

## 7. 停止 DMA 通道

在有关的 DMA 通道配置寄存器中，设置停止位。用于当前源请求，直到停止位被清除，任何另外的来源的 DMA 请求被忽略。

## 8. 编程 DMA 通道

编程 DMA 通道的步骤：

(1) 决定是否使用安全 DMAC (SDMAC) 或一般 DMAC (DMAC)。在正常状态下使用一般 DMAC，禁用系统控制器的安全 DMA 控制寄存器 (sdma\_sel)。(重置 SDMAC)

(2) 根据优先需要，选择自由 DMA 通道。DMA 通道 0 有最高优先权，DMA 通道 7 优先权最低。

(3) 通过写 DMACIntTCClr 和 DMACIntErrClr 寄存器，清除通道中要用到的未处理的中断。先前的通道操作可能使剩余的中断有效。

(4) 写源地址到 DMACCxSrcAddr 寄存器中。

(5) 写目标地址到 DMACCxSrcAddr 寄存器中。

(6) 将下一个 LLI 的地址写入 DMACCxLLI 寄存器。如果是传输单一的包，那么必须写入

该寄存器。如表 11-3 所示。

表 11-3 写下一个 LLI 地址到 DMACCxLLI 寄存器

偏移	内容
下一个 LLI 地址	用于下一个传输的源地址
下一个 LLI 地址 0x04	用于下一个传输的目标址
下一个 LLI 地址 0x08	用于下一个传输的下一个 LLI 地址
下一个 LLI 地址 0x0C	用于下一个传输的 DMACCxControl0 数据
下一个 LLI 地址 0x10	下一个传输的 DMACCxControl1 数据

(7) 写控制信息在 DMACCxControl 寄存器中。

(8) 写通道配置信息到 DMACCxConfiguration 寄存器中。如果使能位被设置，那么该 DMA 通道自动启用。

## 11.7 寄存器描述

有四个 DMA 控制器名称，其为 DMAC0，DMAC1，SDMAC0 和 SDMAC1。

DMAC0，DMAC1，SDMAC0 和 SDMAC1 寄存器基地址分别是 0x7500\_0000，0x7510\_0000，0x7DB0\_0000 和 0x7DC0\_0000 。

用于 OneNAND 控制器的页面访问功能是通过 DMAC0 和 SDAMC0 的通道 3 添加的。

DMA 寄存器的位置。如表 11-4 所示，对 DMA 寄存器进行概括的描述。

表 11-4 DMA 寄存器概要

名称	类型	宽度	描述	偏移量	复位值
DMACIntStatus	读	8	该寄存器提供 DMA 控制器的中断状态。 高位指示一个特殊的 DMA 通道中断有效。	0x000	0x00
DMACIntTCStatus	读	8	该寄存器是用来判断处理过程中是否产生中断，高位指出传输被完成。	0x004	0x00
DMACIntTCClear	写	8	当写入该寄存器，每个数据位都是高位 将 使 DMACIntTCStatus 和 DMACRawIntTCStatus 寄存器清除。数	0x008	–

			据位是低位则对相应的寄存器没有影响。		
DMACIntErrorStatus	读	8	这寄存器是用来判断由于一个错误是否有中断产生。	0x00C	0x00
DMACIntErrClr	写	8	当写这个寄存器时，每个数据位都是高位将使 DMACIntErrorStatus 和 DMACRawIntErrorStatus 寄存器清除，数据位是低位则对相应的寄存器没有影响。	0x010	–
DMACRawIntTCStatus	读	8	屏蔽前，该寄存器提供 DMA 终端计数中断的原始状态。高位指出中断请求优先屏蔽，被激活。	0x014	–
DMACRawIntErrorStatus	读	8	屏蔽前，该寄存器提供 DMA 错误的原始状态。高位指出中断请求优先屏蔽，被激活。	0x018	–
DMACEnbldChns	读	8	寄存器显示 DMA 通道被激活，高位指出中断请求优先屏蔽，被激活。	0x01C	0x00
DMACSoftBReq	读/写	16	该寄存器通过软件允许 DMA 的脉冲模块产生。	0x020	0x0000
DMACSoftSReq	读/写	16	这寄存器通过软件允许 DMA 的单一请求产生。	0x024	0x0000
DMACSoftLBReq	读/写	16	该寄存器通过软件允许 DMA 的最后脉冲模块产生。	0x028	0x0000
DMACSoftLSReq	读/写	16	该寄存器通过软件允许 DMA 的最后单一请求产生。	0x02C	0x0000
DMACConfiguration	读/写	3	该寄存器是用来配置 DMA 控制器。	0x030	0b000
DMACSync	读/写	16	该寄存器启用或禁止用于 DMA 请求信号的同步逻辑。	0x034	0x0000
DMAC0SrcAddr	读/写	32	DMA 通道 0 的初始化地址。	0x100	0x00000000



DMACC0DestAddr	读/写	32	DMA 通道 0 的目标地址。	0x104	0x00000000
DMACC0LLI	读/写	32	DMA 通道 0 的链表地址。	0x108	0x00000000
DMACC0Control0	读/写	32	DMA 通道 0 控制器 0。	0x10C	0x00000000
DMACC0Control1	读/写	32	DMA 通道 0 控制器 1。	0x110	0x00000000
DMACC0Configuration	读/写	19	DMA 通道 0 配置寄存器。	0x114	0x00000
DMACC1SrcAddr	读/写	32	DMA 通道 1 的初始化地址。	0x120	0x00000000
DMACC1DestAddr	读/写	32	DMA 通道 1 的目标地址。	0x124	0x00000000
DMACC1LLI	读/写	32	DMA 通道 1 的链表地址。	0x128	0x00000000
DMACC1Control0	读/写	32	DMA 通道 1 控制器 0。	0x12C	0x00000000
DMACC1Control1	读/写	32	DMA 通道 1 控制器 1。	0x130	0x00000000
DMACC1Configuration	读/写	19	DMA 通道 1 配置寄存器。	0x134	0x00000
DMACC2SrcAddr	读/写	32	DMA 通道 2 的初始化地址。	0x140	0x00000000
DMACC2DestAddr	读/写	32	DMA 通道 2 的目标地址。	0x144	0x00000000
DMACC2LLI	读/写	32	DMA 通道 2 的链表地址。	0x148	0x00000000
DMACC2Control0	读/写	32	DMA 通道 2 控制器 0。	0x14C	0x00000000
DMACC2Control1	读/写	32	DMA 通道 2 控制器 1。	0x150	0x00000000
DMACC2Configuration	读/写	19	DMA 通道 2 配置寄存器。	0x154	0x00000
DMACC3SrcAddr	读/写	32	DMA 通道 3 的初始化地址。	0x160	0x00000000
DMACC3DestAddr	读/写	32	DMA 通道 3 的目标地址。	0x164	0x00000000
DMACC3LLI	读/写	32	DMA 通道 3 的链表地址。	0x168	0x00000000
DMACC3Control0	读/写	32	DMA 通道 3 控制器 0。	0x16C	0x00000000
DMACC3Control1	读/写	32	DMA 通道 3 控制器 1。	0x170	0x00000000
DMACC3Configuration	读/写	19	DMA 通道 3 配置寄存器。	0x174	0x00000
DMACC4SrcAddr	读/写	32	DMA 通道 4 的初始化地址。	0x180	0x00000000
DMACC4DestAddr	读/写	32	DMA 通道 4 的目标地址。	0x184	0x00000000
DMACC4LLI	读/写	32	DMA 通道 4 的链表地址。	0x188	0x00000000
DMACC4Control0	读/写	32	DMA 通道 4 控制器 0。	0x18C	0x00000000
DMACC4Control1	读/写	32	DMA 通道 4 控制器 1。	0x190	0x00000000

DMACC4Configuration	读/写	19	DMA 通道 4 配置寄存器。	0x194	0x00000
DMACC5SrcAddr	读/写	32	DMA 通道 5 的初始化地址。	0x1A0	0x00000000
DMACC5DestAddr	读/写	32	DMA 通道 5 的目标地址。	0x1A4	0x00000000
DMACC5LLI	读/写	32	DMA 通道 5 的链表地址。	0x1A8	0x00000000
DMACC5Control0	读/写	32	DMA 通道 5 控制器 0。	0x1AC	0x00000000
DMACC5Control1	读/写	32	DMA 通道 5 控制器 1。	0x1B0	0x00000000
DMACC5Configuration	读/写	19	DMA 通道 5 配置寄存器。	0x1B4	0x00000
DMACC6SrcAddr	读/写	32	DMA 通道 6 的初始化地址。	0x1C0	0x00000000
DMACC6DestAddr	读/写	32	DMA 通道 6 的目标地址。	0x1C4	0x00000000
DMACC6LLI	读/写	32	DMA 通道 6 的链表地址。	0x1C8	0x00000000
DMACC6Control0	读/写	32	DMA 通道 6 控制器 0。	0x1CC	0x00000000
D MACC6Control1	读/写	32	DMA 通道 6 控制器 1。	0x1D0	0x00000000
D MACC6Configuration	读/写	19	DMA 通道 6 配置寄存器。	0x1D4	0x00000
DMACC7SrcAddr	读/写	32	DMA 通道 7 的初始化地址。	0x1E0	0x00000000
DMACC7DestAddr	读/写	32	DMA 通道 7 的目标地址。	0x1E4	0x00000000
DMACC7LLI	读/写	32	DMA 通道 7 的链表地址。	0x1E8	0x00000000
DMACC7Control	读/写	32	DMA 通道 7 控制器 0。	0x1EC	0x00000000
DMACC7Control1	读/写	32	DMA 通道 7 控制器 1。	0x1F0	0x00000000
DMACC7Configuration	读/写	19	DMA 通道 7 配置寄存器。	0x1F4	0x00000

11.7.1. 中断状态寄存器 DMACIntStatus

该 DMACIntStatus 寄存器是只读类型，并且在屏蔽后指示中断状态。高位指示一个特殊的 DMA 通道中断请求有效。由于错误或者终端计数中断请求，该请求产生。

显示 DMACIntStatus 寄存器的位分配，如表 11-5 所示。

表 11-5 DMACIntStatus 寄存器的位分配

DMACIntStatus	位	类型	功能
---------------	---	----	----

IntStatus	[7:0]	R	DMA 的地位屏蔽后中断
-----------	-------	---	--------------

### 11.7.2. 中断终端计数状态寄存器，DMACIntTCStatus

DMACIntTCStatus 寄存器是只读类型，并且屏蔽后显示终端计数状态。如果结合中断请求，这个寄存器必须用于关联 DMACIntStatus 寄存器。DMACINTCOMBINE，用于中断请求。

如果使用 DMACINTTC 的中断请求，那么就得读 DMACIntTCStatus 寄存器来确定中断请求的来源。如表 11-6 所示。

表 11-6 显示 DMACIntTCStatus 寄存器的位分配

DMACIntTCStatus	位	类型	功能
IntTCStatus	[7:0]	读	中断终端计数状态

### 11.7.3. 中断终端计数清除寄存器，DMACIntTCClear

DMACIntTCClear 寄存器是只写类型，用于清除中断计数中断请求。

当写入这个寄存器，每个数据位设置为高位，原因是在状态寄存器中的相应位被清除。数据位为低位，不影响该寄存器中的相应位。如表 11-7 所示，显示了 DMACIntTCClear 寄存器的位分配。

表 11-7 DMACIntTCClear 寄存器的位分配

DMACIntTCClear	位	类型	功能
IntTCClear	[7:0]	写	终端计数请求清除

### 11.7.4. 中断错误状态寄存器，DMACIntErrorStatus

DMACIntErrorStatus 寄存器是只读类型，屏蔽后显示错误请求的状态。如果结合中断请求，该寄存器必须被用于关联 DMACIntStatus 寄存器，DMACINTCOMBINE 用于请求中断。

如果 DMACINTERROR 中断请求只用于 DMACIntErrorStatus 寄存器，需要读取。表 11-8 所示，显示 DMACIntErrorStatus 寄存器的位分配。

表 11-8 DMACIntErrorStatus 寄存器的位分配

DMACIntErrorStatus	位	类型	功能
IntErrorStatus	[7:0]	读	中断错误状态

### 11.7.5. 中断错误清除寄存器，DMACIntErrClr

DMACIntErrClr 寄存器是只写类型，它用于清除错误中断请求。当读这个寄存器时，每个数据为是高位，原因是在状态寄存器中的相应位被清除。当数据位是低位，不影响该寄存器中的相应位。如表 11-9 所示，显示 DMACIntErrClr 寄存器的位分配。

表 11-9 DMACIntErrClr 寄存器的位分配

DMACIntErrClr	位	类型	功能
IntErrClr	[7:0]	写	中断错误清除

### 11.7.6. 原始中断终端计数状态寄存器，DMACRawIntTCStatus

DMACRawIntTCStatus 寄存器是只读类型。它指示其中的 DMA 通道请求一个传输完成（终端计数中断）。高位指示，终端计数中断请求是有效的，优先于屏蔽。如表 11-10 所示，显示 DMACRawIntTCStatus 寄存器的位分配。

表 11-10 DMACRawIntTCStatus 寄存器的位分配

DMACRawIntTCStatus	位	类型	功能
RawIntTCStatus	[7:0]	读	终端计数中断状态优先于屏蔽

### 11.7.7. 原始错误中断状态寄存器，DMACRawIntErrorStatus

DMACRawIntErrorStatus 寄存器是只读类型，它指示其中 DMA 通道屏蔽前，请求传输完成。高位指示终端的计数中断请求优先于屏蔽被激活。如表 11-11 所示，显示 DMACRawIntErrorStatus 寄存器的位分配。

表 11-11 DMACRawIntErrorStatus 寄存器的位分配

DMACRawIntErrorStatus	位	类型	功能
-----------------------	---	----	----

RawIntErrorStatus	[7:0]	读	终端计数中断状态优先于屏蔽
-------------------	-------	---	---------------

### 11.7.8. 通道启动状态寄存器 DMACEnbldChns

DMACEnbldChns 寄存器是只读类型，它指示其中 DMA 通道由 DMACCxConfiguration 寄存器中的启动位启动。高位指示，DMA 通道已启动。该位在 DMA 传输的完成时被清除。如表 11-12 所示，显示 DMACRawIntErrorStatus 寄存器的位分配。

表 11-12 DMACEnbldChns 寄存器的位分配

DMACEnbldChns	位	类型	功能
EnabledChannels	[7:0]	读	通道启动状态

### 11.7.9. 软件脉冲请求寄存器，DMACSoftBReq

DMACSoftBReq 寄存器是读/写类型，它通过软件允许 DMA 脉冲请求发生。DMA 请求可以发生用于每个来源，写入 1 到相应的寄存器位。当完成传输时，寄存器位被清除。写入 0 到这个寄存器没有影响。

读取该寄存器指示其中源请求单一 DMA 传输。从外设或者软件请求寄存器产生一个请求。如表 11-13 所示，显示 DMACSoftBReq 寄存器的位分配。

表 11-13 DMACSoftBReq 寄存器的位分配

DMACSoftBReq	位	类型	功能
SoftBReq	[15:0]	读/写	软件脉冲请求

注：它被推荐，软件和硬件外设请求不能使用在同一时间。

### 11.7.10. 软件单一请求寄存器，DMACSoftSReq

DMACSoftSReq 寄存器是读/写类型，它允许 DMA 信号请求通过软件而发生。DMA 请求可以发生用于每个来源，写入 1 到相应的寄存器位。当传输完成时，寄存器位被清除。写入 0 到这个寄存器没有影响。

读这个寄存器指示，来源是请求 DMA 信号传输。一个请求可以从外设或软件请求寄存器那发生。如表 11-14 所示，显示 DMACSoftSReq 寄存器的位分配。

表 11-14 DMACSoftSReq 寄存器的位分配

DMACSoftSReq	位	类型	功能
SoftSReq	[15:0]	读/写	软件单一请求

注：它被推荐，软件和硬件外设请求不可以用于同一时间。

11.7.11. 软件最后脉冲请求寄存器，DMACSoftLBReq

DMACSoftLBReq 寄存器是读/写类型，它允许 DMA 最后脉冲请求通过软件发生。DMA 请求可以用于每个来源，写入 1 到相应的寄存器位。当传输完成时，寄存器位被清除。写入 0 到这个寄存器没有影响。

读这个寄存器指出，来源是请求最后脉冲 DMA 传输。一个请求可以从外设或软件请求寄存器那发生。如表 11-15 所示，显示 DMACSoftLBReq 寄存器的位分配。

表 11-15 DMACSoftLBReq 寄存器的位分配

DMACSoftLBReq	位	类型	功能
SoftLBReq	[15:0]	读/写	软件最后脉冲请求

11.7.12. 软件最后单一请求寄存器，DMACSoftLSReq

DMACSoftLSReq 读/写寄存器，允许 DMA 最后信号请求通过软件发生。DMA 请求可以用于每个初始化，写入 1 到相应的寄存器位。当传输完成时，寄存器位被清除。写入 0 到这个寄存器没有影响。

读这个寄存器指出，来源是请求最后信号 DMA 传输。一个请求可以从外设或软件请求寄存器那发生。如表 11-16 所示，显示 DMACSoftLBReq 寄存器的位分配。

表 11-16 DMACSoftLSReq 寄存器的位分配

DMACSoftLSReq	位	类型	功能
SoftLSReq	[15:0]	读/写	软件最后单一请求

### 11.7.13. 配置寄存器，DMACConfiguration

DMACConfiguration 读/写寄存器，用于配置 DMA 控制器的操作。个别 AHB 主接口的字节序，可以通过写入这个寄存器的 M1 和 M2 被改变。M1 允许 AHB 主机口 1 的字节序被改变。M2 允许 AHB 主机口 2 的字节序被改变。AHB 主机口在复位上被设置为小端模式。如表 11-17 所示，显示 DMACConfiguration 寄存器的位分配。

表 11-17 DMACConfiguration 寄存器的位分配

DMACConfiguration	位	类型	功能
M2	[2]	读/写	AHB 主接口 2 字节序配置：  0 =小端模式  1 =大端模式  这个位复位为 0
M1	[1]	读/写	AHB 主接口 1 字节序配置：  0 =小端模式  1 =大端模式  这个位复位为 0
E	[0]	读/写	DMA 控制器启动：  0 =禁止  1 =启动  这个位复位为 0。禁止 DMA 控制器还原电力消耗。

注：它不是强制性的为 AHB 主机接口有同样的字节序。

### 11.7.14. 同步寄存器，DMACSync

DMACSync 读/写寄存器，用于为 DMA 请求信号启动/禁止同步逻辑。DMA 请求信号由 DMACBREQ[15:0]，DMACSREQ[15:0]，DMACLBREQ[15:0]和 DMACLSREQ[15:0]信号组成。位设置为

0，用于 DMA 请求的特殊组启动同步逻辑。位设置为 1，用于 DMA 请求的特殊组禁止同步逻辑。这个寄存器复位为 0，同步逻辑启动。如表 11-18 所示，显示 DMACSync 寄存器的位分配。

表 11-18 DMACSync 寄存器的位分配

DMACSync	位	类型	功能
DMACSync	[15:0]	读/写	用于 DMA 请求信号启动或禁止的 DMA 同步逻辑。低位指示，同步逻辑用于 DMACBREQ[15:0] ， DMACSREQ[15:0] ， DMACLBREQ[15:0]，和 DMACLSREQ[15:0] 请求信号的启动。高位指示，同步逻辑禁止。

注：当外设产生 DMA 请求运行 DMA 控制器的不同时钟时，必须使用同步逻辑。外设以和 DMA 控制器禁止同步逻辑相同的时钟运行，来改进 DMA 请求应答时间。如果有必要，DMA 响应信号，DMACCLK 和 DMACTC 在外设上必须是同步的。

11.7.15. 通道源地址寄存器，DMACCxSrcAddr

8 位读/写 DMACCxSrcAddr 寄存器，它包含当前数据的源地址（字节对齐）被传输。通道启动前，每个寄存器都直接通过软件编程。

当 DMA 通道启动这个寄存器：

- 作为源地址是增量。
- 当完成一个数据包的传输时，根据链接列表。

当通道有效，但不提供有用的信息时，读这个寄存器。这是因为通过时间，软件处理读取值，通道可能有进步。当通道停止时，只能故意被读取，在这样情况下，它显示最后信息源地址被读取。如表 11-19 所示，显示 DMACCxSrcAddr 寄存器的位分配。

注：源地址和目标地址必须对准源和目标的宽度。

表 11-19 DMACCxSrcAddr 寄存器的位分配

DMACCxSrcAddr	位	类型	功能
SrcAddr	[31:0]	读/写	DMA 源地址



### 11.7.16. 通道目标地址寄存器，DMACCxDestAddr

8 位读/写 DMACCxDestAddr 寄存器，它包含当前数据的源地址（字节定位）被传输。通道启动前，每个寄存器都直接通过软件编程。当 DMA 通道启动时，随着目标地址的增加，该寄存器不断进行更新。

当通道有效，但不提供有用的信息时，读这个寄存器。这是因为通过时间，软件处理读取值，通道可能有进步，当通道停止时，只能故意被读取，在这样情况下，它显示最后信息源地址被读取。如表 11-20 所示，显示 DMACCxDestAddr 寄存器的位分配。

注：源地址和目标地址必须对准源和目标的宽度。

表 11-20 DMACCxDestAddr 寄存器的位分配

DMACCxDestAddr	位	类型	功能
DestAddr	[31:0]	读/写	DMA 目标地址

### 11.7.17. 通道链表列表项目寄存器，DMACCxLLI

8 位读/写 DMACCxLLI 寄存器，它包含下一个链表列表项目（LLI）的字对齐地址。如果还有 LLI，当前 LLI 的最后一个时，并一旦所有的 DMA 发送完成，DMA 通道被禁止。如表 11-21 所示，显示 DMACCxLLI 寄存器的位分配。

注：当 DMA 通道启动不可预知的副作用时，设计该寄存器。用于一些系统加载更有效的 LLI，LLI 数据构造由四个字对齐组成。

表 11-21 DMACCxLLI 寄存器的位分配

DMACCxLLI	位	类型	功能
LLI	[31:2]	读/写	链表列表项目。 用于下一个 LLI，地址的位 [31:2] 。 地址的位 [1:0] 。
R	[1]	读/写	保留，必须被写入为 0，屏蔽读 。
LM	[0]	读/写	用于下载下一个 LLI 的 AHB 主选择：

			LM = 0 = AHB 主机 1
			LM = 1 = AHB 主机 2

### 11.7.18. 通道控制寄存器，DMACCxControl0

8 位读/写 DMACCxControl0 寄存器，它包含 DMA 通道控制信息，如脉冲大小和传输宽度。通道启动前，每个寄存器都直接通过软件编程。当 DMA 通道启动时，随着目标地址的增加，该寄存器不断进行更新。并当完成一个数据包的传输时，读这个寄存器。同时通道有效，但不发送有用的信息。这是因为通过时间，软件处理读取值，通道可能有进步。当通道停止时，只能故意被读取。如表 11-22 所示，显示 DMACCxControl0 寄存器的位分配。

表 11-22 DMACCxControl0 寄存器的位分配

DMACCxControl	位	类型	功能
I	[31]	读/写	终端计数中断启动位。控制是否当前的 LLI 期望引发终端计数中断。
Prot	[30:28]	读/写	保护。
DI	[27]	读/写	目标增量。每个传输后，随设置的目标地址递增。
SI	[26]	读/写	源增量。每个传输后，随设置的源地址递增。
D	[25]	读/写	目标 AHB 主机选择： 0 = AHB 主机 1 (AXI_SPINE)，用于目标传输选择。 1 = AHB 主机 2 (AXI_PERI)，用于目标传输选择
S	[24]	读/写	源 AHB 主选择： 0 = AHB 主机 1 (AXI_SPINE)，用于源传输选择。 1 = AHB 主机 2 (AXI_PERI)，用于源传输选择
Dwidth	[23:21]	读/写	目标传输宽度。 传输宽度比 AHB 主总线宽度宽是非法的。源与目标宽度和其它的宽度是不同的。硬件自动压缩和解压数据包作为请求。
Swidth	[20:18]	读/写	源传输宽度。 传输宽度比 AHB 主总线宽度宽是非法的。源与目标宽度和其它的宽度是不同的。硬件自动压缩和解压数据包作为请求。

DBSize	[17:15]	读/写	目标脉冲大小。  显示传输的数目，提出一个目标脉冲传输请求。这个值必须设置为目标外设的脉冲大小，或如果目标是内存，就是内存边界大小。脉冲大小是数据量，当 DMACxBREQ 信号在目标外设中有效时，脉冲大小被传输。脉冲大小与 AHB HBURST 没有关系。
SBSize	[14:12]	读/写	源脉冲大小。  显示传输数目，提出一个源脉冲。这个值必须设置为源外设的脉冲大小，或如果源是内存，就是内存边界大小。脉冲大小是数据量，当 DMACxBREQ 信号在源外设中有效时，脉冲大小被传输。脉冲大小与 AHB HBURST 没有关系。
Reserved	[11:0]	读	保留。

如表 11-23 所示，显示了源或目标脉冲大小。

表 11-23 源或目标脉冲大小

DBSize 或 SBSize 的位值	源或目标脉冲传输请求大小
0b000	1
0b001	4
0b010	8
0b011	16
0b100	32
0b101	64
0b110	128
0b111	256

如表 11-24 所示，显示了源或目标传输宽度。

表 11-24 源或目标传输宽度

SWidth 或 DWidth 位值	源或目标宽度
0b000	字节（9 位）
0b001	半字节（16 位）

0b010	字（32 位）
0b011	保留
0b100	保留
0b101	保留
0b110	保留
0b111	保留

当传输发生时，AHB 存取信息提供给源和目标外设。编程 DMA 通道提供传输信息（DMACCxControl 寄存器的 Prot 位和 DMACCxConfiguration 寄存器的 Lock 位）。这些位通过软件编程和外设使用，来判断这个信息是否有必要。提供信息的三个位，如表 11-25 所示，显示这三个保护位的目标。

表 11-25 保护位

位	描述	目标
0	特权或用户	表明，该访问是在用户或特权模式： 0 = 用户模式 1 = 特权模式 该位控制 AHB HPROT[1]信号。
1	Bufferable 或 not bufferable	表明，该访问是 bufferable 或 not bufferable : 0 = not bufferable 1 = bufferable 这个位控制 AHB HPROT[2]信号
2	Cacheable 或 not cacheable	表明，该访问是 cacheable 或 not cacheable : 0 =not cacheable 1 = cacheable 这个位控制 AHB HPROT[3]信号。

### 11.7.19. 通道控制寄存器，DMACCxControl1

8 位读/写 DMACCxControl1 寄存器，它包含 DMA 通道控制信息，如，传输大小。在 DMA 通道启动之前，每个寄存器通过软件直接编程。当 DMA 通道启动时，随着目标地址的增加，该寄存器不断进行更新。当通道有效，但不提供有用的信息时，读这个寄存器。这是因为通过时间，软件处理读取值，通道可

能有进步，当通道停止时，只能故意被读取，在这样情况下，它显示最后信息源地址被读取。如表 11-26 所示，显示 DMACCxControl1 寄存器的位分配。

表 11-26 DMACCxControl1 寄存器的位分配

DMACCxControl	位	类型	功能
TransferSize	[24:0]	读/写	传输大小。用于写入，当 DMA 控制器是流量控制器时，此栏显示传输（源的宽度）的数目来执行。用于读取，传输大小在目标总线上显示传输完成的数目。当通道有效但没有给予有意义的值时，软件已经处理该读取的值，通道已经进行，这时读该寄存器。  当通道启动和禁止时，特意被使用。如果 DMAC 控制器不是流量控制器，传输值的大小不可以被使用。

11.7.20. 通道配置寄存器，DMACCxConfiguration

8位读/写DMACCxConfiguration寄存器，用于配置DMA通道。当请求一个新的LLI时，该寄存器不更新。如表11-27所示，介绍了DMACCxConfiguration寄存器的位的分配情况。

表11-27 DMACCxConfiguration寄存器的位分配

DMACCxConfiguration	位	类型	功能
H	[18]	读/写	停止：  0 = 允许 DMA 请求  1 = 屏蔽进一步的源 DMA 请求  FIFO通道的内容被传输完成。该值有效，并通道启动位完全禁止一个DMA通道。
A	[17]	读	有效：  0 = 通道的FIFO中没有数据  1 = 通道的FIFO中有数据

			该值有效，并通道启动位完全禁止DMA通道。
L	[16]	读/写	锁  设置该位来使锁定的传输启动。
ITC	[15]	读/写	终端计数中断屏蔽。当清除该位，屏蔽相关通道的终端计数错误中断。
IE	[14]	读/写	中断错误屏蔽。当清除该位，屏蔽相关通道的错误中断。
FlowCntrl	[13:11]	读/写	流控制和传输类型。该值用于指示流控制器和传输类型。支持的流控制器只有DMA控制器。传输类型有存储器到存储器，存储器到外设，外设到存储器，或者外设到外设。
Reserved (OneNandModeDst)	[10]	读/写	保留，必须写入0，并且屏蔽读取操作。  用于DMAC0和SDMAC0的通道3，该位用于支持页写入类型。如果该位设置为1，并且目标地址指向OneNAND控制器的地址域，则目标地址增量设置支持OneNAND控制器的 01 指令。当该位设置为1，D应当是“AHB主控制器1”，D1应当是“增量”，DWidth应当为“字”，DBSize应当是4的倍数。
DestPeripheral	[9:6]	读/写	目标外设。该值选择DMA目标请求外设。如果是传输到存储器，则该区域屏蔽。
Reserved (OneNandModeSrc)	[5]	读/写	保留。必须写0，并且屏蔽读取。  对于通道3，该位用于支持页写入类型。如果该位设置为1，并且目标地址指向OneNAND控制器的地址域，则目标地址增量设置支持OneNAND控制器的 01指令。当该位设置为1，D应当是“AHB主控制器1”，D1应当是“增量”，DWidth应当为“字”，DBSize应当是4的倍数。
SrcPeripheral	[4:1]	读/写	源外设。该值选择DMA源请求外设。如果传输源是存储器，则屏蔽该区域。
E	[0]	读/写	通道有效。读取该位来指示当前通道是有效还是无效。  0 = 通道无效  1 = 通道有效  通过读取DMACEnbldChns寄存器也能知道通道有效位的状态。  通过设置该位来使通道无效。当前AHB传输完成并且通道无效时，通道FIFO的数据将丢失。通过设置通道有效位，来重启通道有不可预知的结果，并且通道必须充分初始化。

			<p>当最后的LLI被读取，或者有通道错误，通道有效位被清除，通道也无效。</p> <p>如果通道不得不停止，并且不丢失通道FIFO的数据，则必须设置停止位以屏蔽进一步的DMA请求。必须轮询激活位直到它为0，也就是FIFO中没有剩余数据了。最后可以清除通道有效位了。</p>
--	--	--	---

## 11.8 DMA 控制器应用举例

本节主要针对 DMA 控制器流控制下的内存到内存的处理，做举例说明。通过具体代码的分析，对 DMA 控制器功能的理解有所帮助。

下面是 ARM11 处理器中，内存到内存传输处理的具体代码实现：

DMAT\_MemtoMem 功能函数

输入：None

输出：None

```
static void DMAT_MemtoMem(void)
{
    u32 i, csel, usel;
    u32 uTsize, uBurst, uCh;
    // DMA 控制器的四种方式选择
    Disp("\nSelect DMA Controller : 0:DMAC0, 1:DMAC1, 2:SDMAC0, 3:SDMAC1: ");
    csel=GetIntNum();
    Disp("\n");
    switch(csel)
    {
        case 0: //DMAC0 选择
            Disp("Selected DMAC 0 ..... \n");
            DMAC_InitCh(DMA0, DMA_ALL, &oDmac0);
```

```

        INTC_SetVectAddr(NUM_DMA0, Dma0Done);
        INTC_Enable(NUM_DMA0);
        break;
    case 1:                                     //DMAC1 选择
        Disp("Selected DMAC 1 ..... \n");
        DMAC_InitCh(DMA1, DMA_ALL, &oDmac1);
        INTC_SetVectAddr(NUM_DMA1, Dma1Done);
        INTC_Enable(NUM_DMA1);
        break;
    case 2:                                     //SDMAC0 选择
        Disp("Selected SDMAC 0 ..... \n");
        DMAC_InitCh(SDMA0, DMA_ALL, &oDmac2);
        INTC_SetVectAddr(NUM_SDMA0, Sdma0Done );
        INTC_Enable(NUM_SDMA0);
        break;
    case 3:                                     //SDMAC1 选择
        Disp("Selected SDMAC 1 ..... \n");
        DMAC_InitCh(SDMA1, DMA_ALL, &oDmac3);
        INTC_SetVectAddr(NUM_SDMA1, Sdma1Done);
        INTC_Enable(NUM_SDMA1);
        break;
    default : Assert(0);
}
//通道选择
Disp("\nSelect Channel : 0:CH0, 1:CH1, 2:CH2, 3:CH3, 4:CH4, 5:CH5, 6:CH6, 7:CH7      :
");
usel=GetIntNum();
Disp("\n");
switch(usel)

```



```

{
    case 0:
        uCh= DMA_A;
        break;
    case 1:
        uCh= DMA_B;
        break;
    case 2:
        uCh= DMA_C;
        break;
    case 3:
        uCh = DMA_D;
        break;
    case 4:
        uCh = DMA_E;
        break;
    case 5:
        uCh = DMA_F;
        break;
    case 6:
        uCh = DMA_G;
        break;
    case 7:
        uCh = DMA_H;
        break;
    default : Assert(0);
}

```

//传输宽度选择

```
Disp("\nSelect Transfer Width : 0:BYTE, 1:WORD, 2:WORD: ");
```

```

usel=GetIntNum();
Disp("\n");
switch(usel)
{
    case 0:                                     //字节选择
        uTsize = BYTE;
        break;

    case 1:                                     //半字选择
        uTsize = HWORD;
        break;

    case 2:                                     //字选择
        uTsize = WORD;
        break;

    default : Assert(0);
}
//脉冲大小选择
Disp("\nSelect Burst Size : 0:SINGLE, 1:BURST4, 2:BURST8, 3:BURST16, 4:BURST32, 5:BURST64,
        6:BURST128, 7:BURST256: ");
usel=GetIntNum();
Disp("\n");
switch(usel)
{
    case 0:
        uBurst = SINGLE;
        break;

    case 1:
        uBurst = BURST4;
        break;

```

```

case 2:
    uBurst = BURST8;
    break;
case 3:
    uBurst = BURST16;
    break;
case 4:
    uBurst = BURST32;
    break;
case 5:
    uBurst = BURST64;
    break;
case 6:
    uBurst = BURST128;
    break;
case 7:
    uBurst = BURST256;
    break;
default : Assert(0);
}

```

// 传输大小选择

```
Disp("\nSelect Transfer Size [1~0x200_0000] : ");
```

```
uDataCnts=GetIntNum();
```

```
Disp("\n");
```

// 0.清除 rx/tx 缓冲器

```
for (i = 0; i<(uDataCnts*uTsize+16); i++)
```

```
{
```

```
    *(u8 *) (uRxBuffAddr+i) = 0;
```

```

        *(u8 *) (uTxBuffAddr+i) = 0;
    }

// 1. 建立 tx 缓冲器
for (i = 0; i<uDataCnts*uTsize; i++)
    *(u8 *) (uTxBuffAddr+i) = (u8) (i+2)%0xff;
    switch(csel)
    {
        case 0 :
// Channel, LLI_Address, SrcAddr, Src Type, DstAddr, Dst Type, Transfer Width, Transfer Size,
// OpMode(DEMAND), Src Req, Dst Req, Burst
        DMACH_Setup((DMA_CH)uCh, 0x0, uTxBuffAddr, 0, uRxBuffAddr, 0, (DATA_SIZE)uTsize,
                    uDataCnts, DEMAND, MEM, MEM, (BURST_MODE)uBurst, &oDmac0);
        DMACH_Start(&oDmac0);
        break;

        case 1 :
// Channel, LLI_Address, SrcAddr, Src Type, DstAddr, Dst Type, Transfer Width, Transfer Size,
// OpMode(DEMAND), Src Req, Dst Req, Burst
        DMACH_Setup((DMA_CH)uCh, 0x0, uTxBuffAddr, 0, uRxBuffAddr, 0, (DATA_SIZE)uTsize,
                    uDataCnts, DEMAND, MEM, MEM, (BURST_MODE)uBurst, &oDmac1);
        DMACH_Start(&oDmac1);
        break;

        case 2 :
// Channel, LLI_Address, SrcAddr, Src Type, DstAddr, Dst Type, Transfer Width, Transfer Size,
// OpMode(DEMAND), Src Req, Dst Req, Burst
        DMACH_Setup((DMA_CH)uCh, 0x0, uTxBuffAddr, 0, uRxBuffAddr, 0, (DATA_SIZE)uTsize,
                    uDataCnts, DEMAND, MEM, MEM, (BURST_MODE)uBurst, &oDmac2);
        DMACH_Start(&oDmac2);
        break;
    }

```

```

        case 3 :
// Channel, LLI_Address, SrcAddr, Src Type, DstAddr, Dst Type, Transfer Width, Transfer Size,
OpMode(DEMAND), Src Req, Dst Req, Burst
        DMACH_Setup((DMA_CH)uCh, 0x0, uTxBuffAddr, 0, uRxBuffAddr, 0, (DATA_SIZE)uTsize,
                    uDataCnts, DEMAND, MEM, MEM, (BURST_MODE)uBurst, &oDmac3);
        DMACH_Start(&oDmac3);
        break;
    default :
        Assert(0);
}

while(g_DmaDone==0); // Int.
//while (!DMAC_IsTransferDone(&oDmac0)); // Polling
if (CompareDMA(uTxBuffAddr, uRxBuffAddr, (DATA_SIZE)uTsize, uDataCnts))
    Disp(" >> Test Tx&Rx -> Ok << \n");
else
{
    Disp(" >>*** Tx-data & Rx-data mismatch ***<< \n");
}

switch(csel)
{
    case 0:
        INTC_Disable(NUM_DMA0);
        DMAC_Close(DMA0, DMA_ALL, &oDmac0);
        break;

    case 1:
        INTC_Disable(NUM_DMA1);

```

```
        DMAC_Close(DMA1, DMA_ALL, &oDmac1);
        break;
case 2:
    INTC_Disable(NUM_SDMA0);
    DMAC_Close(SDMA0, DMA_ALL, &oDmac2);
    break;
case 3:
    INTC_Disable(NUM_SDMA1);
    DMAC_Close(SDMA1, DMA_ALL, &oDmac3);
    break;
default : Assert(0);
}
}
```

## 12 矢量中断控制器

本节主要描述 S3C6410X RISC 微处理器内的矢量中断控制器的功能及用途。

### 12.1 概述

S3C6410X 内的中断控制器由 2 个 VIC(矢量中断控制器, ARM PrimeCell PL192)和 2 个 TZIC(TrustZone 中断控制器, sp890)组成。两个矢量中断控制器和两个 TrustZone 中断控制器链接在一起支持 64 位中断源。

S3C6410X 内的矢量中断控制器的性能如下:

- (1) 每个 VIC 支持 32 位的矢量 IRP 中断
- (2) 支持固定硬件中断优先级和可编程中断优先级
- (3) 支持硬件中断优先级屏蔽和可编程中断优先级屏蔽
- (4) 产生 IRQ 和 FIQ 中断
- (5) 产生软件中断
- (6) raw 中断状态
- (7) 中断请求状态
- (8) 支持限制访问的特权模式

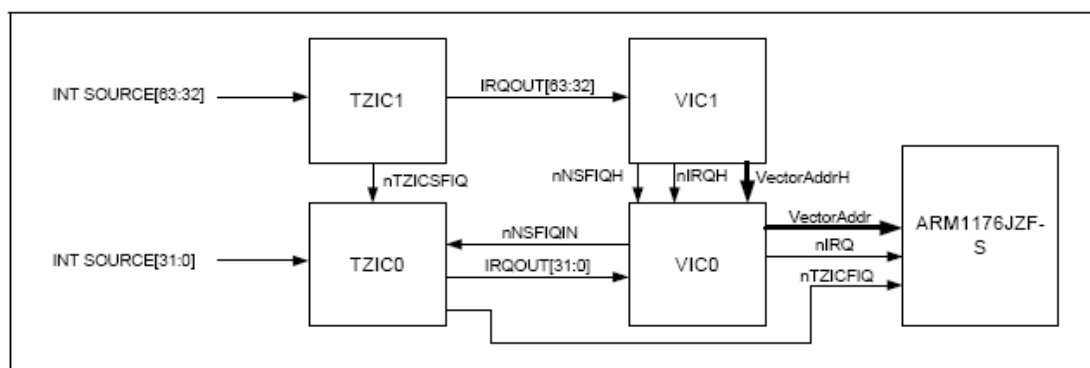


图 12-1 S3C6410X 的中断控制器

# 12.2 中断源

S3C6410X 支持 64 位中断源，不支持 ARM1176HZF-S 镜像中断运行.

中断号	中断源	描述	组
63	INT_ADC	ADC EOC 中断	VIC1
62	INT_PENDNUP	ADC 笔向下/向上中断 中断	VIC1
61	INT_SEC	安全中断	VIC1
60	INT_RTC_ALARM	RTC 警告中断	VIC1
59	INT_IrDA	IrDA 中断	VIC1
58	INT_OTG	USB OTG 中断	VIC1
57	INT_HSMMC1	HSMMC1 中断	VIC1
56	INT_HSMMC0	HSMMC0 中断	VIC1
55	INT_HOSTIF	主机接口中断	VIC1
54	INT_MSM	MSM 调制解调器 I/F 中断	VIC1
53	INT_EINT4	外部中断组 1~组 9	VIC1
52	INT_HSIrx	HS Rx 中断	VIC1
51	INT_HSItx	HS Tx 中断	VIC1
50	INT_I2C0	I2C 0 中断	VIC1
49	INT_SPI/INT_HSMMC2	SPI 中断或 HSMMC2 中断	VIC1
48	INT_SPI0	SPI0 中断	VIC1
47	INT_UHOST	USB 主机中断	VIC1
46	INT_CFC	CFCON 中断	VIC1
45	INT_NFC	NFCON 中断	VIC1
44	INT_ONENAND1	板块 1 的 ONENANE 中断	VIC1
43	INT_ONENAND0	板块 0 的 ONENAND 中断	VIC1
42	INT_DMA1	DMA1 中断	VIC1
41	INT_DMA0	DMA0 中断	VIC1
40	INT_UART3	UART3 中断	VIC1



39	INT_UART2	UART2 中断	VIC1
38	INT_UART1	UART1 中断	VIC1
37	INT_UART0	UART0 中断	VIC1
36	INT_AC97	AC 中断	VIC1
35	INT_PCM1	PCM1 中断	VIC1
34	INT_PCM0	PCM0 中断	VIC1
33	INT_EINT3	外部中断 20~27	VIC1
32	INT_EINT2	外部中断 12~19	VIC1
31	INT_LCD[2]	LCD 中断. 系统 I/F 完成	VIC0
30	INT_LCD[1]	LCD 中断. VSYNC 中断	VIC0
29	INT_LCD[0]	LCD 中断. FIFO 不足	VIC0
28	INT_TIMER4	定时器 4 中断.	VIC0
27	INT_TIMER3	定时器 3 中断.	VIC0
26	INT_WDT	看门狗定时器中断.	VIC0
25	INT_TIMER2	定时器 2 中断.	VIC0
24	INT_TIMER1	定时器 1 中断.	VIC0
23	INT_TIMER0	定时器 0 中断.	VIC0
22	INT_KEYPAD	键盘中断.	VIC0
21	INT_ARM_DMAS	ARM DMAS 中断.	VIC0
20	INT_ARM_DMA	ARM DMA 中断.	VIC0
19	INT_ARM_DMAERR	ARM DMA 错误中断.	VIC0
18	INT_SDMA1	安全 DMA1 中断.	VIC0
17	INT_SDMA0	安全 DMA0 中断.	VIC0
16	INT_MFC	MFC 中断.	VIC0
15	INT_JPEG	JPEG 中断.	VIC0
14	INT_BATF	电池故障中断.	VIC0
13	INT_SCALER	TV 转换器中断.	VIC0
12	INT_TVENC	TV 编码器中断.	VIC0

11	INT_ 2D	2D 中断.	VIC0
10	INT_ ROTATOR	旋转器中断.	VIC0
9	INT_POST0	后处理器中断.	VIC0
8	INT_ 3D	3D 图像控制器中断.	VIC0
7	Reserved	保留	VIC0
6	INT_ I2S0 \INT_I2S1 \INT_I2SV40	I2S0 中断或 I2S1 中断或 I2SV40 中断	VIC0
5	INT_ I2C1	I2C1 中断	VIC0
4	INT_ CAMIF_P	照相机接口中断	VIC0
3	INT_ CAMIF_C	照相机接口中断	VIC0
2	INT_ RTC_TIC	RTC TIC 中断	VIC0
1	INT_ EINT1	外部中断 4~11	VIC0
0	INT_ EINT0	外部中断 0~3	VIC0

## 12.3 矢量中断控制器功能模块图

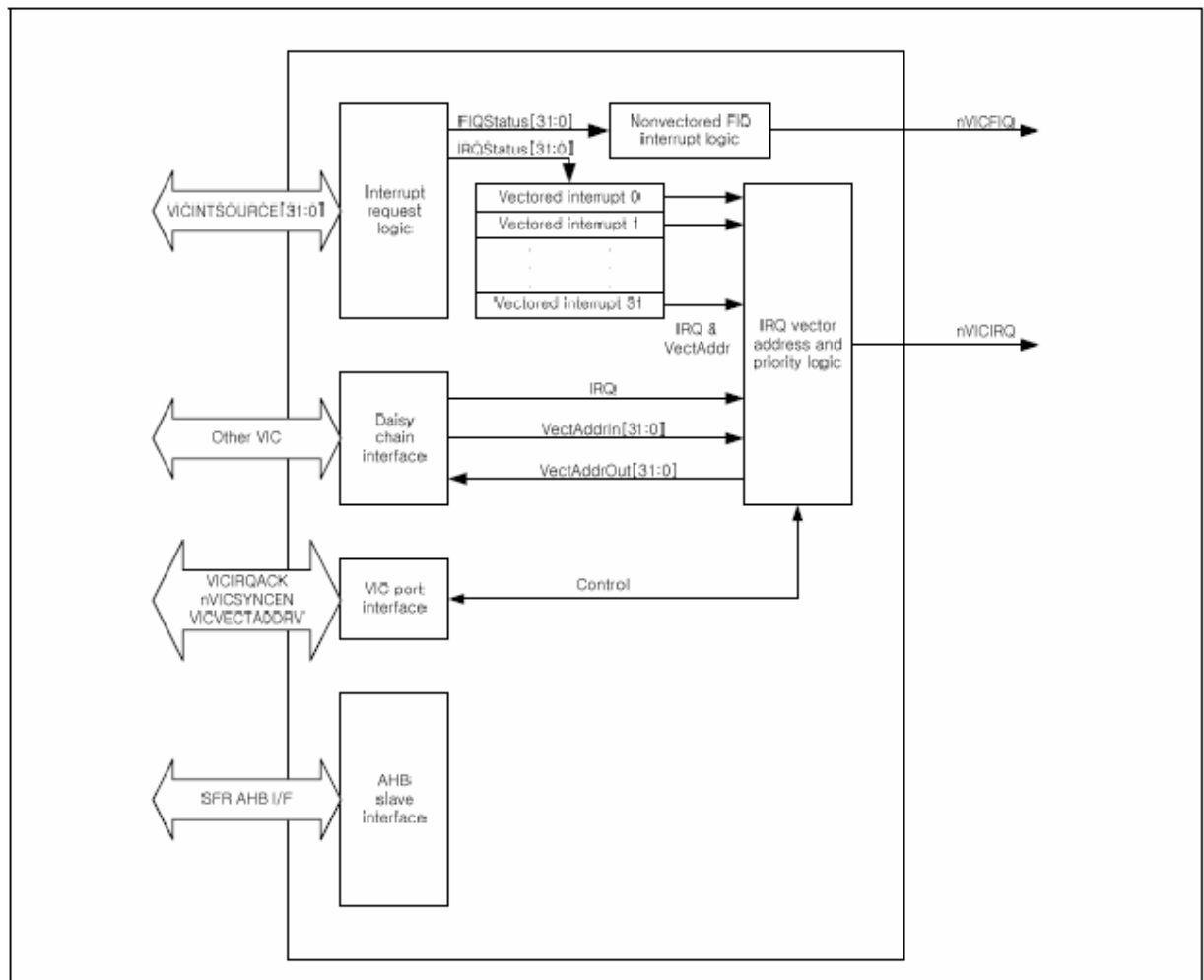


图 12-2 矢量中断控制器模块图

## 12.4 矢量中断控制器摘要

VIC0 的基础地址是 0x7120\_0000

VIC1 的基础地址是 0x7130\_0000

控制寄存器地址=基础地址+补偿区

寄存器	补偿区	类型	描述	复位值
VICxIRQSTATUS	0x000	读	IRQ 状态寄存器	0x00000000
VICxFIQSTATUS	0x004	读	FIQ 状态寄存器	0x00000000
VICxIRAWINTR	0x008	读	原始中断状态寄存器	0x00000000
VICxINTSELECT	0x00C	读写	中断选择寄存器	0x00000000
VICxINTENABLE	0x010	读写	中断使能寄存器	0x00000000
VICxINTENCLEAR	0x014	写	中断使能清除寄存器	-
VICxSOFTINT	0x018	读写	软件中断寄存器	0x00000000
VICxISOFTINTCLEAR	0x01C	写	软件中断清除寄存器	-
VICxPROTECTION	0x020	读写	保护使能寄存器	0x0
VICxSWPRIORITYMASK	0x024	读写	软件优先屏蔽寄存器	0x0FFFF
VICxPRIORITYDAISY	0x028	读写	菊花链的矢量优先寄存器	0xF
VICxVECTADDR0	0x100	读写	矢量地址 0 寄存器	0x00000000
VICxVECTADDR1	0x104	读写	矢量地址 1 寄存器	0x00000000
VICxVECTADDR2	0x108	读写	矢量地址 2 寄存器	0x00000000
VICxVECTADDR3	0x10C	读写	矢量地址 3 寄存器	0x00000000
VICxVECTADDR4	0x110	读写	矢量地址 4 寄存器	0x00000000
VICxVECTADDR5	0x114	读写	矢量地址 5 寄存器	0x00000000
VICxVECTADDR6	0x118	读写	矢量地址 6 寄存器	0x00000000
VICxVECTADDR7	0x11C	读写	矢量地址 7 寄存器	0x00000000
VICxVECTADDR8	0x120	读写	矢量地址 8 寄存器	0x00000000
VICxVECTADDR9	0x124	读写	矢量地址 9 寄存器	0x00000000
VICxVECTADDR10	0x128	读写	矢量地址 10 寄存器	0x00000000
VICxVECTADDR11	0x12C	读写	矢量地址 11 寄存器	0x00000000
VICxVECTADDR12	0x130	读写	矢量地址 12 寄存器	0x00000000
VICxVECTADDR13	0x134	读写	矢量地址 13 寄存器	0x00000000
VICxVECTADDR14	0x138	读写	矢量地址 14 寄存器	0x00000000
VICxVECTADDR15	0x13C	读写	矢量地址 15 寄存器	0x00000000

VICxVECTADDR16	0x140	读写	矢量地址 16 寄存器	0x00000000
VICxVECTADDR17	0x144	读写	矢量地址 17 寄存器	0x00000000
VICxVECTADDR18	0x1408	读写	矢量地址 18 寄存器	0x00000000
VICxVECTADDR19	0x14C	读写	矢量地址 19 寄存器	0x00000000
VICxVECTADDR20	0x150	读写	矢量地址 20 寄存器	0x00000000
VICxVECTADDR21	0x154	读写	矢量地址 21 寄存器	0x00000000
VICxVECTADDR22	0x158	读写	矢量地址 22 寄存器	0x00000000
VICxVECTADDR23	0x15C	读写	矢量地址 23 寄存器	0x00000000
VICxVECTADDR24	0x160	读写	矢量地址 24 寄存器	0x00000000
VICxVECTADDR25	0x164	读写	矢量地址 25 寄存器	0x00000000
VICxVECTADDR26	0x168	读写	矢量地址 26 寄存器	0x00000000
VICxVECTADDR27	0x16C	读写	矢量地址 27 寄存器	0x00000000
VICxVECTADDR28	0x170	读写	矢量地址 28 寄存器	0x00000000
VICxVECTADDR29	0x174	读写	矢量地址 29 寄存器	0x00000000
VICxVECTADDR30	0x178	读写	矢量地址 30 寄存器	0x00000000
VICxVECTADDR31	0x17C	读写	矢量地址 31 寄存器	0x00000000
VICxVECPRIORITY0	0x200	读写	矢量优先 0 寄存器	0xF
VICxVECPRIORITY1	0x204	读写	矢量优先 1 寄存器	0xF
VICxVECPRIORITY2	0x208	读写	矢量优先 2 寄存器	0xF
VICxVECPRIORITY3	0x20C	读写	矢量优先 3 寄存器	0xF
VICxVECPRIORITY4	0x210	读写	矢量优先 4 寄存器	0xF
VICxVECPRIORITY5	0x214	读写	矢量优先 5 寄存器	0xF
VICxVECPRIORITY6	0x218	读写	矢量优先 6 寄存器	0xF
VICxVECPRIORITY7	0x21C	读写	矢量优先 7 寄存器	0xF
VICxVECPRIORITY8	0x220	读写	矢量优先 8 寄存器	0xF
VICxVECPRIORITY9	0x224	读写	矢量优先 9 寄存器	0xF
VICxVECPRIORITY10	0x228	读写	矢量优先 10 寄存器	0xF
VICxVECPRIORITY11	0x22C	读写	矢量优先 11 寄存器	0xF

VICxVECTPRIORITY12	0x230	读写	矢量优先 12 寄存器	0xF
VICxVECTPRIORITY13	0x234	读写	矢量优先 13 寄存器	0xF
VICxVECTPRIORITY14	0x238	读写	矢量优先 14 寄存器	0xF
VICxVECTPRIORITY15	0x23C	读写	矢量优先 15 寄存器	0xF
VICxVECTPRIORITY16	0x240	读写	矢量优先 16 寄存器	0xF
VICxVECTPRIORITY17	0x244	读写	矢量优先 17 寄存器	0xF
VICxVECTPRIORITY18	0x2408	读写	矢量优先 18 寄存器	0xF
VICxVECTPRIORITY19	0x24C	读写	矢量优先 19 寄存器	0xF
VICxVECTPRIORITY20	0x250	读写	矢量优先 20 寄存器	0xF
VICxVECTPRIORITY21	0x254	读写	矢量优先 21 寄存器	0xF
VICxVECTPRIORITY22	0x258	读写	矢量优先 22 寄存器	0xF
VICxVECTPRIORITY23	0x25C	读写	矢量优先 23 寄存器	0xF
VICxVECTPRIORITY24	0x260	读写	矢量优先 24 寄存器	0xF
VICxVECTPRIORITY25	0x264	读写	矢量优先 25 寄存器	0xF
VICxVECTPRIORITY26	0x268	读写	矢量优先 26 寄存器	0xF
VICxVECTPRIORITY27	0x26C	读写	矢量优先 27 寄存器	0xF
VICxVECTPRIORITY28	0x270	读写	矢量优先 28 寄存器	0xF
VICxVECTPRIORITY29	0x274	读写	矢量优先 29 寄存器	0xF
VICxVECTPRIORITY30	0x278	读写	矢量优先 30 寄存器	0xF
VICxVECTPRIORITY31	0x27C	读写	矢量优先 31 寄存器	0xF
VICxADDRESS	0xf00	读写	矢量地址寄存器	0x00000000

## 12.5 寄存器描述

### 12.5.1. IRQ 状态寄存器，VICIRQSTATUS

寄存器	地址	读/写	描述	复位值
-----	----	-----	----	-----

VIC0IRQSTATUS	0x7120_0000	读	IRQ 状态寄存器(VIC0)	0x0000_0000
VIC1IRQSTATUS	0x7130_0000	读	IRQ 状态寄存器(VIC1)	0x0000_0000

名称	位	描述	复位值
IRQStatus	[31:0]	在屏蔽之后,通过 VICxINTENABLE 和 VICxINTSELECT 寄存器显示中断的状态  0=中断不被激活(复位)  1=中断被激活  每个中断源都有一个寄存器位	0x0

### 12.5.2.FIQ 状态寄存器, VICFIQSTATUS

寄存器	地址	读/写	描述	复位值
VIC0FIQSTATUS	0x7120_0004	读	FIQ 状态寄存器(VIC0)	0x0000_0000
VIC1FIQSTATUS	0x7130_0004	读	FIQ 状态寄存器(VIC1)	0x0000_0000

名称	位	描述	复位值
FIQStatus	[31:0]	在屏蔽之后,通过 VICINTENABLE 和 VICINTSELECT 寄存器显示 FIQ 中断的状态  0=中断不被激活(复位)  1=中断被激活  每个中断源都有一个寄存器位	0x0

### 12.5.3.原始中断状态寄存器, VICRAWINTR

寄存器	地址	读/写	描述	复位值
VIC0RAWINTR	0x7120_0008	读	原始中断状态寄存器(VIC0)	0x0000_0000

VIC1RAWINTR	0x7130_0008	读	原始中断状态寄存器(VIC1)	0x0000_0000
-------------	-------------	---	-----------------	-------------

名称	位	描述	复位值
RawInterrupt	[31:0]	<p>在屏蔽之前，通过 VICINTENABLE 和 VICINTSELECT 寄存器显示 FIQ 中断的状态</p> <p>0=屏蔽之前中断不被激活（复位）</p> <p>1=屏蔽之前中断被激活</p> <p>因为原始寄存器可以直接看到原始的输入中断，因此不知道复位值。</p> <p>每个中断源都有一个寄存器位。</p>	0x0

#### 12.5.4.中断选择寄存器, VICINTSELET

寄存器	地址	读/写	描述	复位值
VIC0INTSELECT	0x7120_000C	读/写	中断选择寄存器(VIC0)	0x0000_0000
VIC1INTSELECT	0x7130_000C	读/写	中断选择寄存器(VIC1)	0x0000_0000

名称	位	描述	复位值
IntSelect	[31:0]	<p>为中断请求选择中断的状态</p> <p>0=IRQ 中断（复位）</p> <p>1=FIQ 中断</p> <p>每个中断源都有一个寄存器位。</p>	0x0

#### 12.5.5.中断使能寄存器, VICINTENABLE

寄存器	地址	读/写	描述	复位值
VIC0INTENABLE	0x7120_0010	读/写	中断使能寄存器(VIC0)	0x0000_0000
VIC1INTENABLE	0x7130_0010	读/写	中断使能寄存器(VIC1)	0x0000_0000



名称	位	描述	复位值
IntEnable	[31:0]	<p>使能中断请求，允许中断到达处理器</p> <p>读：</p> <p>0=中断禁止（复位）</p> <p>1=中断使能</p> <p>中断使能只能用寄存器设置。VICINTENCLEAR 寄存器用来清除中断使能。</p> <p>写：</p> <p>0=没有影响</p> <p>1=中断使能</p> <p>每个中断源都有一个寄存器位。</p>	0x0

## 12.5.6.中断使能清除，VICINTENCLEAR

寄存器	地址	读/写	描述	复位值
VIC0INTENCLEAR	0x7120_0014	写	中断使能清除寄存器(VIC0)	-
VIC1INTENCLEAR	0x7130_0014	写	中断使能清除寄存器(VIC1)	-

名称	位	描述	复位值
IntEnable Clear	[31:0]	<p>在 VICINTENABLE 寄存器内清除相应的位</p> <p>0=没有影响（复位）</p> <p>1=在 VICINTENABLE 寄存器内中断 disabled</p> <p>每个中断源都有一个寄存器位。</p>	-

12.5.7.软件中断寄存器，VICSOFTINT

寄存器	地址	读/写	描述	复位值
VIC0SOFTINT	0x7120_0018	读/写	软件中断寄存器(VIC0)	0x0000_0000
VIC1SOFTINT	0x7130_0018	读/写	软件中断寄存器(VIC1)	0x0000_0000

名称	位	描述	复位值
IntEnable	[31:0]	在中断屏蔽之前设置 HIGH 位对选择的源产生软件中断  读： 0=软件中断不被激活（复位） 1=软件中断被激活  写： 0=没有影响 1=软件中断使能  每个中断源都有一个寄存器位。	0x0

12.5.8.软件中断清除寄存器，VICSOFTINTCLEAR

寄存器	地址	读/写	描述	复位值
VIC0SOFTINTENCLEAR	0x7120_001C	写	软件中断清除寄存器 (VIC0)	-
VIC1SOFTINTENCLEAR	0x7130_001C	写	软件中断清除寄存器 (VIC1)	-

名称	位	描述	复位值
SoftInt Clear	[31:0]	在 VICSOFTINT 寄存器内清除相应的位  0=没有影响（复位） 1=在 VICSOFTINT 寄存器内中断 disabled	-

		每个中断源都有一个寄存器位。	
--	--	----------------	--

### 12.5.9.保护使能寄存器，VICPROTECTION

寄存器	地址	读/写	描述	复位值
VIC0PROTECTION	0x7120_0020	读/写	保护使能寄存器(VIC0)	0x0000_0000
VIC1PROTECTION	0x7130_0020	读/写	保护使能寄存器(VIC1)	0x0000_0000

名称	位	描述	复位值
Reserved	[31:1]	保留，作为 0 读取，不要修改	0x0
IntEnable	[0]	使能或禁止保护寄存器访问：  0=保护模式禁止（复位）  1=保护模式使能  当保护模式使能时，只有特权模式可以访问（进行读和写）中断控制寄存器。当保护模式禁止时，用户模式和特权模式都可以访问寄存器。  当保护模式禁止时，这个寄存器只能在特权模式下被访问。	0

### 12.5.10 软件优先级屏蔽寄存器，VICSWPRIORITYMASK

寄存器	地址	读/写	描述	复位值
VIC0SWPRIORITYMASK	0x7120_0024	读/写	软件优先级屏蔽寄存器(VIC0)	0x0000_FFFF
VIC1SWPRIORITYMASK	0x7130_0024	读/写	软件优先级屏蔽寄存器(VIC1)	0x0000_FFFF

名称	位	描述	复位值
Reserved	[31:16]	保留，作为 0 读取，不要修改	0x0
SWPriorityMask	[15:0]	控制 16 位中断信号优先级软件屏蔽	0xFFFF

		0=中断优先级被屏蔽 1=中断优先级未被屏蔽 寄存器的位于 16 位中断优先级相适应。	
--	--	---	--

### 12.5.11.菊花链矢量优先寄存器

寄存器	地址	读/写	描述	复位值
VIC0PRIORITYDAISY	0x7120_0028	读/写	菊花链矢量优先寄存器 (VIC0)	0x0000_000F
VIC1PRIORITYDAISY	0x7130_0028	读/写	菊花链矢量优先寄存器 (VIC1)	0x0000_000F

名称	位	描述	复位值
Reserved	[31:16]	保留，作为 0 读取，不要修改	0x0
SWPriorityMask	[15:0]	选择矢量中断优先级。可以选择 16 进制数 0~15 范围内的任何一个矢量中断优先级值运行寄存器。	0xF

### 12.5.12. 矢量地址寄存器，VICVECTADDR[0-31]

寄存器	地址	读/写	描述	复位值
VIC0VECTADDR[31:0]	0x7120_0100 ~0x7120_017C	读/写	矢量地址[31:0]寄存器(VIC0)	0x0000_0000
VIC1 VECTADDR[31:0]	0x7130_0100 ~0x7130_017C	读/写	矢量地址[31:0]寄存器(VIC1)	0x0000_0000

名称	位	描述	复位值
VectorAddr	[31:0]	包含 ISR 矢量地址	0x0000_0000

### 12.5.13. 矢量优先寄存器，VICVECTRPRIORITY[0-31]

寄存器	地址	读/写	描述	复位值
VIC0VECTRPRIORITY[31:0]	0x7120_0200 ~0x7120_027C	读/写	矢量优先[31:0]寄存器(VIC0)	0x0000_000F
VIC1 VECTRPRIORITY[31:0]	0x7130_0200 ~0x7130_027C	读/写	矢量优先[31:0]寄存器(VIC1)	0x0000_000F

名称	位	描述	复位值
Reserved	[31:4]	保留，作为 0 读取，不要修改	0x0
VectorAddr	[3:0]	选择矢量中断优先级。可以选择 16 进制数 0~15 范围内的任何一个矢量中断优先级值运行寄存器。	0x0000_0000

### 12.5.14. 矢量地址寄存器，VICADDRESS

寄存器	地址	读/写	描述	复位值
VIC0ADDRESS	0x7120_0F00	读/写	矢量地址寄存器(VIC0)	0x0000_0000
VIC1ADDRESS	0x7130_0F00	读/写	矢量地址寄存器(VIC1)	0x0000_0000

名称	位	描述	复位值
VectAddr	[31:0]	包含当前激活的 ISR 地址，复位值是 0x00000000  寄存器的读取操作可以返回 ISR 的地址，设置当前中断处于正在服务状态。  只有当有激活中断的时候可以进行读操作。  向寄存器写入任何值都可以清除当前中断。只有在终端服务快要结束的时候才可以进行写入操作。	0x0

## 13 安全子系统

### 13.1 概述

安全子系统是加密功能的加速器。安全子系统架构提供高速数据处理功能，有双层的 AHB 总线和 FIFOs。可以对 FIFO-Rx 和 FIFO-Tx 进行编程，监测 AES 或 DES/3DES 或 SHA-1/PRNG 模块。FIFO-Rx 和 FIFO-Tx 从目标模块内自动转换输入输出数据。安全子系统不需要 CPU 便可以完成高速数据处理。

图 13-1 为 安全子系统模块图

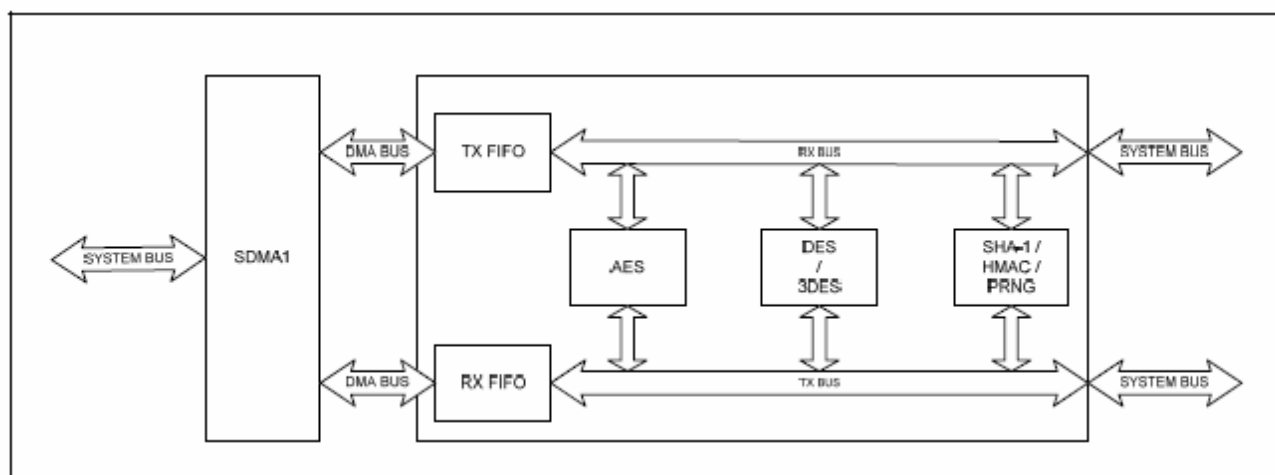


图 13-1 安全子系统模块图

安全子系统的主要性能有：

- (1) 对称密钥加速器：AES： 支持 ECB, CBC, CTR 模式  
DES/3DES: 支持 ECB, CBC 模式
- (2) Hash 引擎：支持 SHA-1  
支持 H/W HMAC
- (3) 随机数发生器：每 160 周期产生 PRNG 320 位
- (4) FIFO-Rx/Tx： 两个 32 字的输入输出流

### 13.1.1 AES 编程向导

可以根据数据和 Meta 数据转换方式，通过 FIFO 操作或 ARM 操作执行 AES。这里描述 FIFO 基本操作。

1.在寄存器内写入密钥和 Meta 数据。根据密钥大小向密钥寄存器内写入密钥。

2.在 AES 控制寄存器内，设置 AES 操作的 AES 配置密钥长度，设置 AES 操作的防线，操作模式及计数器大小。

3.Rx/Tx FIFO 相关寄存器的设置遵循以下规则：

(1) FIFO-Rx 信息长度寄存器设置：AES 操作的整体信息的长度（32 位）。

(2) FIFO-Rx 模块尺寸寄存器设置：AES 运算法则特殊模块尺寸（32 位）

(3) FIFO-Rx 目标寄存器设置：AES 输入缓冲区开始地址。

(4) FIFO-Tx 信息设置：AES 操作的整体信息的长度。

(5) FIFO-Tx 模块尺寸寄存器设置：AES 特殊模块尺寸（32 位）

(6) FIFO-Tx 目标寄存器设置：AES 输入缓冲区开始地址。

(7) FIFO-Rx 控制寄存器设置：1) FRx\_Host\_Module (FRx\_Ctrl[7:6]: 选择特殊算法目标模块 (AES:00) ), 2) FRx\_Host\_Rd\_En(FRx\_Ctrl[5]:主机 FRx\_Ctrl[31:16]和 FTx\_MlenCnt 区域的读使能), 3) FTx\_Host\_Wr\_En(FRx\_Ctrl[4]:主机 FRx\_Rx 写使能), 4)FRx\_Sync\_Tx(FRx\_Ctrl[3]: FTx\_Rx 写等待使能), 5) FTx\_Start(FRx\_Ctrl[0]:目标模块 FRx\_Rx 数据转换器开始)

(8) FiFo-Tx 控制寄存器设置：1) FRx\_Src\_Module (FRx\_Ctrl[7:6]: 选择特殊运算法则源模块 (AES:00) ), 2) FRx\_Host\_Rd\_En(FRx\_Ctrl[5]:主机 FRx\_Ctrl[31:16]和 FTx\_MlenCnt 区域的读使能), 3) FTx\_Host\_Wr\_En(FRx\_Ctrl[4]:主机 FRx\_Rx 写使能), 4)FTx\_Start(FRx\_Ctrl[0]: 源模块 FRx\_Rx 数据转换器开始)

4. AES 操作中，向 FiFo\_Rx 内写入数据。FiFo\_Rx 和 FiFo\_Tx 通过以下过程完成数据和 AES 的通信。

(1) 向 AES\_Rx\_DIN\_0 寄存器写入数据

(2) 开始 AES 运行：根据需要重复下面步骤的第一步到第二步

1) FiFo\_Rx 准备轮询 AES 输入

2) FiFo\_Rx 向 AES\_Rx\_DIN\_0 寄存器写入数据

3) FiFo\_Rx 准备轮询 AEC 输出

4) FiFo\_Rx 从 AES\_Rx\_DIN\_0 寄存器内读取数据

### 5) AES 运行开始

(3) FiFo\_Rx 准备轮询 AES 输出

(4) FiFo\_Rx 从 AES\_Rx\_DIN\_0 寄存器内读取数据

5.应用 FiFo\_Rx 空检测从 FiFo\_Rx 到 AES 的数据传输是否完成。

6.通过检测 FiFo\_Rx 的转换运行检测，确认 AES 操作的结果传送到 FiFo\_Rx 内，并从 FiFo\_Rx 内读取 AES 的运行结果。

## 13.1.2 TDES

可以根据数据和 Meta 数据转换方式，通过 FIFO 操作或 ARM 操作执行 TDES。这里描述 FIFO 基本操作。

1.向寄存器内写入密钥、Meta 数据和 ECB 模式。设置 TDES 控制寄存器的 DES/TDES 选择、方向、或操作方式。

2. 遵循以下规则设置 Rx/Tx FIFO 相关寄存器：

(1) FIFO-Rx 信息长度寄存器设置：TDES 操作的整体信息的长度（32 位）。

(2) FIFO-Rx 模块尺寸寄存器设置：TDES 运算法则特殊模块尺寸（32 位）

(3) FIFO-Rx 目标寄存器设置：TDES 输入缓冲区开始地址。

(4) FIFO-Tx 信息设置：TDES 操作整体信息的长度。

(5) FIFO-Tx 模块尺寸寄存器设置：TDES 特殊模块尺寸（32 位）

(6) FIFO-Tx 目标寄存器设置：TDES 输入缓冲区开始地址。

(7) FIFO-Rx 控制寄存器设置：1) FRx\_Dest\_Module(FRx\_Ctrl[7:6]: 选择特殊算法目标模块 (TDES/DES:01:00) ), 2) FRx\_Dest\_Rd\_En(FRx\_Ctrl[5]:主机 FRx\_Ctrl[31:16]和 FTx\_MlenCnt 区域的读使能), 3) FTx\_Host\_Wr\_En(FRx\_Ctrl[4]:主机 FRx\_Rx 写使能), 4)FRx\_Sync\_Tx(FRx\_Ctrl[3]: FTx\_Rx 写等待使能), 5) FTx\_Start(FRx\_Ctrl[0]:目标模块 FRx\_Rx 数据转换器开始)

(8) FiFo-Tx 控制寄存器设置：1) FRx\_Src\_Module (FRx\_Ctrl[7:6]: 选择特殊运算法则源模块 (TDES/DES:01:00) ), 2) FRx\_Host\_Rd\_En(FRx\_Ctrl[5]:主机 FRx\_Ctrl[31:16]和 FTx\_MlenCnt 区域的读使能), 3) FTx\_Host\_Wr\_En(FRx\_Ctrl[4]:主机 FRx\_Rx 写使能), 4)FTx\_Start(FRx\_Ctrl[0]: 源模块 FRx\_Rx 数据转换器开始)

3.TDES 操作时，主机向 FiFo\_Rx 内写入数据。FiFo\_Rx 和 FiFo\_Tx 通过以下过程完成数据和 TDES



之间的通信。

- (1) 向 TDES\_Rx\_INPUT\_0 寄存器写入数据
- (2) 开始 TDES 运行：根据需要重复下面步骤的第一步到第二步
  - 1) FiFo\_Rx 准备轮询 TDES 输入
  - 2) FiFo\_Rx 向 TDES\_Rx\_INPUT\_0 寄存器写入数据
  - 3) FiFo\_Rx 准备轮询 TDEC 输出
  - 4) FiFo\_Rx 从 TDES\_Rx\_INPUT\_0 寄存器内读取数据
  - 5) TDES 运行开始

(3) FiFo\_Rx 准备轮询 AES 输出

(4) FiFo\_Rx 从 TDES\_Rx\_INPUT\_0 寄存器内读取数据

4.应用 FiFo\_Rx 空检检测 TDES 操作的所有数据是否从 FiFo\_Rx 传输到 TDES。

5.通过检测 FiFo\_Rx 的转换运行检测，确认 AES 操作的结果传送到 FiFo\_Rx 内，并从 FiFo\_Rx 内读取 AES 的运行结果。

### 13.1.3 SHA-1&PRNG

根据数据和 Meta 数据转换方式，通过 FIFO 操作或 ARM 操作执行 SHA-1 和 PRNG。这里描述 FIFO 基本操作。

1.配置 HASH\_CTRL 寄存器(选择引擎 (SHA-1, HMAC\_SHA-1 , PRNG)，密钥或文本、引擎开始等等)

2. 遵循以下规则设置 Rx/Tx FIFO 相关寄存器：

- (1) FIFO-Rx 信息长度寄存器设置：操作的整体信息的长度（32 位）。
- (2) FIFO-Rx 模块尺寸寄存器设置：Hash 运算法则特殊模块尺寸（32 位）
- (3) FIFO-Rx 目标寄存器设置：输入缓冲区开始地址。
- (4) FIFO-Tx 信息设置：操作的整体信息的长度。
- (5) FIFO-Tx 模块尺寸寄存器设置：Hash 运算法则特殊模块尺寸（32 位）
- (6) FIFO-Tx 目标寄存器设置：输入缓冲区开始地址。

(7) FIFO-Rx 控制寄存器设置：1) FRx\_Dest\_Module(FRx\_Ctrl[7:6]: 选择特殊算法目标模块 (SHA-1/PRNG:01) )，2) FRx\_Host\_Rd\_En(FRx\_Ctrl[5]:主机 FRx\_Ctrl[31:16]和 FTx\_MlenCnt 区域的读

使能), 3) FTx\_Host\_Wr\_En(FRx\_Ctrl[4]:主机 FRx\_Rx 写使能), 4)FRx\_Sync\_Tx(FRx\_Ctrl[3]: FTx\_Rx 写等待使能), 5) FTx\_Start(FRx\_Ctrl[0]:目标模块 FRx\_Rx 数据转换器开始)

(8) FiFo-Tx 控制寄存器设置: 1) FRx\_Src\_Module (FRx\_Ctrl[7: 6]: 选择特殊运算法则源模块 (SHA-1/PRNG:01)) ,2) FRx\_Host\_Rd\_En(FRx\_Ctrl[5]:主机 FRx\_Ctrl[31:16]和 FTx\_MlenCnt 区域的读使能), 3) FTx\_Host\_Wr\_En(FRx\_Ctrl[4]:主机 FRx\_Rx 写使能), 4)FTx\_Start(FRx\_Ctrl[0]: 源模块 FRx\_Rx 数据转换器开始)

3.Hash 操作时, 向 FiFo\_Rx 内写入数据。FiFo\_Rx 和 FiFo\_Tx 通过以下过程完成数据和 Hash 的通信。

- (1) 向 Hash\_Rx\_DIN\_0~ Hash\_Rx\_DIN\_15 寄存器写入数据
  - (2) 开始运行: 根据需要重复下面步骤的第一步到第二步
    - 1) FiFo\_Rx 准备轮询输入
    - 2) FiFo\_Rx 向 Hash\_Rx\_DIN\_0~ Hash\_Rx\_DIN\_15 寄存器写入数据
  - (3) FiFo\_Rx 准备轮询 SHA-1\_PRNG 输出
  - (4) FiFo\_Rx 从 HASH\_OUTPUT\_0 寄存器内读取数据
- 4.应用 FiFo\_Rx 空检测数据是否从 FiFo\_Rx 传输到 SHA-1/PRNG。

5.通过检测 FiFo\_Rx 转换器运行确认 SHA-1/PRNG 操作的结果传送到 FiFo\_Rx 内, 并从 FiFo\_Rx 内读取运行结果。

## 13.2 特殊功能寄存器

### 1.安全子系统寄存器映射

表 13-2 DMA 和中断控制寄存器映射

地址	读/写	复位值	名称	描述
基本地址+0x00	读/写	0x0000_0000	Dn1_CFG	DMA 和终端配置寄存器
基本地址=0x7D00_0000				

表 13-3 FiFo\_Rx 寄存器映射

地址	读/写	复位值	名称	描述
基本地址+0x00	读/写	0x0420_0000	FRx_Ctrl	FiFo-Rx 控制和状态寄存器
基本地址+0x04	读/写	0x0000_0000	FRx_Mlen	FiFo-Rx 信息长度寄存器

基本地址+0x08	读/写	0x0000_0000	FRx_BlkSz	FIFO-Rx 加密算法模块尺寸寄存器
基本地址+0x0C	读/写	0x0000_0000	FRx_DestAddr	FIFO-Rx 输入缓冲地址寄存器
基本地址+0x10	读/写	0x0000_0000	FRx_MlenCnt	FIFO-Rx 信息计数寄存器
基本地址+0x40	写	0x0000_0000	FRx_WrBuf	FIFO-Rx 写缓冲区
...	...	...	...	...
基本地址+0x7C	写	0x0000_0000	FRx_WrBuf	FIFO-Rx 写缓冲区
基本地址=0x7D40_0000				
基本地址=0x7D90_0000				

注：写访问 FTx\_WrBuf 使 FIFO-Tx 忽略已给的地址向 FIFO 存储器内写入数据。这就意味着在 0x0040 和 0x0080 之间的任何地址将会触发 FIFO 存储器的读操作。这个性能用突发写操作向 FIFO-Tx 产生程序。

表 13-4 FIFO-Tx 寄存器映射

地址	读/写	复位值	名称	描述
基本地址+0x00	读/写	0x0420_0000	FTx_Ctrl	FIFO-Rx 控制和状态寄存器
基本地址+0x04	读/写	0x0000_0000	FTx_Mlen	FIFO-Tx 信息长度寄存器
基本地址+0x08	读/写	0x0000_0000	FTx_BlkSz	FIFO-Tx 加密算法模块尺寸寄存器
基本地址+0x0C	读/写	0x0000_0000	FTx_DestAddr	FIFO-Tx 输入缓冲地址寄存器
基本地址+0x10	读/写	0x0000_0000	FTRx_MlenCnt	FIFO-Tx 信息计数寄存器
基本地址+0x40	读	0x0000_0000	FTx_RdBuf	FIFO-Tx 读缓冲区
...	...	...	...	...
基本地址+0x7C	读	0x0000_0000	FTx_RdBuf	FIFO-Tx 读缓冲区
基本地址=0x7D80_0000				
基本地址=0x7DA0_0000				

注：读访问 FTx\_WrBuf 使 FIFO-Tx 忽略已给的地址从 FIFO 存储器内读取数据。这就意味着在 0x0040 和 0x007c 之间的任何地址将会触发 FIFO 存储器的读操作。这个性能用突发读操作向 FIFO-Tx 产生程序。

表 13-5 AES 寄存器映射

地址	读/写	复位值	名称	描述
Rx-AES 寄存器映射（RX 部分）				
基本地址+0x00	读/写	0x0000_0200	AES_Rx_Ctrl	AES Rx 控制和状态寄存器

基本地址+0x10	读/写	0x0000_0000	AES_Rx_DIN_01	AES Rx 数据输入寄存器 01
基本地址+0x14	读/写	0x0000_0000	AES_Rx_DIN_02	AES Rx 数据输入寄存器 02
基本地址+0x18	读/写	0x0000_0000	AES_Rx_DIN_03	AES Rx 数据输入寄存器 03
基本地址+0x1C	读/写	0x0000_0000	AES_Rx_DIN_04	AES Rx 数据输入寄存器 04
基本地址+0x20	读	0x0000_0000	AES_Rx_DOUT_01	AES Rx 数据输出寄存器 01
基本地址+0x24	读	0x0000_0000	AES_Rx_DOUT_02	AES Rx 数据输出寄存器 02
基本地址+0x28	读	0x0000_0000	AES_Rx_DOUT_03	AES Rx 数据输出寄存器 03
基本地址+0x2C	读	0x0000_0000	AES_Rx_DOUT_04	AES Rx 数据输出寄存器 04
基本地址+0x80	读/写	0x0000_0000	AES_Rx_KEY_01	AES Rx 密钥输入寄存器 01
基本地址+0x84	读/写	0x0000_0000	AES_Rx_KEY_02	AES Rx 密钥输入寄存器 02
基本地址+0x88	读/写	0x0000_0000	AES_Rx_KEY_03	AES Rx 密钥输入寄存器 03
基本地址+0x8C	读/写	0x0000_0000	AES_Rx_KEY_04	AES Rx 密钥输入寄存器 04
基本地址+0x90	读/写	0x0000_0000	AES_Rx_KEY_05	AES Rx 密钥输入寄存器 05
基本地址+0x94	读/写	0x0000_0000	AES_Rx_KEY_06	AES Rx 密钥输入寄存器 06
基本地址+0x98	读/写	0x0000_0000	AES_Rx_KEY_07	AES Rx 密钥输入寄存器 07
基本地址+0x9C	读/写	0x0000_0000	AES_Rx_KEY_08	AES Rx 密钥输入寄存器 08
基本地址+0xA0	读/写	0x0000_0000	AES_Rx_IV_01	AES Rx IV 输入寄存器 01
基本地址+0xA4	读/写	0x0000_0000	AES_Rx_IV_02	AES Rx IV 输入寄存器 02
基本地址+0xA8	读/写	0x0000_0000	AES_Rx_IV_03	AES Rx IV 输入寄存器 03
基本地址+0xAC	读/写	0x0000_0000	AES_Rx_IV_04	AES Rx IV 输入寄存器 04
基本地址+0xB0	读/写	0x0000_0000	AES_Rx_CTR_01	AES Rx 计数器预算寄存器 01
基本地址+0xB4	读/写	0x0000_0000	AES_Rx_CTR_02	AES Rx 计数器预算寄存器 02
基本地址+0xB8	读/写	0x0000_0000	AES_Rx_CTR_03	AES Rx 计数器预算寄存器 03
基本地址+0xBC	读/写	0x0000_0000	AES_Rx_CTR_04	AES Rx 计数器预算寄存器 04
Tx-AES 寄存器映射 (TX 部分)				
基本地址+0x20	读	0x0000_0000	AES_Tx_DIN_01	AES Tx 数据输入寄存器 01
基本地址+0x24	读	0x0000_0000	AES_Tx_DIN_02	AES Tx 数据输入寄存器 02
基本地址+0x28	读	0x0000_0000	AES_Tx_DIN_03	AES Tx 数据输入寄存器 03

基本地址+0x2C	读	0x0000_0000	AES_Tx_DIN_04	AES Tx 数据输入寄存器 04
Rx 基本地址=0x7D10_0000 Tx 基本地址=0x7D50_0000				

表 13-6 DES/TDES 寄存器映射

地址	读/写	复位值	名称	描述
Rx-DES/3DES 寄存器映射 (RX 部分)				
基本地址+0x00	读/写	0x0000_0040	TDES_Rx_CTRL	DES/3DES Rx 控制和状态寄存器
基本地址+0x10	读/写	0x0000_0000	TDES_Rx_KEY1_0	DES/3DES 密钥输入寄存器 1_0
基本地址+0x14	读/写	0x0000_0000	TDES_Rx_KEY1_1	DES/3DES 密钥输入寄存器 1_1
基本地址+0x18	读/写	0x0000_0000	TDES_Rx_KEY2_0	DES/3DES 密钥输入寄存器 2_0
基本地址+0x1C	读/写	0x0000_0000	TDES_Rx_KEY2_1	DES/3DES 密钥输入寄存器 2_1
基本地址+0x20	读/写	0x0000_0000	TDES_Rx_KEY3_0	DES/3DES 密钥输入寄存器 3_0
基本地址+0x24	读/写	0x0000_0000	TDES_Rx_KEY3_1	DES/3DES 密钥输入寄存器 3_1
基本地址+0x40	读/写	0x0000_0000	TDES_Rx_INPUT_0	DES/3DES 数据输入寄存器 0
基本地址+0x44	读/写	0x0000_0000	TDES_Rx_INTPUT_1	DES/3DES 数据输入寄存器 1
基本地址+0x48	读	0x0000_0000	TDES_Rx_OUTPUT_0	DES/3DES Rx 数据输出寄存器 0
基本地址+0x4C	读	0x0000_0000	TDES_Rx_OUTPUT_1	DES/3DES Rx 数据输出寄存器 1
基本地址+0x50	读/写	0x0000_0000	TDES_Rx_IV_0	TDES Rx IV 输入寄存器 0
基本地址+0x54	读/写	0x0000_0000	TDES_Rx_IV_1	TDES Rx IV 输入寄存器 1
Tx-DES/3DES 寄存器映射 (TX 部分)				
基本地址+0x48	读	0x0000_0000	TDES_Tx_OUTPUT_0	DES/3DES Tx 数据输出寄存器 0
基本地址+0x4C	读	0x0000_0000	TDES_Tx_OUTPUT_1	DES/3DES Tx 数据输出寄存器 1
Rx 基本地址=0x7D20_0000 Tx 基本地址=0x7D60_0000				

表 13-7 SHA-1/PRNG 寄存器映射

地址	读/写	复位值	名称	描述
Rx-SHA-1/PRNG 寄存器映射 (RX 部分)				
基本地址+0x00	读/写	0x0000_0000	HASH_CONTROL	Hash 引擎控制寄存器
基本地址+0x04	读/写	0x0000_0000	HASH_DATA	Hash 数据或 HMAC 密钥输入寄存器
基本地址+0x08	读/写	0x0000_0000	HASH_DATA	Hash 数据或 HMAC 密钥输入寄存器
...	...	...	...	...
基本地址+0x20	读/写	0x0000_0000	HASH_DATA	Hash 数据或 HMAC 密钥输入寄存器
基本地址+0x08	读/写	0x0000_0000	SEED_DATA_01	PRNG 种子数据[31:0]
基本地址+0x0C	读/写	0x0000_0000	SEED_DATA_02	PRNG 种子数据[63:32]
基本地址+0x10	读/写	0x0000_0000	SEED_DATA_03	PRNG 种子数据[95:64]
基本地址+0x14	读/写	0x0000_0000	SEED_DATA_04	PRNG 种子数据[127:96]
基本地址+0x18	读/写	0x0000_0000	SEED_DATA_05	PRNG 种子数据[159:128]
基本地址+0x1C	读/写	0x0000_0000	SEED_DATA_06	PRNG 种子数据[191:160]
基本地址+0x20	读/写	0x0000_0000	SEED_DATA_07	PRNG 种子数据[223:192]
基本地址+0x24	读/写	0x0000_0000	SEED_DATA_08	PRNG 种子数据[255:224]
基本地址+0x28	读/写	0x0000_0000	SEED_DATA_09	PRNG 种子数据[287:256]
基本地址+0x2C	读/写	0x0000_0000	SEED_DATA_10	PRNG 种子数据[319:288]
基本地址+0x30	读	0x0000_0010	HASH_STATUS	状态检测
基本地址+0x34	读	0x0000_0000	HASH_OUTPUT_01 PRNG_OUTPUT_01	HASH (PRNG) 输出 (h0)
基本地址+0x38	读	0x0000_0000	HASH_OUTPUT_02 PRNG_OUTPUT_02	HASH (PRNG) 输出 (h1)
基本地址+0x3C	读	0x0000_0000	HASH_OUTPUT_03 PRNG_OUTPUT_03	HASH (PRNG) 输出 (h2)
基本地址+0x40	读	0x0000_0000	HASH_OUTPUT_04 PRNG_OUTPUT_04	HASH (PRNG) 输出 (h3)
基本地址+0x44	读	0x0000_0000	HASH_OUTPUT_05 PRNG_OUTPUT_05	HASH (PRNG) 输出 (h4)

基本地址+0x48	读	0x0000_0000	HASH_OUTPUT_06	PRNG 输出
基本地址+0x4C	读	0x0000_0000	HASH_OUTPUT_07	PRNG 输出
基本地址+0x50	读	0x0000_0000	HASH_OUTPUT_08	PRNG 输出
基本地址+0x54	读	0x0000_0000	HASH_OUTPUT_09	PRNG 输出
基本地址+0x58	读	0x0000_0000	HASH_OUTPUT_10	PRNG 输出
基本地址+0x5C	读	0x0000_0000	HASH_MIDOUT_01	HASH_MIDOUT[159:128]
基本地址+0x60	读	0x0000_0000	HASH_MIDOUT_02	HASH_MIDOUT[127: 96]
基本地址+0x64	读	0x0000_0000	HASH_MIDOUT_03	HASH_MIDOUT[95: 64]
基本地址+0x68	读	0x0000_0000	HASH_MIDOUT_04	HASH_MIDOUT[63: 32]
基本地址+0x6C	读	0x0000_0000	HASH_MIDOUT_05	HASH_MIDOUT[31: 0]
基本地址+0x70	写	0x0000_0000	HASH_IV_01	HASH 初始值 01
基本地址+0x74	写	0x0000_0000	HASH_IV_02	HASH 初始值 02
基本地址+0x78	写	0x0000_0000	HASH_IV_03	HASH 初始值 03
基本地址+0x7C	写	0x0000_0000	HASH_IV_04	HASH 初始值 04
基本地址+0x80	写	0x0000_0000	HASH_IV_05	HASH 初始值 01
基本地址+0x84	写	0x0000_0000	PRE_MSG_LENGT_1	HASH 长度[63:32]
基本地址+0x88	写	0x0000_0000	PRE_MSG_LENGT_2	HASH 长度[31:0]
Tx-SHA-1/PRNG 寄存器映射 (TX 部分)				
基本地址+0x34	读	0x0000_0000	HASH_OUTPUT_01 PRNG_OUTPUT_01	HASH (PRNG) 输出 (h0)
基本地址+0x38	读	0x0000_0000	HASH_OUTPUT_02 PRNG_OUTPUT_02	HASH (PRNG) 输出 (h1)
基本地址+0x3C	读	0x0000_0000	HASH_OUTPUT_03 PRNG_OUTPUT_03	HASH (PRNG) 输出 (h2)
基本地址+0x40	读	0x0000_0000	HASH_OUTPUT_04 PRNG_OUTPUT_04	HASH (PRNG) 输出 (h3)
基本地址+0x44	读	0x0000_0000	HASH_OUTPUT_05 PRNG_OUTPUT_05	HASH (PRNG) 输出 (h4)

基本地址+0x48	读	0x0000_0000	HASH_OUTPUT_06	PRNG 输出
基本地址+0x4C	读	0x0000_0000	HASH_OUTPUT_07	PRNG 输出
基本地址+0x50	读	0x0000_0000	HASH_OUTPUT_08	PRNG 输出
基本地址+0x54	读	0x0000_0000	HASH_OUTPUT_09	PRNG 输出
基本地址+0x58	读	0x0000_0000	HASH_OUTPUT_10	PRNG 输出
基本地址+0x5C	读	0x0000_0000	HASH_MIDOUT_01	HASH_MIDOUT[159:128]
基本地址+0x60	读	0x0000_0000	HASH_MIDOUT_02	HASH_MIDOUT[127: 96]
基本地址+0x64	读	0x0000_0000	HASH_MIDOUT_03	HASH_MIDOUT[95: 64]
基本地址+0x68	读	0x0000_0000	HASH_MIDOUT_04	HASH_MIDOUT[63: 32]
基本地址+0x6C	读	0x0000_0000	HASH_MIDOUT_05	HASH_MIDOUT[31: 0]
Rx 基本地址=0x7D20_0000				
Tx 基本地址=0x7D60_0000				

### 13.3 DMA 和中断控制模版

#### 安全子系统 DMA 和中断寄存器

寄存器	地址	读/写	描述	复位值
Dnl_Cfg	0x7D00_0000	读/写	DMA 与中断配置控制寄存器和状态寄存器	0x0000_0000

Dnl_Cfg	位	描述	初始状态
WrPrivMismatch	[31]	SFR 的写访问权限不匹配状态位。如果设置' 1' , 发生 SFR 的写访问权限不匹配状态。	0b
RdPrivMismatch	[30]	SFR 的读访问权限不匹配状态位。如果设置' 1' , 发生 SFR 的读访问权限不匹配状态。	0b
Reserved	[29:23]	保留	0x00
SHA_intr_status	[22]	SHA-1/PRNG 中断状态和悬挂位。当读操作发生时, 此位被清除。	0b
DES_intr_status	[21]	DES/3DES 中断状态和悬挂位。当读操作发生时,	0b



		此位被清除。	
AES_intr_status	[20]	AES 中断状态和悬挂位。当读操作发生时，此位被清除。	0b
Reserved	[19:18]	保留	00b
FTX_intr_status	[17]	FIFO-TX 中断状态和悬挂位。当读操作发生时，此位被清除。	0b
FRX_intr_status	[16]	FIFO-RX 中断状态和悬挂位。当读操作发生时，此位被清除。	0b
Reserved	[15]	保留	00b
SHA_intr_En	[14]	结束操作以后 SHA-1/PRNG 中断状态使能。	0b
DES_intr_En	[13]	结束操作后 DES/3DES 中断状态使能	0b
AES_intr_En	[12]	结束操作后 AES 中断状态使能	0b
Reserved	[11: 10]	保留	00b
FTx_intr_En	[9]	结束操作以后 FIFO-Tx 中断状态使能。	0b
FRx_intr_En	[8]	结束操作后 FIFO-Rx 中断状态使能	0b
TxTrgLevel	[7:5]	Tx 部分触发电平设置 000=1 个字节 001=4 个字节 010=8 个字节 011= 12 个字节 100= 16 个字节 101=20 个字节 110=24 个字节 111=28 个字节	000b
TxDmaEnb	[4]	Tx 部分 DMS 使能位（1：使能）	0b
RxTrgLevel	[3:1]	Rx 部分触发电平设置 000=1 个字节 001=4 个字节 010=8 个字节 011= 12 个字节 100= 16 个字节 101=20 个字节 110=24 个字节 111=28 个字节	000b
RxDmaEnb	[0]	Tx 部分 DMS 使能位（1：使能）	0b

# 13.4 安全子系统 RX FIFO 模快

## 1. FIFO-RX 控制寄存器

寄存器	地址	读/写	描述	复位值
FRx_Ctrl	0x7D40_0000	读/写	FIFO-Rx 控制/状态寄存器（只可以读取 16 位 数据）	0x0420_0000

FRx_Ctrl	位	描述	初始状态
FRx_WrPrivError	[31]	如果写访问 FIFO-Rx 导致特权错误此位设置为 1。（主机和命令不允许访问 FIFO-Rx）	0b
FRx_RdPrivError	[30]	如果读访问 FIFO-Rx 导致特权错误此位设置为 1。（主机和命令不允许访问 FIFO-Rx）	0b
Reserved	[29;28]	保留	0b
FRx_Full	[27]	如果 FIFO-Rx 缓冲区满时设置此位为 1	0b
FRx_Empty	[26]	如果 FIFO-Rx 缓冲区空时设置此位为 1	0b
FRx_Done	[25]	如果 FIFO-Rx 完成向目标的数据 FRx_Mlen 字节的转换，设置此位为 1	0b
FRx_Running	[24]	如果 FIFO-Rx 完成目标的数据转换设置此位为 1，或者准备好目标输入缓冲等待时，设置此位为 1。当 FRx_Start 位复位为 0 时，设置此位为 1。	0b
FRx_Wd2Write	[23; 16]	可以写入 FIFO 存储器的字节数（FRx_WrBuf）	0x00
FRx_Wd2Read	[15; 8]	可以从 FIFO 存储器内读取的字节数（FRx_WrBuf）	0x00
FRx_Dest_Module	[7; 6]	选择目标模块 (00: AES, 01:DES/3DES, 10:SHA-1/PRNG, 11:保留)	0b
FRx_Host_Rd_En	[5]	主机可以从 FRx_Ctrl[31:16]和 FRx_MlenCnt 内读取数据。	0b
FRx_Host_Wr_En	[4]	主机可以 FRx_WrBuf 内写入数据。	0b
FRx_Sync_Tx	[3]	当 enabled, FIFO-Rx 等待 FIFO-Tx 在向目标转换数据之前找回远模块的输出数据。	0b

FRx-Reset	[2]	停止当前的 FIFO-Rx 转换器，复位 FSM 和所有寄存器。	0b
FRx_ERROR_En	[1]	当不允许通过 FRx_Ctrl[4]或[5]访问 FIFO-Rx 而主机试图进行访问时，通过 HRESP 端口可以进行错位提示。	0b
FRx_Start	[0]	FIFO-Rx 转换器开始位。当内部 FSM 开始转换数据时复位为 0。	

## 2. FIFO-RX 信息长度寄存器

寄存器	地址	读/写	描述	复位值
FRx_Mlen	0x7D40_0004	读/写	FIFO-Rx 信息长度寄存器	0x0000_0000

FRx_Mlen	位	描述	初始状态
FRx_Mlen	[31: 0]	当设置 FRx_Ctrl 寄存器的 FRx_Reset 位设置时，将复位值重设。	0x0000_0000

## 3. FIFO\_RX 模块尺寸寄存器

寄存器	地址	读/写	描述	复位值
FRx_Blksz	0x7D40_0008	读/写	FIFO-Rx 保密数据模块尺寸寄存器	0x0000_0000

FRx_Blksz	位	描述	初始状态
Reserved	[31: 18]	保留	0x0000
LastValidByte	[17: 16]	向 SHA-1/PRNG 模块转换的最后字的有效字节。 SHA 文本字节的末尾 00: 第一字节      01: 第二字节 10 第三字节      11: 第四字节 当 FRx_Ctrl 寄存器的 FRx-Reset 被设置时，从新设置复位值。	00b
Blksz	[15:0]	目标模块的模块尺寸。FIFO-Rx 转换 FRx_Blksz 字，目标模块会被触发，并开始运行。通过 FRx_Ctrl 内的 FRx_Dest_Module 选择目标模块。 当 FRx_Ctrl 寄存器的 FRx-Reset 被设置时，从新设置复	0x0000

		位值。	
--	--	-----	--

#### 4. FIFO\_Rx 目标地址寄存器

寄存器	地址	读/写	描述	复位值
FRx_DestAddr	0x7D40_000C	读/写	FIFO-Rx 输入缓冲地址寄存器	0x0000_0000

SKEY_IDx	位	描述	初始状态
DestAddr	[31: 0]	目标输入缓冲的内部存储器地址。FIFO_Rx 可以向此地址转换数据。当设置 FRx_Ctrl 寄存器的 FRX_Reset 位设置时，将复位值重设。	0x0000_0000

#### 5. FIFO\_Rx 信息长度计数器

寄存器	地址	读/写	描述	复位值
FRx_MlenCnt	0x7D40_0010	读/写	FIFO-Rx 信息长度计数寄存器	0x0000_0000

FRx_MlenCnt	位	描述	初始状态
FRx_MlenCnt	[31: 0]	剩下用于转换的字的数目。	0x0000_0000

#### 6. FIFO\_Rx 写缓冲区

寄存器	地址	读/写	描述	复位值
FRx_WrBuf	0x7D40_0040 ~ 0x7D40_007C	写	FIFO-Rx 写缓冲区（32 字） 注：这个地址是 CPU 访问地址	0x0000_0000
	0x7D90_0040 ~ 0x7D90_007C		FIFO-Rx 写缓冲区（32 字） 注：这个地址是 SDMA1 的访问地址。可以在此地址用 SDMA1 进行存储器到 FRx_WrBuf 的数据转换。	

FRx_WrBuf	位	描述	初始状态
FRx_WrBuf	[31: 0]	FIFO-Rx 写缓冲区（32x32 位）	0x0000_0000

注：向 FRx\_WrBuf 进行的写访问操作使 FIFO\_Rx 忽略已给的地址向 FIFO 存储器内写入数据。这就意味着在 0x0040 和 0x0080 之间的任何地址将会触发 FIFO 存储器的读操作。这个性能用突发写操作向 FIFO-Tx 产生程序。

# 13.5 安全子系统 TX FIFO 模块

## 1. FIFO-TX 控制寄存器

寄存器	地址	读/写	描述	复位值
FTx_Ctrl	0x7D80_0000	读/写	FIFO-Tx 控制/状态寄存器（只可以读取 16 位 数据）	0x0400_0000

Ftx_Ctrl	位	描述	初始状态
WrPrivError	[31]	如果写访问 FIFO-Tx 导致特权错误，此位设置为 1。（主机和命令不允许访问 FIFO-Tx）	0b
RdPrivError	[30]	如果读访问 FIFO-Tx 导致特权错误，此位设置为 1。（主机和命令不允许访问 FIFO-Tx）	0b
Reserved	[29:28]	保留	0b
Full	[27]	如果 FIFO-Tx 缓冲区满时设置此位为 1	0b
Empty	[26]	如果 FIFO-Tx 缓冲区空时设置此位为 1	0b
Done	[25]	如果 FIFO-Tx 完成向目标的数据 FTx_Mlen 字节的转换，设置此位为 1	0b
Running	[24]	如果 FIFO-Tx 完成目标的数据转换设置此位为 1，或者准备好目标的输入缓冲等待时，设置此位为 1。当 FTx_Start 位复位为 0 时，设置此位为 1。	0b
Wd2 Read	[23: 16]	可以从 FIFO 存储器内读取的字节数（FTx_RdBuf）	0x00
Wd2 Write	[15: 8]	可以写入 FIFO 存储器的字节数（FTx_RdBuf）	0x00
Src_Module	[7: 6]	选择目标模块 (00: AES, 01:DES/3DES, 10:SHA-1/PRNG, 11:保留)	0b

Host_Rd_En	[5]	主机可以从 FTx_Ctrl[31:16]和 FTx_MlenCnt 内读取数据。	0b
Host_Wr_En	[4]	主机可以 FTx_WrBuf 内写入数据。	0b
Reset	[2]	停止当前的 FIFO-Tx 转换器，复位 FSM 和所有寄存器。	0b
ERROR_En	[1]	当不允许通过 FTx_Ctrl[4]或[5]访问 FIFO-Tx 而主机试图进行访问时，通过 HRESP 端口可以进行错位提示。	0b
Start	[0]	FIFO-Tx 转换器开始位。当内部 FSM 开始转换数据时复位为 0。	

## 2. FIFO-TX 信息长度寄存器

寄存器	地址	读/写	描述	复位值
FTx_Mlen	0x7D80_0004	读/写	FIFO-Tx 信息长度寄存器	0x0000_0000

FTx_Mlen	位	描述	初始状态
FTx_Mlen	[31: 0]	当设置 FTx_Ctrl 寄存器的 FTx_Reset 位设置时，将复位值重设。	0x0000_0000

## 3. FIFO\_TX 模块尺寸寄存器

寄存器	地址	读/写	描述	复位值
FTx_BlkSz	0x7D80_0008	读/写	FIFO-Tx 保密数据模块尺寸寄存器	0x0000_0000

FTx_BlkSz	位	描述	初始状态
BlkSz	[31:0]	目标模块的模块尺寸。FIFO-Tx 转换 FTx_BlkSz 字，目标模块会被触发，并开始运行。通过 FTx_Ctrl 内的 FTx_Dest_Module 选择目标模块。  当 FTx_Ctrl 寄存器的 FTx-Reset 被设置时，从新设置复位值。	0x0000

## 4. FIFO\_Tx 源地址寄存器

寄存器	地址	读/写	描述	复位值
-----	----	-----	----	-----

FTx_SrcAddr	0x7D80_000C	读/写	FIFO-Tx 输出缓冲地址寄存器	0x0000_0000
-------------	-------------	-----	-------------------	-------------

FTx_SrcAddr	位	描述	初始状态
SrcAddr	[31: 0]	源输出缓冲的内部存储器地址。FIFO-Tx 可以向此地址转换数据。当设置 FTx_Ctrl 寄存器的 FTX_Reset 位设置时，将复位值重设。	0x0000_0000

## 5. FIFO-Tx 信息长度计数器

寄存器	地址	读/写	描述	复位值
FTx_MlenCnt	0x7D40_0010	读/写	FIFO-Tx 信息长度计数寄存器	0x0000_0000

FTx_MlenCnt	位	描述	初始状态
MlenCnt	[31: 0]	剩下用于转换的字的数目。	0x0000_0000

## 6. FIFO-Tx 写缓冲区

寄存器	地址	读/写	描述	复位值
FTx_WrBuf	0x7D80_0040 ~ 0x7D80_007C	写	FIFO-Tx 写缓冲区（32 字） 注：这个地址是 CPU 访问地址	0x0000_0000
	0x7DA0_0040 ~ 0x7DA0_007C		FIFO-Tx 写缓冲区（32 字） 注：这个地址是 SDMA1 的访问地址。可以在此地址用 SDMA1 进行存储器到 FTx_WrBuf 的数据转换。	

FTx_RdBuf	位	描述	初始状态
RdBuf	[31: 0]	FIFO-Tx 读缓冲区（32x32 位）	0x0000_0000

注：读访问 FTx\_WrBuf 时，FIFO-Tx 根据已给的地址从 FIFO 存储器内读取数据。这就意味着在 0x0040 和 0x007c 之间的任何地址将会触发 FIFO 存储器的读操作。这个性能用突发读操作向 FIFO-Tx 产生程序。

# 13.6 安全子系统 AES 模块

## 1. AES\_CTRL

寄存器	地址	读/写	描述	复位值
AES_Rx_CTRL	0x7D10_0000	读/写	AES 控制/状态寄存器	0x0000_0200

SKEY_IDx	位	描述	初始状态
WrPrivMismatch	[31]	SFR 写访问权限不匹配状态位。如果此位设置为 ‘1 ‘，会发生 SFR 写访问权限不匹配现象。	0b
RdPrivMismatch	[30]	SFR 读访问权限不匹配状态位。如果此位设置为 ‘1 ‘，会发生 SFR 读访问权限不匹配现象。	0b
Reserved	[29;11]	保留	0x000000
AesOutReady	[10]	如果设置为 ‘1 ‘，AES 输出缓冲为满置状态, 允许 ARM 或 Rx FIFO 读取当前 128 位结果数据。	0b
AesInReady	[9]	如果设置为 ‘1 ‘，AES 输出缓冲为空状态, 允许 ARM 或 Rx FIFO 读取下一个 128 位结果数据。	1b
AesContDecOn	[8]	继续解密启用位： 0：改变解密密钥 1：不改变解密密钥	0b
CtrWidth	[7： 6]	计数器模式计数器宽度位  00： 16 位计数器          01： 32 位计数器  10： 64 位计数器          11： 保留	00b
AesOpMode	[5： 4]	AES 运行模式选择位  00=保留                  01=ECB 模式  10=CBC 模式          11=CTR 模式	00b
AesOpDirection	[3]	AES 操作方向选择位  00： 加密                  01=解密	0b
AesKeyMode	[2： 1]	AES 密钥模式选择位	0b



		00: 128 位                      01: 192 位式 10: 256 位                      11: 保留	
AesOpEnable	[0]	如果设置 ‘1 ‘，AES 开始通过 ARM 运行。如果产生 AES_OP_DONE，此位变成 ‘0 ‘	0b

## 2. AES\_RX\_DIN\_01~ AES\_RX\_DIN\_04

寄存器	地址	读/写	描述	复位值
AES_Rx_DIN_01	0x7D10_0010	读/写	AES 数据输入寄存器 01	0x0000_0000
AES_Rx_DIN_02	0x7D10_0014	读/写	AES 数据输入寄存器 02	0x0000_0000
AES_Rx_DIN_03	0x7D10_0018	读/写	AES 数据输入寄存器 03	0x0000_0000
AES_Rx_DIN_04	0x7D10_001C	读/写	AES 数据输入寄存器 04	0x0000_0000

AES_Rx_DIN	位	描述	初始状态
AESDIN	[31:0]	AES 第一个到第四个 32 位数据输入寄存器	0x0000_0000

## 3. AES\_RX\_DOUT\_01~ AES\_RX\_DOUT\_04/ AES\_TX\_DOUT\_01~ AES\_TX\_DOUT\_04

寄存器	地址	读/写	描述	复位值
AES_RX_DOUT_01	0x7D10_0020	读/写	AES 数据输出寄存器 01	0x0000_0000
AES_RX_DOUT_02	0x7D10_0024	读/写	AES 数据输出寄存器 02	0x0000_0000
AES_RX_DOUT_03	0x7D10_0028	读/写	AES 数据输出寄存器 03	0x0000_0000
AES_RX_DOUT_04	0x7D10_002C	读/写	AES 数据输出寄存器 04	0x0000_0000
AES_TX_DOUT_01	0x7D10_0020	读/写	AES 数据输出寄存器 01	0x0000_0000
AES_TX_DOUT_02	0x7D10_0024	读/写	AES 数据输出寄存器 02	0x0000_0000
AES_TX_DOUT_03	0x7D10_0028	读/写	AES 数据输出寄存器 03	0x0000_0000
AES_TX_DOUT_04	0x7D10_002C	读/写	AES 数据输出寄存器 04	0x0000_0000

AES_Tx_DOUT	位	描述	初始状态
AESDout	[31:0]	AES 第一个到第四个 32 位数据输出寄存器	0x0000_0000

4. AES\_RX\_KEY\_01~ AES\_RX\_KEY\_08

寄存器	地址	读/写	描述	复位值
AES_RX_KEY_01	0x7D10_0080	读/写	AES 密钥输入寄存器 01	0x0000_0000
AES_RX_KEY_02	0x7D10_0084	读/写	AES 密钥输入寄存器 02	0x0000_0000
AES_RX_KEY_03	0x7D10_0028	读/写	AES 密钥输入寄存器 03	0x0000_0000
AES_RX_KEY_04	0x7D10_008C	读/写	AES 密钥输入寄存器 04	0x0000_0000
AES_TX_KEY_05	0x7D10_0090	读/写	AES 密钥输入寄存器 05	0x0000_0000
AES_TX_KEY_06	0x7D10_0094	读/写	AES 密钥输入寄存器 06	0x0000_0000
AES_TX_KEY_07	0x7D10_0098	读/写	AES 密钥输入寄存器 07	0x0000_0000
AES_TX_KEY_08	0x7D10_009C	读/写	AES 密钥输入寄存器 08	0x0000_0000

AES_Rx_KEY	位	描述	初始状态
AESKey	[31:0]	AES 第一个到第八个 32 位密钥输入寄存器	0x0000_0000

5. AES\_RX\_IV\_01~ AES\_RX\_IV\_04

寄存器	地址	读/写	描述	复位值
AES_Rx_IV_01	0x7D10_00A0	读/写	AES IV 输入寄存器 01	0x0000_0000
AES_Rx_IV_02	0x7D10_00A4	读/写	AES IV 输入寄存器 02	0x0000_0000
AES_Rx_IV_03	0x7D10_00A8	读/写	AES IV 输入寄存器 03	0x0000_0000
AES_Rx_IV_04	0x7D10_00AC	读/写	AES IV 输入寄存器 04	0x0000_0000

AES_Rx_IV	位	描述	初始状态
AESIV	[31:0]	AES 第一个到第四个 32 位 IV 输入寄存器	0x0000_0000

6. AES\_RX\_CTR\_01~ AES\_RX\_CTR\_04

寄存器	地址	读/写	描述	复位值
AES_Rx_CTR_01	0x7D10_00B0	读/写	AES 计数器预先输入寄存器 01	0x0000_0000
AES_Rx_CTR_02	0x7D10_00B4	读/写	AES 计数器预先输入寄存器 02	0x0000_0000

AES_Rx_CTRL_03	0x7D10_00B8	读/写	AES 计数器预先输入寄存器 03	0x0000_0000
AES_Rx_CTRL_04	0x7D10_00BC	读/写	AES 计数器预先 V 输入寄存器 04	0x0000_0000

AES_Rx_CTRL	位	描述	初始状态
AECtr	[31:0]	AES 第一个到第四个 32 位计数器预先输入寄存器	0x0000_0000

## 13.7 安全子系统 DES/CDES 模块

### 1. TDES\_RX\_CTRL

寄存器	地址	读/写	描述	复位值
TDES_Rx_CTRL	0x7D20_0000	读/写	TDES 控制/状态寄存器	0x0000_0200

TDES_Rx_CTRL	位	描述	初始状态
WrPrivMismatch	[31]	SFR 写访问权限不匹配状态位。如果此位设置为 ‘1’，会发生 SFR 写访问权限不匹配现象。	0b
RdPrivMismatch	[30]	SFR 读访问权限不匹配状态位。如果此位设置为 ‘1’，会发生 SFR 读访问权限不匹配现象。	0b
Reserved	[29;8]	保留	0x0000_00
TdesOutReady	[7]	如果设置为 ‘1’，TDES 输出缓冲为满置状态, 允许 ARM 或 Rx FIFO 读取当前 128 位结果数据。	0b
TdesInReady	[6]	如果设置为 ‘1’，TDES 输出缓冲为空状态, 允许 ARM 或 Rx FIFO 读取下一个 128 位结果数据。	1b
DesOrTdes	[5]	DES 或者 TDES 运行选择位 0: DES 方式      1: TDES 方式	0b
TdesMode	[4: 3]	AES 模式选择位 01=ECB 模式      10=CBC 模式	00b
TdesOpDirection	[2]	AES 操作方向选择位 00: 加密              01=解密	0b

TdesIntMode	[1]	TDES 操作结束模式选择位 00: 轮询方式                      01=中断方式	0b
TdesOpEnable	[0]	如果设置 ‘1 ‘，TDES 开始通过 ARM 运行。如果产生 DES_ 或者 TDES_OP_DONE，此位变成 ‘0 ‘	0b

## 2. TDES\_RX\_KEY1\_0

寄存器	地址	读/写	描述	复位值
TDES_RX_KEY1_0	0x7D20_0010	读/写	TDES 密钥输入寄存器 1_0	0x0000_0000
TDES_RX_KEY1_1	0x7D20_0014	读/写	TDES 密钥输入寄存器 1_1	0x0000_0000
TDES_RX_KEY2_0	0x7D10_0018	读/写	TDES 密钥输入寄存器 2_0	0x0000_0000
TDES_RX_KEY2_1	0x7D10_001C	读/写	TDES 密钥输入寄存器 2_1	0x0000_0000
TDES_RX_KEY3_0	0x7D10_0020	读/写	TDES 密钥输入寄存器 3_0	0x0000_0000
TDES_RX_KEY3_1	0x7D10_0024	读/写	TDES 密钥输入寄存器 3_1	0x0000_0000

TDES_Rx_KEY	位	描述	初始状态
TDESKey	[31:0]	AES 第一个到第六个 32 位密钥输入寄存器	0x0000_0000

## 3. TDES\_RX\_INPUT\_0/ TDES\_RX\_INPUT\_1

寄存器	地址	读/写	描述	复位值
TDES_RX_INPUT_0	0x7D20_0040	读/写	TDES 数据输入寄存器 0	0x0000_0000
TDES_RX_INPUT_1	0x7D20_0044	读/写	TDES 数据输入寄存器 1	0x0000_0000

TDES_Rx_INPUT	位	描述	初始状态
TDESdinx	[31:0]	AES 第一个到第二个 32 位数据输入寄存器	0x0000_0000

## 4. TDES\_RX\_OUTPUT\_0/ TDES\_TX\_OUTPUT\_0/ TDES\_RX\_INPUT\_1 TDES\_TX\_INPUT\_1

寄存器	地址	读/写	描述	复位值
TDES_RX_OUTPUT_0	0x7D20_0048	读/写	TDES 数据输出寄存器 0	0x0000_0000

TDES_RX_OUTPUT_1	0x7D20_004C	读/写	TDES 数据输出寄存器 1	0x0000_0000
TDES_TX_OUTPUT_0	0x7D60_0048	读/写	TDES 数据输出寄存器 0	0x0000_0000
TDES_TX_OUTPUT_1	0x7D60_004C	读/写	TDES 数据输出寄存器 1	0x0000_0000

TDES_Rx_OUTPUT	位	描述	初始状态
TDESdout	[31:0]	AES 第一个到第二个 32 位数据输出寄存器	0x0000_0000

### 5. TDES\_RX\_IV\_0~ TDES\_RX\_IV\_1

寄存器	地址	读/写	描述	复位值
TDES_Rx_IV_0	0x7D20_0050	读/写	TDES IV 输入寄存器 0	0x0000_0000
TDES_Rx_IV_1	0x7D20_0054	读/写	TDES IV 输入寄存器 1	0x0000_0000

TDES_Rx_IV	位	描述	初始状态
TdesIv	[31:0]	TDES 第一个到第二个 32 位 IV 输入寄存器	0x0000_0000

## 13.8 安全子系统 SHA-1/PRNG 模块

### 1. HASH\_CONTROL

寄存器	地址	读/写	描述	复位值
HASH_CONTROL	0x7D30_0000	读/写	HASH 引擎控制寄存器	0x0000_0000

HASH_CONTROL	位	描述	初始状态
Reserved	[31:9]	保留	0x0000_00
USE_IV	[8]	使用任意 IV 替代 SHA-1 常数 0: 常数            1:任意 IV	0b
Edn_of_Hash_byte	[7:6]	SHA 文本字节的末端 00: 第一字节            01: 第二字节 10: 第三字节            04: 第四字节	00b

SEED_SETTING_ENABLE	[5]	种子设置使能	0b
Hash_input_finished	[4]	结束 Hash 输入（由硬件清除）	0b
Hash_start	[3]	开始 Hash	0b
Date_Selection	[2]	指明寄存器下一个数值是密钥还是文本 0: 文本                      1: 密钥	0b
Engine_Selection	[1:0]	选择使用 SHA-1 或 HMAC 或 PRNG 00:HMAC                      01:SHA1 10:PRNG                      11=保留	00b

## 2. HASH-DATA

寄存器	地址	读/写	描述	复位值
HASH_DATA	0x7D30_0004	读/写	HASH 数据	0x0000_0000

HASH_DATA	位	描述	初始状态
HASH_DATA	[31:0]	Hash 数据输入寄存器	0x0000_0000

## 3. SEED\_DATA-01~SEED\_DATA\_10

寄存器	地址	读/写	描述	复位值
SEED_DATA_01	0x7D30_0008	读/写	PRNG 种子数据 1（[31:0]）	0x0000_0000
SEED_DATA_02	0x7D30_000C	读/写	PRNG 种子数据 2（[63:32]）	0x0000_0000
SEED_DATA_03	0x7D30_0010	读/写	PRNG 种子数据 3（[95:64]）	0x0000_0000
SEED_DATA_04	0x7D30_0014	读/写	PRNG 种子数据 4（[127:96]）	0x0000_0000
SEED_DATA_05	0x7D30_0018	读/写	PRNG 种子数据 5（[159:128]）	0x0000_0000
SEED_DATA_06	0x7D30_001C	读/写	PRNG 种子数据 6（[191:160]）	0x0000_0000
SEED_DATA_07	0x7D30_0020	读/写	PRNG 种子数据 7（[223:192]）	0x0000_0000
SEED_DATA_08	0x7D30_0024	读/写	PRNG 种子数据 8（[255:224]）	0x0000_0000
SEED_DATA_09	0x7D30_0028	读/写	PRNG 种子数据 9（[287:256]）	0x0000_0000
SEED_DATA_10	0x7D30_002C	读/写	PRNG 种子数据 10（[319:288]）	0x0000_0000

SEED_DATA	位	描述	初始状态
SEED_DATA	[31:0]	PRNG 种子数据	0x0000_0000

#### 4. HASH\_STATUS

寄存器	地址	读/写	描述	复位值
HASH_STATUS	0x7D30_0030	读	HASH 状态	0x0000_0010

HASH_STATUS	位	描述	初始状态
Reserved	[31:5]	保留	0x0000_00
BUFFER_IN_ENABLE	[4]	1:缓冲区输入使能（缓冲区为空） 0: 缓冲区不能输入数据（缓冲区已满）	1b
HASH_engine_ready	[3]	已准备接收输入数据的下一个 64 字节	0b
Random_Nunber_Ready	[2]	随机数字	0b
32bit_ready	[1]	只在测试时使用 准备运行下一个 32 位	0b
HASH_output_ready	[0]	已经完成 160 位的 Hash 计算，准备应用	0b

#### 5. HASH\_OUTPUT\_01 (PRNG\_OUTPUT\_01\_)~HASH\_OUTPUT\_10 (PRNG\_OUTPUT\_10)

寄存器	地址	读/写	描述	复位值
HASH_OUTPUT_01	0x7D30_0034	读	HASH 输出（01）或 PRNG 输出[31:0]	0x0000_0000
HASH_OUTPUT_02	0x7D30_0038	读	HASH 输出（02）或 PRNG 输出[63:32]	0x0000_0000
HASH_OUTPUT_03	0x7D30_003C	读	HASH 输出（03）或 PRNG 输出[95: 64]	0x0000_0000
HASH_OUTPUT_04	0x7D30_0040	读	HASH 输出（04）或 PRNG 输出[127: 96]	0x0000_0000
HASH_OUTPUT_05	0x7D30_0044	读	HASH 输出（05）或 PRNG 输出[159: 128]	0x0000_0000
HASH_OUTPUT_06	0x7D30_0048	读	HASH 输出（06）或 PRNG 输出[191: 160]	0x0000_0000
HASH_OUTPUT_07	0x7D30_004C	读	HASH 输出（07）或 PRNG 输出[223: 192]	0x0000_0000
HASH_OUTPUT_08	0x7D30_0050	读	HASH 输出（08）或 PRNG 输出[255: 224]	0x0000_0000
HASH_OUTPUT_09	0x7D30_0054	读	HASH 输出（09）或 PRNG 输出[287: 256]	0x0000_0000

HASH_OUTPUT_10	0x7D30_0058	读	HASH 输出 (10) 或 PRNG 输出 [319: 288]	0x0000_0000
HASH_OUTPUT_01	0x7D70_0034	读	HASH 输出 (01) 或 PRNG 输出 [31:0]	0x0000_0000
HASH_OUTPUT_02	0x7D70_0038	读	HASH 输出 (02) 或 PRNG 输出 [63:32]	0x0000_0000
HASH_OUTPUT_03	0x7D70_003C	读	HASH 输出 (03) 或 PRNG 输出 [95: 64]	0x0000_0000
HASH_OUTPUT_04	0x7D70_0040	读	HASH 输出 (04) 或 PRNG 输出 [127: 96]	0x0000_0000
HASH_OUTPUT_05	0x7D70_0044	读	HASH 输出 (05) 或 PRNG 输出 [159: 128]	0x0000_0000
HASH_OUTPUT_06	0x7D70_0048	读	HASH 输出 (06) 或 PRNG 输出 [191: 160]	0x0000_0000
HASH_OUTPUT_07	0x7D70_004C	读	HASH 输出 (07) 或 PRNG 输出 [223: 192]	0x0000_0000
HASH_OUTPUT_08	0x7D70_0050	读	HASH 输出 (08) 或 PRNG 输出 [255: 224]	0x0000_0000
HASH_OUTPUT_09	0x7D70_0054	读	HASH 输出 (09) 或 PRNG 输出 [287: 256]	0x0000_0000
HASH_OUTPUT_10	0x7D70_0058	读	HASH 输出 (10) 或 PRNG 输出 [319: 288]	0x0000_0000

HASH_OUTPUT	位	描述	初始状态
HASH_OUTPUT	[31:0]	DANG Engine_selection[1:0]==2' b10, PRNG_output, 或者 Rx/Tx 为 Hash_output[31:0]	0x0000_0000

## 6. HASH\_MIDOUT\_01~HASH\_MIDOUT\_05

寄存器	地址	读/写	描述	复位值
HASH_MIDOUT_01	0x7D30_005C	读	HASH_MIDOUT [159: 128]	0x0000_0000
HASH_MIDOUT_02	0x7D30_0060	读	HASH_MIDOUT [127: 96]	0x0000_0000
HASH_MIDOUT_03	0x7D30_0064	读	HASH_MIDOUT [95: 64]	0x0000_0000
HASH_MIDOUT_04	0x7D30_0068	读	HASH_MIDOUT [63:32]	0x0000_0000
HASH_MIDOUT_05	0x7D30_006C	读	HASH_MIDOUT [31:0]	0x0000_0000
HASH_MIDOUT_01	0x7D70_005C	读	HASH_MIDOUT [159: 128]	0x0000_0000
HASH_MIDOUT_02	0x7D70_0060	读	HASH_MIDOUT [127: 96]	0x0000_0000
HASH_MIDOUT_03	0x7D70_0064	读	HASH_MIDOUT [95: 64]	0x0000_0000
HASH_MIDOUT_04	0x7D70_0068	读	HASH_MIDOUT [63:32]	0x0000_0000
HASH_MIDOUT_05	0x7D70_006C	读	HASH_MIDOUT [31:0]	0x0000_0000



HASH_MIDOUT	位	描述	初始状态
HASH_MIDOUT	[31:0]	HASH_MIDOUT	0x0000_0000

## 7. HASH\_IV\_01~HASH\_IV\_05

寄存器	地址	读/写	描述	复位值
HASH_IV_01	0x7D30_0070	读	HASH_IV [159: 128]	0x0000_0000
HASH_IV_02	0x7D30_0074	读	HASH_IV [127: 96]	0x0000_0000
HASH_IV_03	0x7D30_0078	读	HASH_IV [95: 64]	0x0000_0000
HASH_IV_04	0x7D30_007C	读	HASH_IV [63:32]	0x0000_0000
HASH_IV_05	0x7D30_0080	读	HASH_IV [31:0]	0x0000_0000
HASH_IV_01	0x7D70_0070	读	HASH_IV [159: 128]	0x0000_0000
HASH_IV_02	0x7D70_0074	读	HASH_IV [127: 96]	0x0000_0000
HASH_IV_03	0x7D70_0078	读	HASH_IV [95: 64]	0x0000_0000
HASH_IV_04	0x7D70_007C	读	HASH_IV [63:32]	0x0000_0000
HASH_IV_05	0x7D70_0080	读	HASH_IV [31:0]	0x0000_0000

HASH_IV	位	描述	初始状态
HASH_IV	[31:0]	HASH_IV	0x0000_0000

## 8. PRE\_MSG\_LENGTH\_01/PRE\_MSG\_LENGTH\_02

寄存器	地址	读/写	描述	复位值
PRE_MSG_LENGTH_01	0x7D70_0084	读/写	PRE_MSG_LENGTH[63:32]	0x0000_0000
PRE_MSG_LENGTH_02	0x7D70_0088	读/写	PRE_MSG_LENGTH[31:0]	0x0000_0000

PRE_MSG_LENGTH	位	描述	初始状态
PRE_MSG_LENGTH	[31:0]	PRE_MSG_LENGTH	0x0000_0000

## 14 显示控制器

### 14.1 概述

显示控制器有一个用于转换图像数据的逻辑，这个逻辑是指从本地总线的后处理器或系统内存内的视频缓冲区到外部 LCD 驱动器接口的传输图像数据的逻辑。LCD 驱动接口有四种接口，如传统的 RGB 接口，I80 接口，NTSC/PAL 标准 TV 编码器接口和 IT-R BT. 601 接口。显示控制器支持 5 层图像窗口。覆盖图像窗口支持多种颜色格式、16 级 alpha 混合、color key，纵横坐标位置控制，软滚动，可变窗口尺寸等等。

显示控制器支持各种颜色格式，如 RGB，YcbCr4: 4: 4。

显示控制器的屏幕可以支持不同环境，环境与水平或垂直像素值，数据接口的数据行宽度，接口时序和刷新频率等等有关。

显示控制器用来转换视频数据，并产生需要的控制信号，如，RGB\_VSYNC，RGB\_HSYNC，RGB\_VCLK，RGB\_VDEN 和 SYS-CS0（作为控制信号）。显示控制器有视频数据端口，这些数据端口是 RGB\_VD[23:0]和 SYS\_VD[17:0]，TV\_OUT 如图 14-1 所示。。

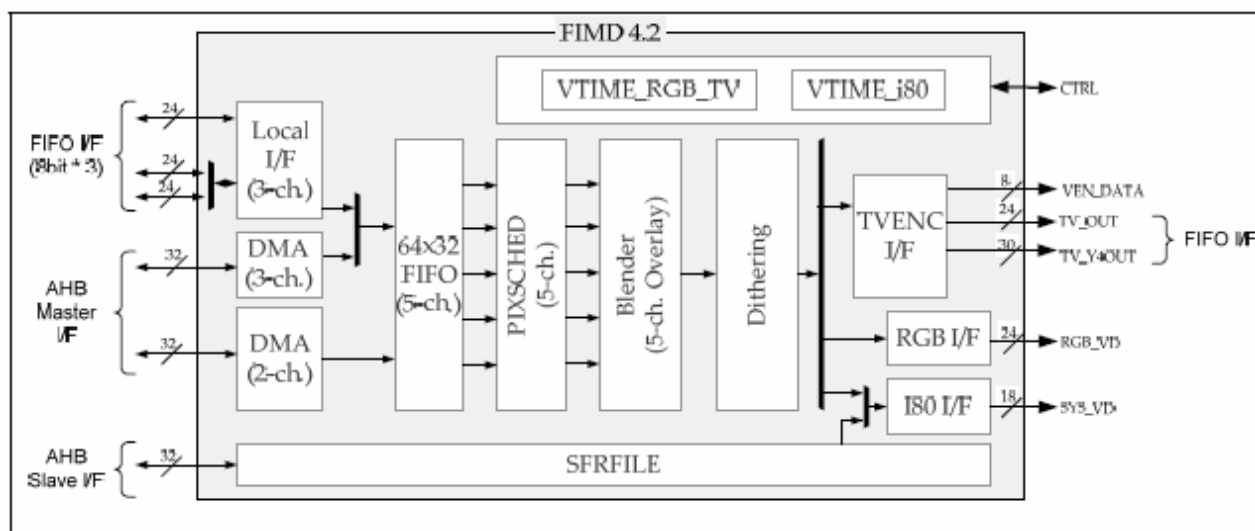


图 14-1 显示控制器顶层模块图

## 14.2 功能描述

### 14.2.1 .子模块简要

显示控制器由 VSFR,VDMA,VPRCS,VTIME 和视频时钟产生器组成。VSFR 包括可编程寄存器设置和 2 个 256x25 的调色板存储器，VSFR 用于配置显示控制器。VDMA 专用于显示 DMA，VDMA 可以将帧存储器内的视频数据转换到 VPRCS。通过使用特殊的 DMA-VDMA，可以不使用 CPU，直接将视频数据显示在屏幕上。VPRCS 接收 VDMA 发出的视频数据，将其转换到适合的数据格式，如 8 位像素或 16 位像素后，将视频数据发送到显示设备上。VTIME 包括可编程逻辑，以支持在不同 LCD 驱动上发现的接口时序和速率的各种环境。VIIME 模块产生 RGB\_VSYNC, RGB\_HSYNC, RGB\_VCLK, RBG\_VDEN, SYS\_CS, SYS\_CS0 等等。

### 14.2.2. 数据流

FIFO 中存在 VDMA。当 FIFO 全部空或部分为空时，VDMA 从具有突发内存传输模式的帧存储器内请求数据。当存储控制器内的总线仲裁允许这种形式的转换请求时，将会有 4/8/16 连续数据从系统内转换到内部 FIFO 中。FIFO 的尺寸大小为 64 个字。FIFO 的大小由数据转换速率决定。显示控制器有 5 个 FIFO，因为显示控制器需要支持覆盖窗口显示模式。一个屏幕显示模式，需要一个 FIFO。通过 VPRCS 获取穿过 FIFO 的数据，VPRCS 对最后的图像数据有混合，调度的功能。VPRCS 支持覆盖功能，可以覆盖 5 个窗口的图像。图 14-2 表示出系统总线到输出缓冲区的数据流。VDMA 有 5 个 DMA 通道和 3 个本地输入 I/F 区域。为了混合运行，CSC 模块将 YcbCr 数据改变为 RGB 数据。写入 SFR 的 Alpha 值决定混合的程度。输出缓冲区的数据将会出现在视屏数据端口上。

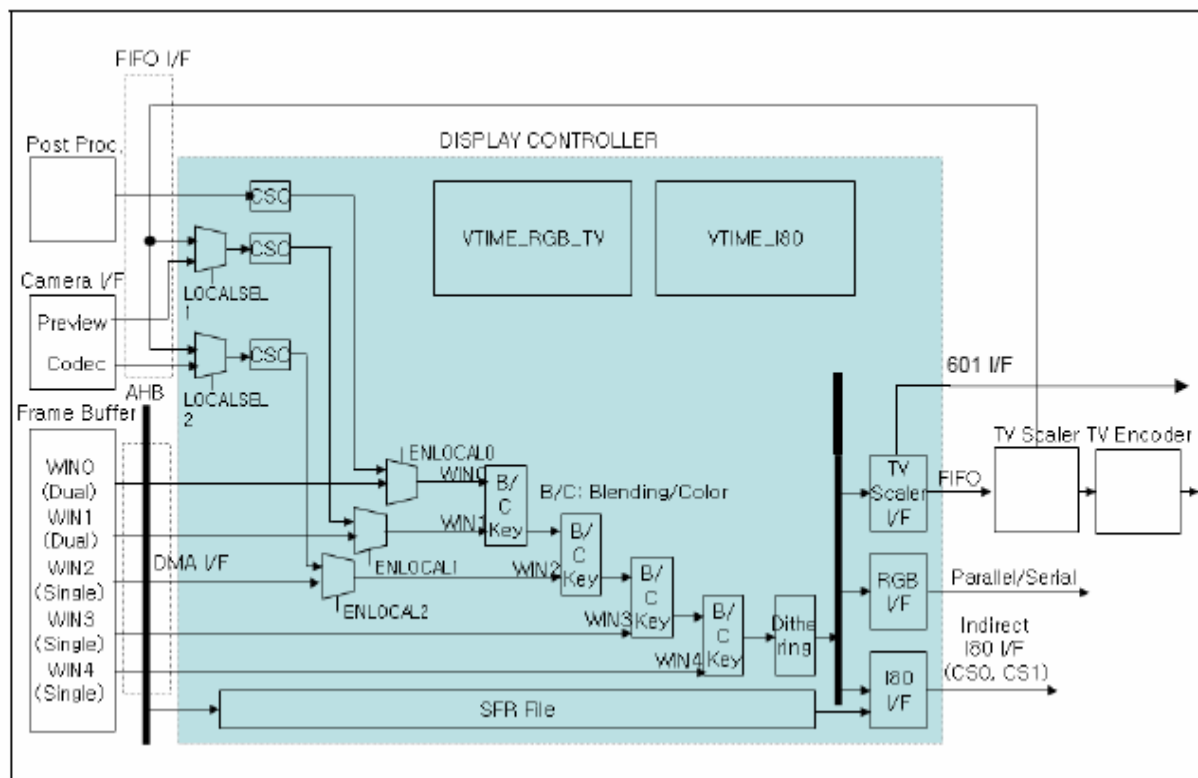


图 14-2 数据流模块图

### 14.2.3. 接口

显示控制器支持 4 个类型的显示设备。1 为传统的 RGB 接口类型，使用 RGB 数据，垂直/水平同步，数据有效信号和数据同步时钟。第二种类型是 I80 接口，使用地址，数据，芯片选择，以及读/写控制和寄存器/状态指示信号。在这种类型的 LCD 设备内有一个帧缓冲区，可以进行自动刷新。因此，显示控制器一次只能向 LCD 上传一个静态图像。第三种类型是 ITU\_R BT.601 接口，使用 YUV 数据，垂直/水平同步，可选字段信号，数据有效信号和数据同步时钟。第四种类型是有 TV 编码器的 FIFO 接口。

### 14.2.4. 颜色数据概要

#### RGB 数据格式

显示控制器需要指定帧缓冲区的存储器格式。下面的表格显示出每个显示模式的一些实例。

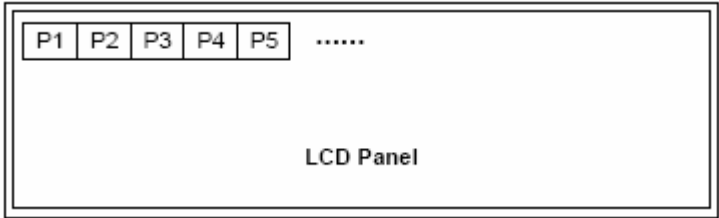
1. 28BPP 显示（A4+888）

（BSWP=0, HWSWP=0）

	D[31:24]	D[23:0]
000H	虚位	P1
004H	虚位	P2
008H	虚位	P3
...		

(BSWP=0,HWSWP=0,BLD\_PIX=1,ALPHA\_SE=1)

	D[31:28]	D[27:24]	D[23:0]
000H	虚位	Alpha 值	P1
004H	虚位	Alpha 值	P2
008H	虚位	Alpha 值	P3
...			



注：D[23:16]=红数据， D[15:8]=绿数据，D[7:0]=蓝数据

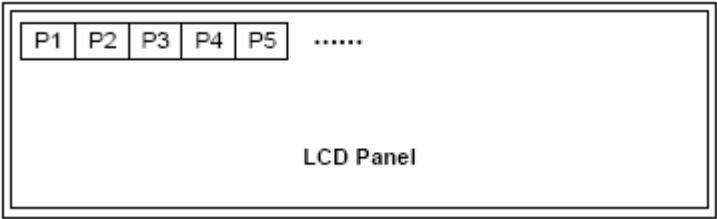
在设置 BLE\_PIX 和 ALPHA\_SEL 的情况下，D[27:24]=Alpha 值,D[23:16]=红数据，D[7:0]=蓝数据。

2. 25BPP 显示（A888）

（BSWP=0, HWSWP=0）

	D[31:25]	D[24]	D[23:0]
000H	虚位	AEN	P1
004H	虚位	AEN	P2

008H	虚位	AEN	P3
...			



注：（1）AEN:透明性值选择位

AEN=0: 使用 ALPHA0\_R/G/B 值

AEN=0:使用 ALPHA1\_R/G/B 值

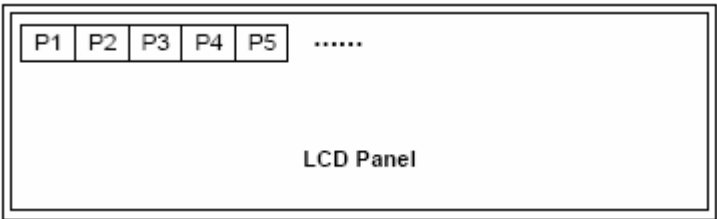
如果设置像素混合后，可以通过 AEN 选择的 alpha 值混合像素。通过 SFR 值选择的 Alpha 值可以为 ALPHA0\_R, ALPHA0\_G, ALPHA1\_R, ALPHA1\_B.

（2）D[23:16]=红数据， D[15:8]=绿数据，D[7:0]=蓝数据

### 3. 24BPP 显示（A887）

（BSWP=0, HWSWP=0）

	D[31:24]	D[23]	D[22:0]
000H	虚位	AEN	P1
004H	虚位	AEN	P2
008H	虚位	AEN	P3
...			



注：（1）AEN:透明性值选择位

AEN=0: 使用 ALPHA0\_R/G/B 值

AEN=0:使用 ALPHA1\_R/G/B 值

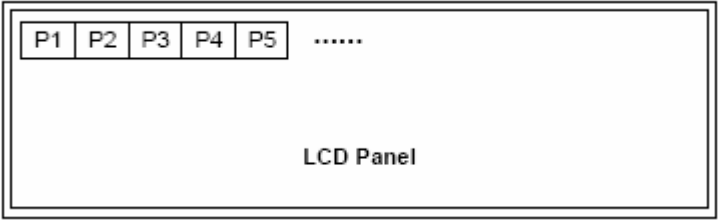
如果设置像素混合后，可以通过 AEN 选择的 alpha 值混合像素。通过 SFR 值选择的 Alpha 值可以为 ALPHA0\_R, ALPHA0\_G, ALPHA1\_R, ALPHA1\_B.

(2) D[22:15]=红数据， D[14:7]=绿数据， D[6:0]=蓝数据

4. 24BPP 显示（888）

(BSWP=0, HWSWP=0)

	D[31:24]	D[23:0]
000H	虚位	P1
004H	虚位	P2
008H	虚位	P3
...		

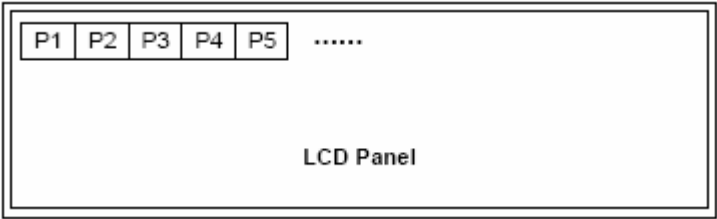


注： D[23:16]=红数据， D[15:8]=绿数据， D[7:0]=蓝数据

5. 19BPP 显示（A666）

(BSWP=0, HWSWP=0)

	D[31:19]	D[18]	D[17:0]
000H	虚位	AEN	P1
004H	虚位	AEN	P2
008H	虚位	AEN	P3
...			



注：（1）AEN:透明性值选择位

AEN=0: 使用 ALPHA0\_R/G/B 值

AEN=0: 使用 ALPHA1\_R/G/B 值

如果设置像素混合后，可以通过 AEN 选择的 alpha 值混合像素。通过 SFR 值选择的 Alpha 值可以为 ALPHA0\_R, ALPHA0\_G, ALPHA1\_R, ALPHA1\_B.

（2）D[17:12]=红数据， D[11:6]=绿数据， D[5:0]=蓝数据

6. 18BPP 显示（666）

（BSWP=0, HWSWP=0）

	D[31:18]	D[17:0]
000H	虚位	P1
004H	虚位	P2
008H	虚位	P3
...		



注： D[17:12]=红数据， D[11:6]=绿数据， D[5:0]=蓝数据



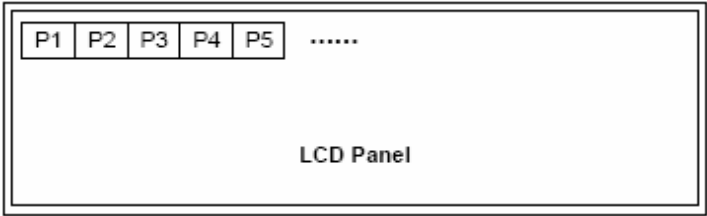
7. 16BPP 显示（A555）

（BSWP=0, HWSWP=0）

	D[31]	D[30:16]	D[15]	D[14:0]
000H	AEN1	P1	AEN2	P2
004H	AEN3	P3	AEN4	P4
008H	AEN5	P5	AEN6	P6
...				

（BSWP=0, HWSWP=1）

	D[31]	D[30:16]	D[15]	D[14:0]
000H	AEN2	P1	AEN1	P2
004H	AEN4	P3	AEN3	P4
008H	AEN6	P5	AEN5	P6
...				



注：（1）AEN:透明性值选择位

AEN=0: 使用 ALPHA0\_R/G/B 值

AEN=0:使用 ALPHA1\_R/G/B 值

如果设置像素混合后，可以通过 AEN 选择的 alpha 值混合像素。通过 SFR 值选择的 Alpha 值可以为 ALPHA0\_R, ALPHA0\_G, ALPHA1\_R, ALPHA1\_B.

（2）D[14:10]=红数据， D[9:5]=绿数据， D[4:0]=蓝数据

8. 16BPP 显示（1+555）

（BSWP=0, HWSWP=0）

	D[31:16]	D[15:0]
000H	P1	P2

004H	P3	P4
008H	P5	P6
...		

(BSWP=0, HWSWP=1)

	D[31:16]	D[15:0]
000H	P2	P1
004H	P4	P3
008H	P6	P5
...		

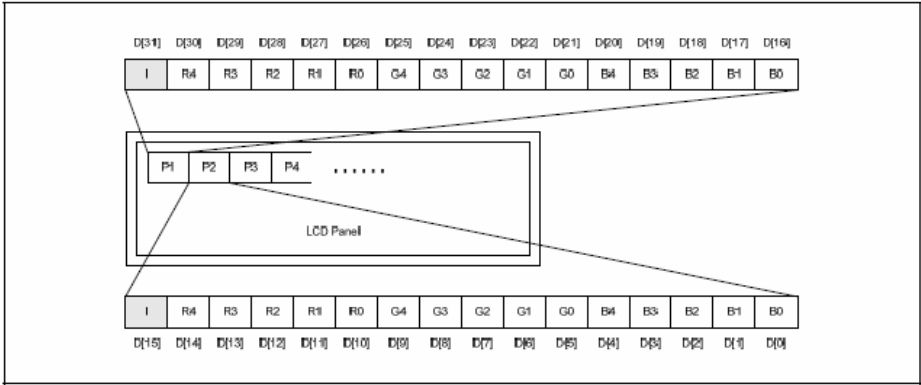


图 14-3 16BPP(1+5:5:5)显示类型

注：{D[14:10], D[15]}=红数据， {D[9:5], D[15]}=绿数据，{D[4:0], D[15]}=蓝数据

9. 16BPP 显示（565）

(BSWP=0, HWSWP=0)

	D[31:16]	D[15:0]
000H	P1	P2
004H	P3	P4
008H	P5	P6
...		

(BSWP=0, HWSWP=1)

	D[31:16]	D[15:0]
000H	P2	P1
004H	P4	P3
008H	P6	P5
...		

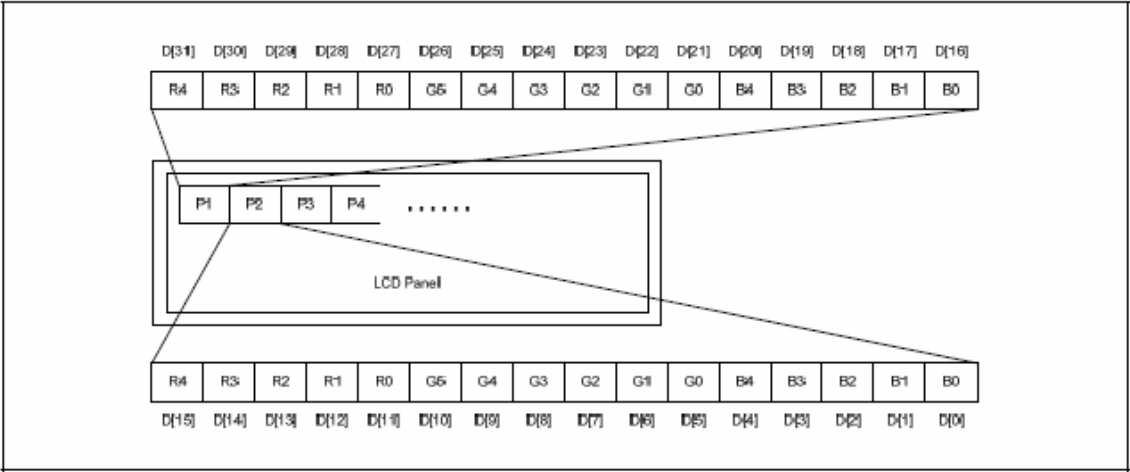


图 14-4 16BPP(5:6:5)显示类型

注：D[15:11]=红数据， D[10:5]=绿数据， D[4:0]=蓝数据

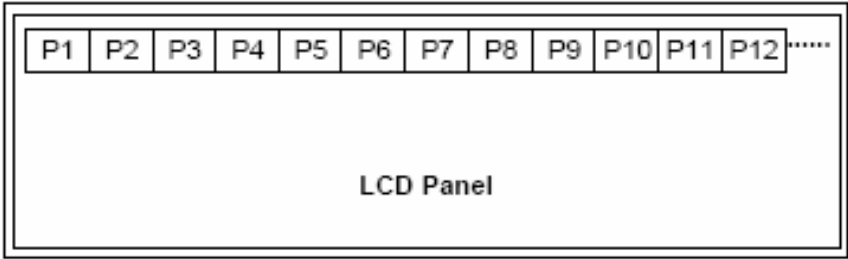
10. 8BPP 显示（调色板）

(BSWP=0, HWSWP=0)

	D[31:24]	D[23:16]	D[15:8]	D[7:0]
000H	P1	P2	P3	P4
004H	P5	P6	P7	P8
008H	P9	P10	P11	P12
...				

(BSWP=1, HWSWP=0)

	D[31:24]	D[23:16]	D[15:8]	D[7:0]
000H	P4	P3	P2	P1
004H	P8	P7	P6	P5
008H	P12	P11	P10	P9
...				



注：（1）由调色板存储器查找帧缓冲区值。

调色板存储器的 MSB 值死后 AEN 位。

AEN=0: 使用 ALPHA0\_R/G/B 值

AEN=0:使用 ALPHA1\_R/G/B 值

如果设置像素混合后，可以通过 AEN 选择的 alpha 值混合像素。通过 SFR 值选择的 Alpha 值可以为 ALPHA0\_R, ALPHA0\_G, ALPHA1\_R, ALPHA1\_B.

11. 4BPP 显示（调色板）

(BSWP=0, HWSWP=0)

	D[31:28]	D[27:24]	D[23:20]	D[19:16]	D[15:12]	D[11:8]	D[7:4]	D[3:0]
000H	P1	P2	P3	P4	P5	P6	P7	P8
004H	P9	P10	P11	P12	P13	P14	P15	P16
008H	P17	P18	P19	P20	P21	P22	P23	P24
...								

(BSWP=1, HWSWP=0)

	D[31:28]	D[27:24]	D[23:20]	D[19:16]	D[15:12]	D[11:8]	D[7:4]	D[3:0]
000H	P7	P8	P5	P6	P3	P4	P1	P2
004H	P15	P16	P13	P14	P11	P12	P9	P10
008H	P23	P24	P21	P22	P19	P20	P17	P18
...								

注：（1）由调色板存储器查找帧缓冲区值。

调色板存储器的 MSB 值死后 AEN 位。

AEN=0: 使用 ALPHA0\_R/G/B 值

AEN=0:使用 ALPHA1\_R/G/B 值

如果设置像素混合后，可以通过 AEN 选择的 alpha 值混合像素。通过 SFR 值选择的 Alpha 值可以为 ALPHA0\_R, ALPHA0\_G, ALPHA1\_R, ALPHA1\_B.

12. 2BPP 显示（调色板）

(BSWP=0, HWSWP=0)

	D[31:30]	D[29:28]	D[27:26]	D[25:24]	D[23:22]	D[21:20]	D[19:18]	D[17:16]
000H	P1	P2	P3	P4	P5	P6	P7	P8
004H	P17	P18	P19	P20	P21	P22	P23	P24
008H	P33	P34	P35	P36	P37	P38	P39	P40
...								

	D[15:14]	D[13:12]	D[11:10]	D[9:8]	D[7:6]	D[5:4]	D[3:2]	D[1:0]
000H	P9	P10	P11	P12	P13	P14	P15	P16
004H	P25	P26	P27	P28	P29	P30	P31	P32
008H	P41	P42	P43	P44	P45	P46	P47	P48
...								

注：（1）由调色板存储器查找帧缓冲区值。

调色板存储器的 MSB 值死后 AEN 位。

AEN=0: 使用 ALPHA0\_R/G/B 值

AEN=0:使用 ALPHA1\_R/G/B 值

如果设置像素混合后，可以通过 AEN 选择的 alpha 值混合像素。通过 SFR 值选择的 Alpha 值可以为 ALPHA0\_R, ALPHA0\_G, ALPHA1\_R,

ALPHA1\_B.

13. 1BPP 显示（调色板）

（BSWP=0, HWSWP=0）

	[31]	[30]	[29]	[28]	[27]	[26]	[25]	[24]	[23]	[22]	[21]	[20]	[19]	[18]	[17]	[16]
002H	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16
006H	P33	P34	P35	P36	P37	P38	P39	P40	P41	P42	P43	P44	P45	P46	P47	P48
...																
	[15]	[14]	[13]	[12]	[11]	[10]	[9]	[8]	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
000H	P17	P18	P19	P20	P21	P22	P23	P24	P25	P26	P27	P28	P29	P30	P31	P32
004H	P49	P50	P51	P52	P53	P54	P55	P56	P57	P58	P59	P60	P61	P62	P63	P64
...																

14.2.5. 调色板的使用

调色板配置和格式控制

显示控制器支持 256 种颜色的调色板，可以进行颜色映射的各种的选择。

用户可以通过四种格式从 25 位颜色中选择 256 颜色。256 颜色调色包由 256x25 位 DPSRAM 组成。调色板支持 8:8:8,6:6:6,5:6:5(R:G:B)等格式。

A:5:5:5 格式，如表 14-1 所示写调色板，将 VD 管脚与 TFT LCD 控制板相连

（R(5)=VD[23:19],G(5)=[15:11],B(5)=[7:3]）.AEN 位控制混合功能可以使能或禁止。最后，设置窗口调色板控制寄存器位 0'b101.

表 14-1 25BPP (A:8:8:8)调色板数据格式

INDEX	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
\Bit	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
Pos.																									
00H	A	R	R	R	R	R	R	R	R	G	G	G	G	G	G	G	G	B	B	B	B	B	B	B	B
	E	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
N																									
01H	A	R	R	R	R	R	R	R	R	G	G	G	G	G	G	G	G	B	B	B	B	B	B	B	B
	E	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
N																									
.....	..	..	...	..	...	..	...	...	...	..	...	..	...	..	...	...	..	...	..	...	..	...	..	...	..
FFH	A	R	R	R	R	R	R	R	R	G	G	G	G	G	G	G	G	B	B	B	B	B	B	B	B
	E	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
N																									
VD 序号	-	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
		3	2	1	0	9	8	7	6	5	4	3	2	1	0										

表 14-2 25BPP (A:8:8:8)调色板数据格式

INDEX	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
\Bit	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
Pos.																									
00H	-	-	-	-	-	-	-	-	-	A	R	R	R	R	R	G	G	G	G	G	B	B	B	B	B
										E	4	3	2	1	0	4	3	2	1	0	4	3	2	1	0
N																									
01H	-	-	-	-	-	-	-	-	-	A	R	R	R	R	R	G	G	G	G	G	B	B	B	B	B
										E	4	3	2	1	0	4	3	2	1	0	4	3	2	1	0
N																									
.....	-	-	-	-	-	-	-	-	-	..	...	..	...	..	...	...	..	...	..	...	..	...	..	...	..
FFH	-	-	-	-	-	-	-	-	-	A	R	R	R	R	R	G	G	G	G	G	B	B	B	B	B

											E	4	3	2	1	0	4	3	2	1	0	4	3	2	1	0
											N															
VD 序	-	-	-	-	-	-	-	-	-	-	-	2	2	2	2	1	1	1	1	1	1	7	6	5	4	3
号												3	2	1	0	9	5	4	3	2	1					

### 14.2.6. 窗口混合

#### 1. 概述

VPRCS 模块的主要功能是窗口混合。显示控制器有 5 个窗口层，具体层的内容接下来会有具体的描述。例如，系统可以将窗口 0 作为 OS 窗口，满 TV 荧屏窗口等等。窗口 1 为一个小的 TV 屏幕，窗口 2 为菜单栏窗口，窗口 3 为标题窗口，窗口 4 为通道信息窗口。窗口 2、窗口 3、窗口 4 有颜色限制，通过颜色 LUT 的索引进行设置。这个性能通过减小整个系统的数据速率可以提高系统的运行能力。

5 个窗口实例说明

窗口 0（基本）：局部/（YcbCr，没有调色板的 RGB）

窗口 1（覆盖 1）：RGB 调色板

窗口 2（覆盖 2）：RGB 调色板

窗口 3（菜单）：16 级颜色 LUT 的 RGB（1/2/4）

窗口 4（光标区）：4 级颜色 LUT RGB(1/2)

覆盖优先级：

窗口 4>窗口 3>窗口 2>窗口 1>窗口 0

Color key: Color key 寄存器的值必选设置为 24 位 RGB 格式。

混合方程：

窗口 01(红)=窗口 0(红)\* Beta1+窗口 1(红)\*Alpha1

窗口 01(绿)=窗口 0(绿)\* Beta1+窗口 1(绿)\*Alpha1

窗口 01(蓝)=窗口 0(蓝)\* Beta1+窗口 1(蓝)\*Alpha1

窗口 012(红)=窗口 01(红)\* Beta2+窗口 2(红)\*Alpha2

窗口 012(绿)=窗口 01(绿)\* Beta2+窗口 2(绿)\*Alpha2



$\text{窗口 012(蓝)} = \text{窗口 01(蓝)} * \text{Beta2} + \text{窗口 2(蓝)} * \text{Alpha2}$   
 $\text{窗口 0123(红)} = \text{窗口 012(红)} * \text{Beta3} + \text{窗口 3(红)} * \text{Alpha3}$   
 $\text{窗口 0123(绿)} = \text{窗口 012(绿)} * \text{Beta3} + \text{窗口 3(绿)} * \text{Alpha3}$   
 $\text{窗口 0123(蓝)} = \text{窗口 012(蓝)} * \text{Beta3} + \text{窗口 3(蓝)} * \text{Alpha3}$   
 $\text{窗口 0123(绿)} = \text{窗口 012(绿)} * \text{Beta3} + \text{窗口 3(绿)} * \text{Alpha3}$   
 $\text{WinOut(红)} = \text{WinOut0123(红)} * \text{Beta4} + \text{WinOut4(红)} * \text{Alpha4}$   
 $\text{WinOut(绿)} = \text{WinOut0123(绿)} * \text{Beta4} + \text{WinOut4(绿)} * \text{Alpha4}$   
 $\text{WinOut(蓝)} = \text{WinOut0123(蓝)} * \text{Beta4} + \text{WinOut4(蓝)} * \text{Alpha4}$

Where

如果 A 位已经设置

$\text{AR1} = \text{窗口 1 红色混合因子 (ALPHA1\_R@VIDOSD1C)}$   
 $\text{AR2} = \text{窗口 2 红色混合因子 (ALPHA1\_R@VIDOSD2C)}$   
 $\text{AR3} = \text{窗口 3 红色混合因子 (ALPHA1\_R@VIDOSD3C)}$   
 $\text{AR4} = \text{窗口 4 红色混合因子 (ALPHA1\_R@VIDOSD4C)}$   
 $\text{AG1} = \text{窗口 1 绿色混合因子 (ALPHA1\_R@VIDOSD1C)}$   
 $\text{AG2} = \text{窗口 2 绿色混合因子 (ALPHA1\_R@VIDOSD2C)}$   
 $\text{AG3} = \text{窗口 3 绿色混合因子 (ALPHA1\_R@VIDOSD3C)}$   
 $\text{AG4} = \text{窗口 4 绿色混合因子 (ALPHA1\_R@VIDOSD4C)}$   
 $\text{AB1} = \text{窗口 1 蓝色混合因子 (ALPHA1\_R@VIDOSD1C)}$   
 $\text{AB2} = \text{窗口 2 蓝色混合因子 (ALPHA1\_R@VIDOSD2C)}$   
 $\text{AB3} = \text{窗口 3 蓝色混合因子 (ALPHA1\_R@VIDOSD3C)}$   
 $\text{AB4} = \text{窗口 4 蓝色混合因子 (ALPHA1\_R@VIDOSD4C)}$

如果 A 位已经清除，那么

$\text{AR1} = \text{窗口 1 红色混合因子 (ALPHA0\_R@VIDOSD1C)}$   
 $\text{AR2} = \text{窗口 2 红色混合因子 (ALPHA0\_R@VIDOSD2C)}$   
 $\text{AR3} = \text{窗口 3 红色混合因子 (ALPHA0\_R@VIDOSD3C)}$   
 $\text{AR4} = \text{窗口 4 红色混合因子 (ALPHA0\_R@VIDOSD4C)}$

AG1=窗口 1 绿色混合因子 (ALPHA0\_R@VIDOSD1C)

AG2=窗口 2 绿色混合因子 (ALPHA0\_R@VIDOSD2C)

AG3=窗口 3 绿色混合因子 (ALPHA0\_R@VIDOSD3C)

AG4=窗口 4 绿色混合因子 (ALPHA0\_R@VIDOSD4C)

AB1=窗口 1 蓝色混合因子 (ALPHA0\_R@VIDOSD1C)

AB2=窗口 2 蓝色混合因子 (ALPHA0\_R@VIDOSD2C)

AB3=窗口 3 蓝色混合因子 (ALPHA0\_R@VIDOSD3C)

AB4=窗口 4 蓝色混合因子 (ALPHA0\_R@VIDOSD4C)

一旦窗口 01 (红), Alpha 和 Beta 值通过下面关系式计算:

$\text{Alphax} = \text{Arx} / 16, \text{Betax} = (114 - \text{Arx}) / 16$

如果  $\text{Arx} == 0xF$ , 那么  $\text{Alphax} = 1, \text{Betax} = 0$

如果  $\text{Arx} == 0x0$ , 那么  $\text{Alphax} = 0, \text{Betax} = 1$

## 2.混合图表/详情

显示控制器可以在同一时间对针对同一个索引混合五个层。每个窗口层和颜色 (R,G,B) 都执行混合因子, 混合因子 alpha 值由 ALPHA0\_R, ALPHA0\_G, ALPHA0\_B, ALPHA1\_R, ALPHA1\_G, ALPHA1\_B 寄存器控制。图 14-5 描述了每个窗口使用 ALPHA\_R 值的红色输出。所有的窗口有两种 alpha 混合值。一个 alpha 混合值用来设置透明性使能 (AEN 值==1), 另一个 alpha 值用来设置透明性禁止 (AEN 值==0)。如果 WINEN\_F 和 BLD\_PLX 使能, 通过应用下面等式可以选择 AR:

$\text{AR} = (\text{像素(红)的 AEN 值} == 1'b1) ? \text{寄存器 (ALPHA1\_R)} : \text{寄存器 (ALPHA0\_R)} ;$

$\text{AG} = (\text{像素(绿)的 AEN 值} == 1'b1) ? \text{寄存器 (ALPHA1\_G)} : \text{寄存器 (ALPHA0\_G)} ;$

$\text{AB} = (\text{像素(蓝)的 AEN 值} == 1'b1) ? \text{寄存器 (ALPHA1\_B)} : \text{寄存器 (ALPHA0\_B)} ;$

(同时,  $\text{BLD\_PIC} == 1$ )

如果 WINEN\_F 使能和 BLD\_PLX 禁止, AR 值只能为 ALPHA0\_R。这种情况下, 混合因子 AR 由 ALPHA0\_R 固定, 禁止使用 AEN 位的信息。

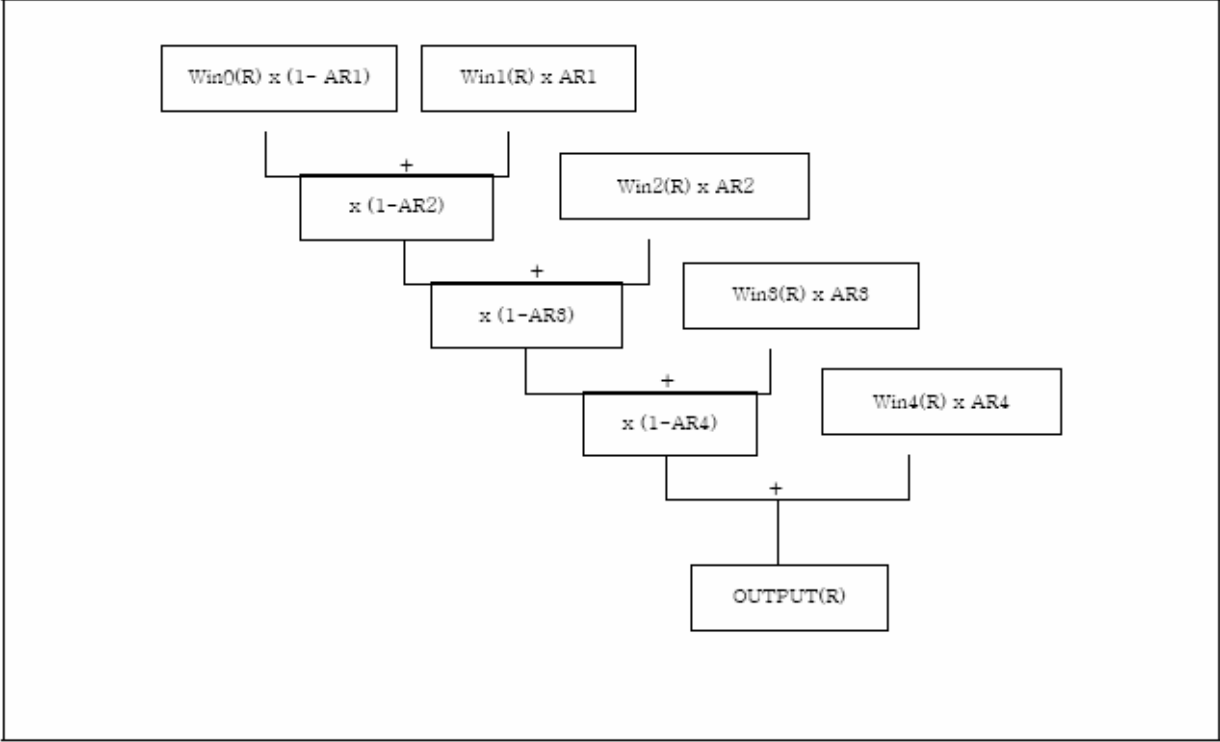


图 14-5 混合图

表 14-3 混合用户表

		ALPHA_SEL[1]value@WINCON1/2/3/4								
		‘0’	‘1’							
BLD_PIX[6]value@WINCON1/2/3/4	‘0’	用 ALPHA0 标准混合	用 ALPHA1 标准混合							
	‘1’	通过 AEN 值选择索引混合	只有当 BPPMODE_F[5:2]值  @WINCON1/2/3/4 为 ‘b1101’ 时，在帧  缓冲区内用 DATA[27:24]帧索引混合，							
		<table><tr><td colspan="2">AENvalue@Frame buffer</td></tr><tr><td>‘0’</td><td>‘1’</td></tr><tr><td>用 ALPHA0</td><td>用 ALPHA1</td></tr></table>		AENvalue@Frame buffer		‘0’	‘1’	用 ALPHA0	用 ALPHA1	
		AENvalue@Frame buffer								
‘0’		‘1’								
用 ALPHA0	用 ALPHA1									
或者										
		通过 KEYBLEN, Color key 混合使能								

KEYBLEN[26]@2W1/2/3/4KEYCON0	
‘0’	‘1’
Key 混合使能	没有 key 空间： 用 ALPHA0  Key 空间：用 ALPHA1

### 14.2.7. Color key 功能

显示控制器支持图像映射各种效果的 Color-Key 功能。OSD 层的彩色图像由 COLOR-KEY 寄存器指定，可以被背景图像替代用于特殊功能，可以替代为照相机的前景图形或光标图像。

ColorKey 寄存器的值必须设置为 24 位的 RGB 格式。

DIRCON 位选择的窗口与 COLVAL 进行比较。如果这个位置设置为 ‘0’，被比较的窗口为窗口 1。

COMPKEY 值聚顶是否与 COLVAL 进行比较，是否选择窗口颜色。换言之，当 COMPKEY 内相应的位设置为 ‘0’ 时，比较器只比较 COLVAL 和选择的窗口颜色。

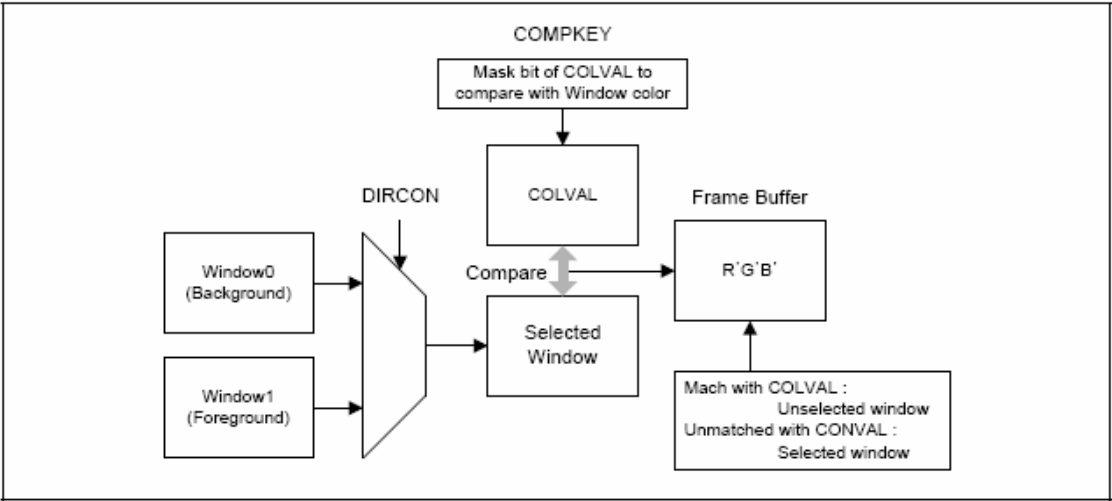


图 14-6 Color Key 操作

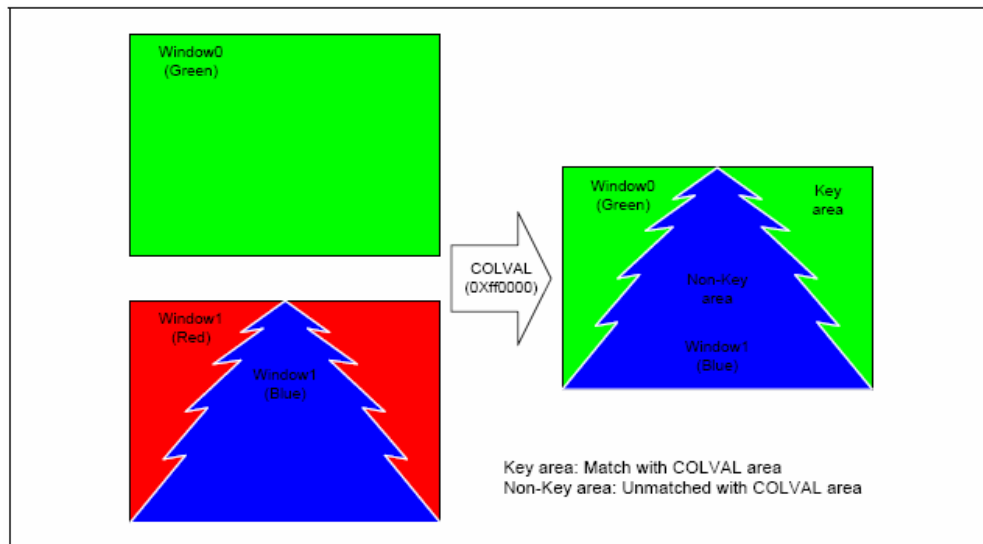


图 14-7 Color Key 操作

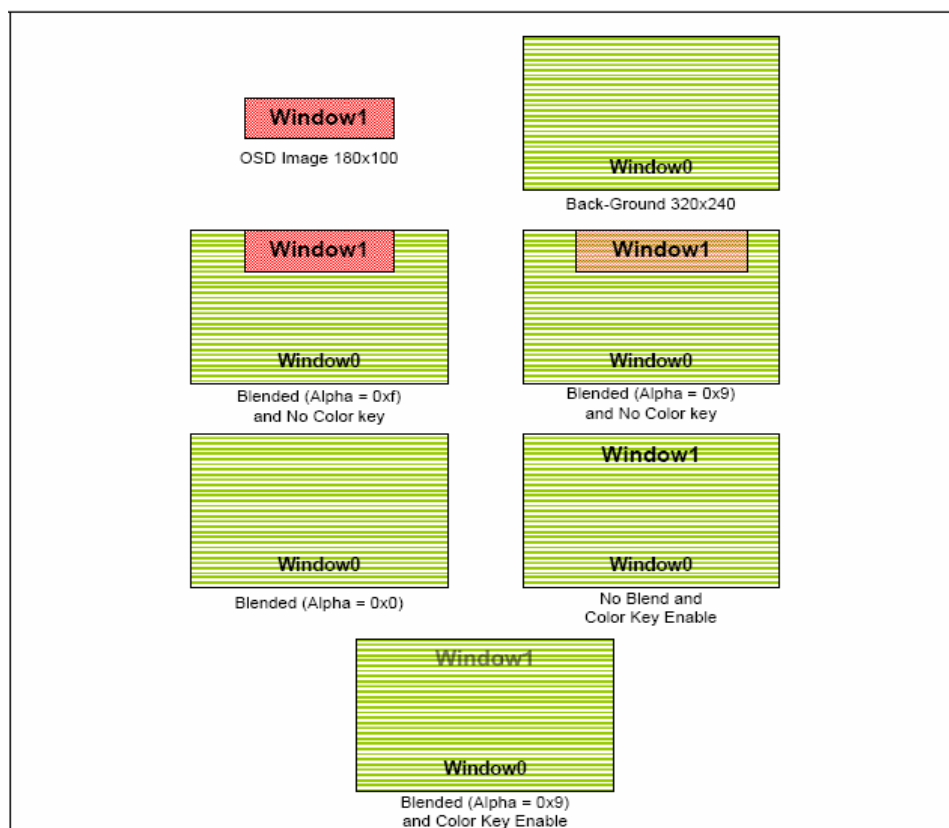


图 14-8 Color key 功能配置

### 14.2.8. VTIME 控制器操作

VTIME 主要分为两个模块。一个是 VTIME\_RGB\_TV 模块，用于 RGB 接口，ITU\_R601 接口和 TV 编码器接口时序控制。另一个是用于 I80 接口时序控制的模块。

#### 1. RGB 接口

VTIME 产生控制信号，如 RGBZ-VSYNC,RGB\_HSYNC,RGB\_VDEN 和 RGB\_VCLK 信号。这些控制信号与 VSFR 寄存器内的 VIDTCON0/1/2 寄存器的配置有很大的关系。根据 VSFR 内显示控制寄存器的可编程配置，VTIME 模块可以产生程序控制信号，这些控制信号适合于很多不同类型的显示设备。

RGB\_VSYNC 信号用来导致 LCD 行指针从显示的顶层开始覆盖。RGB\_VSYNC 和 RGB\_HSYNC 堆栈产生由 HOZVAL 区域和 LINEVAL 寄存器的配置控制。HOZVAL 和 LINEVAL 由 LCD 组的尺寸决定，具体等式如下：

$$\text{HOZVAL}=(\text{水平显示尺寸})-1$$

$$\text{LINEVAL}=(\text{垂直显示尺寸})-1$$

RGB\_VCLK 信号的速率可以由 VIDCON0 寄存器内的 CLKVAL 领域控制。RGB\_VCLK 和 CLKVAL 之间的关系见表 14-4. CLKVAL 最小值为 1.

$$\text{RGB\_VCLK(Hz)}=\text{HCLK}/(\text{CLKVAL}+1) \quad \text{CLKVAL} \geq 1$$

表 14-4 VCLK 和 CLKVAL 之间的关系

CLKVAL	60MHz/X	VCLK
1	60MHz/2	30.0MHz
2	60MHz/3	20.0MHz
3	60MHz/4	15.0MHz
...	..	..
63	60MHz/64	937.5kHz

通过 VSYNC,VBPD,VFPD,HSYNC,HBPD,HFPD, HOZVAL 和 LINEVAL 可以配置 RGB\_HSYNC 和 RGB\_VSYNC 信号。

帧速率是 RGB\_VSYNC 信号频率。帧速率与 VSYNC, VBPD, VFPD, LINEVAL, HSYNC, HBPD, HFPD, HOZVAL, CLKVAL 寄存器有关。很多 LCD 设备有它们自己的帧速率。通过下面关系式可以计算帧速率

$$\text{帧速率} = 1 / [ \{ (VSPW+1) + (VBPD+1) + (LIINEVAL+1) + (VFPD+1) \} * \{ (HSPW+1) + (HBPD+1) + (HFPD+1) + (HOZVAL+1) \} * \{ (CLKVAL+1) / (\text{时钟源频率}) \} ]$$

## 2.I80 接口控制器

VTIME\_I80 控制显示控制器的 CPU 类型 LDI，且具有以下功能：

- 产生 I80 接口控制信号
- CPU 类型 LDI 命令控制信号
- VDMA 和 VDPRCS 的时序控制

(1) 产生输出控制信号

SYS\_CS0, SYS\_CS1, SYS\_WE, SYS\_RS 控制信号由 VTIME\_I80 产生，这些信号的时序参数 LCD\_CS\_SETUP, LCD\_WR\_SETUP, LCD\_WR\_ACT, LCD\_WR\_HOLD 可以通过 I80IFCONA0 和 I80CONA1 SFR 设置。

(2) 部分显示控制

尽管部分显示控制是 CPU 类型 LDI 的主要性能，但是 VTIME\_I80 不支持这种功能。然而，可以通过 SFR 设置执行部分显示控制功能。

## 14.2.9. LDI 命令控制

LDI 可以接收命令和数据。命令表示 LDI 内的 SFR 选择的索引。在命令和数据的控制信号内，只有 SYS\_RS 信号有不同的操作。通常 SYS\_RS 的命令极性是 1，反之亦然。

显示控制器有两种命令控制形式。一种是自动命令控制形式，另一种是常规命令控制形式。自动命令在预定速率内自动发出。通过 S/W 控制可以发出常规命令。

命令设置

### 1.自动命令

如果在每 10 帧内 0x1, 0x32, 0x2, 0x8f, 0x4, 0x99 请求发送到 LDI，可以通过下面的步骤完成。

LDE\_CMD0 ← 0x1, LDE\_CMD1 ← 0x32, LDE\_CMD2 ← 0x2, LDE\_CMD3 ← 0x8f, LDE\_CMD4 ← 0x4,  
LDE\_CMD5 ← 0x99  
CMD0\_EN ← 0x2, CMD1\_EN ← 0x2, CMD2\_EN ← 0x2, CMD3\_EN ← 0x2, CMD4\_EN ← 0x2,  
CMD4\_EN ← 0x2

CMD0\_RS←0x1, CMD1\_RS←0x0, CMD2\_RS←0x1, CMD3\_RS←0x0, CMD4\_RS←0x1,  
CMD5\_RS←0x1  
AUTO\_CMD\_RATE←0x5

注意:

1) RS 极性可以参考 LDI 说明书。

2) LDI\_CMD 不需要从 LDI\_CMD0 到 LDI\_CMD11 连续使用。例如, 可以只用 LDI\_CMD0, LDI\_CMD3 和 LDI\_CMD11.

3) 最多有 12 个可用的自动命令。

## 2. 常规命令

1) 可以向 LDI\_CMO~11 内放入命令 (最多放 12 条命令)

2) 在 LDI\_CMDCON0 内设置 COMx\_EN, 可以使能常规命令 x。例如, 如果想使能命令 4, 可以设置 CMD4\_EN 到 0x01.

3) 在 I80IFCONB0/1 内设置 NORMAL\_CMD\_ST.

显示控制器命令操作有很多不同的特性, 如:

- 每 12 条命令可以有自动/常规/自动和常规模式。
- 在常规操作下, 显示控制器在帧之间最多可以传送 12 条命令。
- 命令产生的顺序是 CMD0←CMD1←CMD2←CMD3←……CMD10←CMD11
- 可以跳过无效的命令。
- 发送多于 12 条命令。只有在常规模式下可以发送多于 12 条的命令, 适合系统初始化。

## 14.2.10. RGB 接口 IO

表 14-5 RGB 接口管脚描述

名称	类型	源/目的地	描述
RGB_HSYNC	输出	Pad	水平同步. 信号
RGB_VSYNC	输出	Pad	垂直同步. 信号
RGB_VCLK	输出	Pad	LCD 视频时钟
RGB_VDEN	输出	Pad	数据使能



RGB_VD[23:0]	输出	Pad	RGB 数据输出。 在 16bpp 下，管脚匹配如下 RGB_VD[23:19]:红 RGB_VD[15:10]:绿 RGB_VD[7:3]:蓝
--------------	----	-----	--

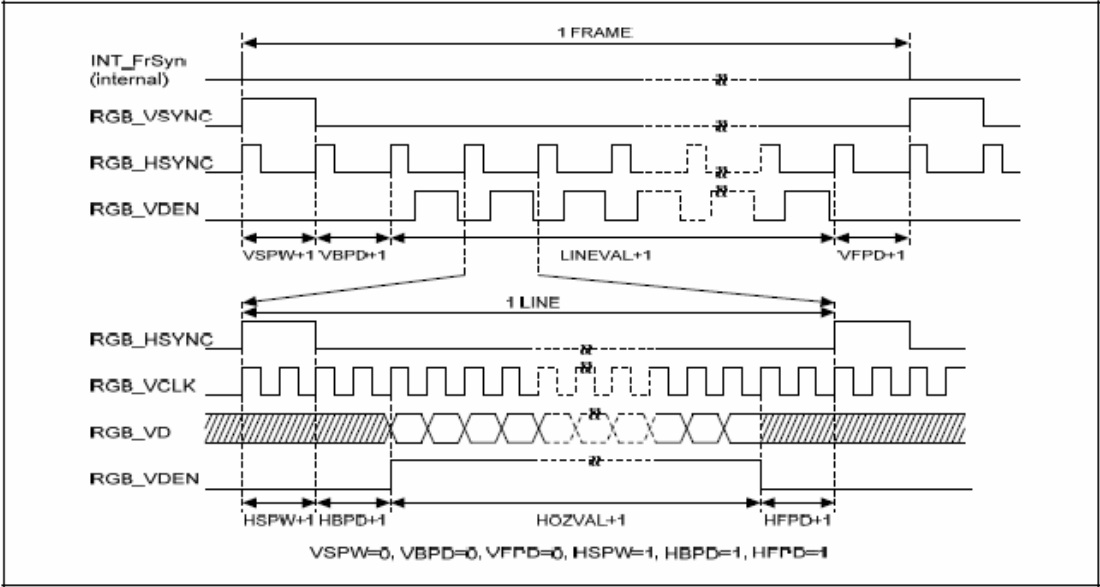


图 14-9 CD RGB 接口时序图

14.2.11. LCD I80 接口 IO

表 14-6 I80 CPU 接口管脚描述

名称	类型	源/目的地	描述
SYS_VDIN[17:0]	输入	Video mux	视频数据输入
SYS_VDOUT[17:0]	输出	Video mux	视频数据输出
SYS_CS0	输出	Video mux	LCD0 芯片选择
SYS_CS1	输出	Video mux	LCD1 芯片选择
SYS_WE	输出	Video mux	写使能

SYS_OE	输出	Video mux	输出使能
SYS_RS	输出	Video mux	寄存器/状态选择

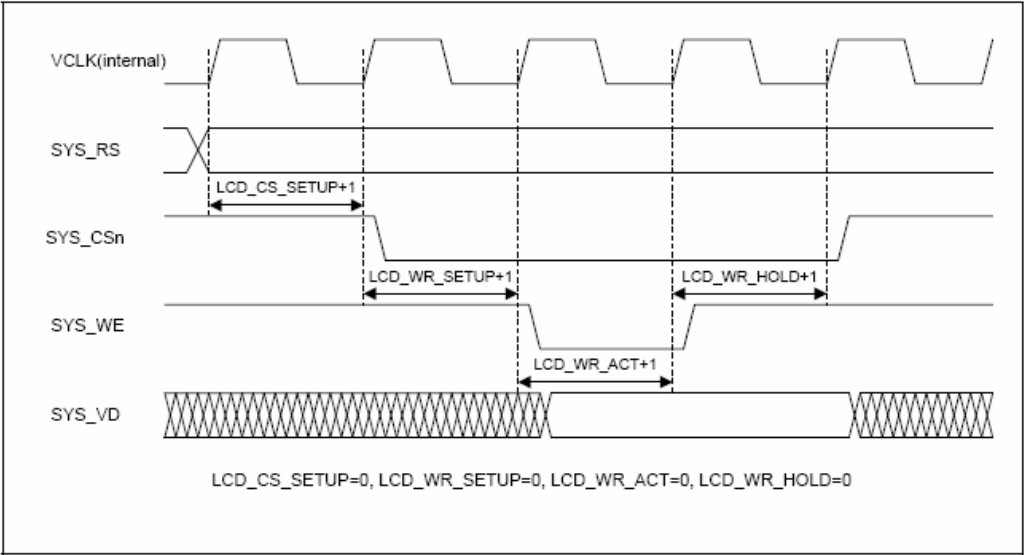


图 14-10 写周期时序图

14.2.12. ITU\_R BT.601 接口 IO

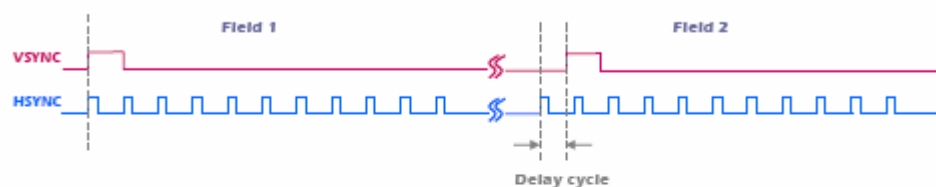
表 14-7 ITU-R BT. 601 接口管脚描述

名称	类型	源/目的地	描述
V601_CLK	输出	Pad	ITU 601 数据时钟
*VEN_HREF	输出	Pad	数据使能
**VEN_VSYNC	输出	Pad	垂直同步信号
VEN_HSYNC	输出	Pad	水平同步信号
**VEN_FIELD	输出	Pad	FIELD 信号（可选）
VEN_DATA[7:0]	输出	Pad	ITU601 YUV422 格式数据输出。

\*VEN\_HREF:数据空（当 I601HREF[0]=1）  
数据使能（当 I601HREF[0]=0）

\*\*VEN\_VSYNC, VEN\_FIELD

When SELVSYNC[0] = 1, Delay Cycle = DLYVSYNC[7:0] + 1



When SELVSYNC[0] = 0,

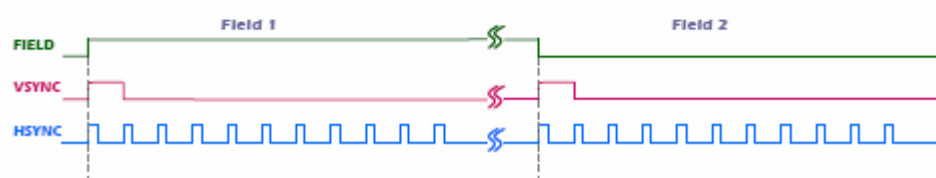


图 14-11 ITU-R BT.601 可控 Vsync

## 14.2.13. LCD 数据管脚映射

表 14-8 并行/串行 RGB, 601 数据管脚映射

	并联 RGB			串联 RGB		601
	24BPP(888)	18BPP(666)	16BPP(565)	24BPP(888)	18BPP(666)	
VD[23]	R[7]	R[5]	R[4]	D[7]	D[5]	
VD[22]	R[6]	R[4]	R[3]	D[6]	D[4]	
VD[21]	R[5]	R[3]	R[2]	D[5]	D[3]	
VD[20]	R[4]	R[2]	R[1]	D[4]	D[2]	
VD[19]	R[3]	R[1]	R[0]	D[3]	D[1]	
VD[18]	R[2]	R[0]	-	D[2]	D[0]	
VD[17]	R[1]	-	-	D[1]	-	
VD[16]	R[0]	-	-	D[0]	-	
VD[15]	G[7]	G[5]	G[5]	-	-	



VD[16]		-	-	-	-	-	-	R[4]	-	-
VD[15]	R[4]	R[5]	-	-	-	R[7]	B[7]	R[3]	-	-
VD[14]	R[3]	R[4]	-	-	-	R[6]	B[6]	R[2]	-	-
VD[13]	R[2]	R[3]	-	-	-	R[5]	B[5]	R[1]	-	-
VD[12]	R[1]	R[2]	-	-	-	R[4]	B[4]	R[0]	-	-
VD[11]	R[0]	R[1]	-	-	-	R[3]	B[3]	G[5]	-	-
VD[10]	G[5]	R[0]	-	-	-	R[2]	B[2]	G[4]	-	-
VD[9]	G[4]	G[5]	-	-	-	R[1]	B[1]	G[3]	-	-
VD[8]	G[3]	G[4]	-	R[5]	G[2]	R[0]	B[0]	G[2]	-	-
VD[7]	G[2]	G[3]	-	R[4]	G[1]	G[7]	-	G[1]	R[4]	G[2]
VD[6]	G[1]	G[2]	-	R[3]	G[0]	G[6]	-	G[0]	R[3]	G[1]
VD[5]	G[0]	G[1]	-	R[2]	B[5]	G[5]	-	B[5]	R[2]	G[0]
VD[4]	B[4]	G[0]	-	R[1]	B[4]	G[4]	-	B[4]	R[1]	B[4]
VD[3]	B[3]	B[5]	-	R[0]	B[3]	G[3]	-	B[3]	R[0]	B[3]
VD[2]	B[2]	B[4]	-	G[5]	B[2]	G[2]	-	B[2]	G[5]	B[2]
VD[1]	B[1]	B[3]	B[1]	G[4]	B[1]	G[1]	-	B[1]	G[4]	B[1]
VD[0]	B[0]	B[2]	B[0]	G[3]	B[0]	G[0]	-	B[0]	G[3]	B[0]

## 14.2.14. LCD 常规/BY-PASS 模式选择

外部调制解调器或 MCU 可以穿过 BY\_PASS 访问系统接口 LCD 板块。复位以后，LCD 控制器初始输出路径是 BY-PASS，如图 14-12 所示。为了在常规显示模式下运行，SEL\_BYPASS[3]值@0x7410800C 必须设置为'0'，设置为'1'的时候为 BY-PASS 模式。

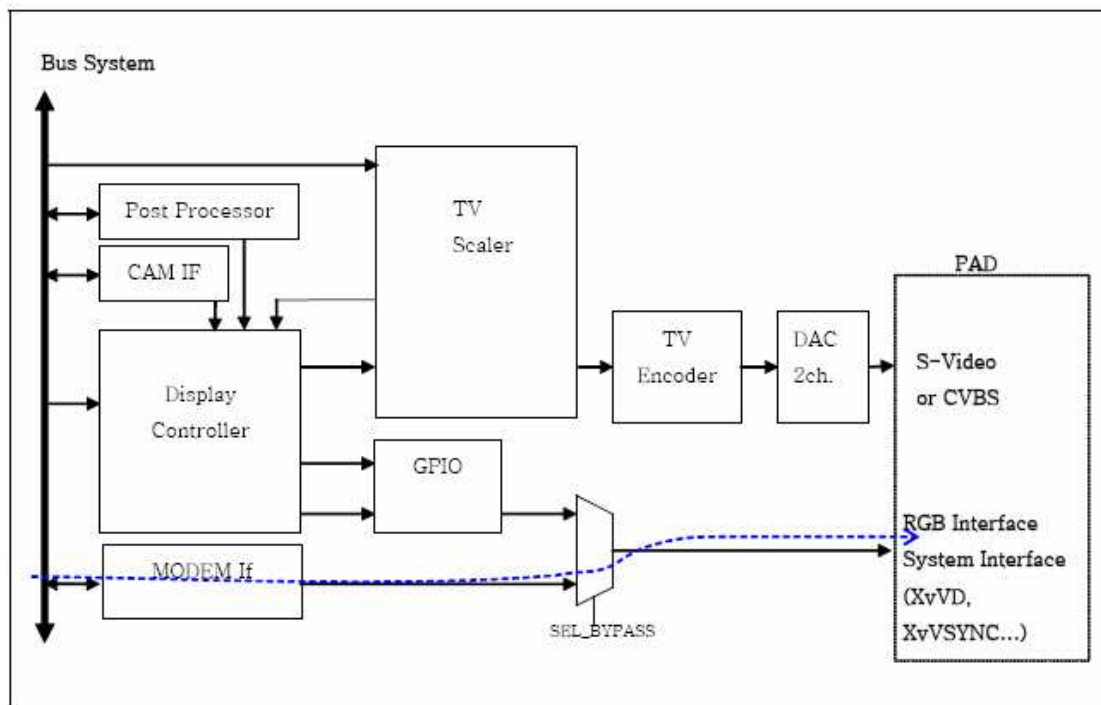


图 14-12 By-Pass 模式（初始 LCD 输出路径）

## 14.3 编程模型

### 14.2.1. 概述

用下面所诉寄存器配置显示控制器：

- (1) MOFPCON:SEL\_BYPASS[3] value@0x7410800C 必须设置为'0'.
- (2) SPCON:LCD\_SEL[1:0]value@0x74F0081A0 必须设置为'00'，使用主机 I/F 类型，或者设置为 '01' 使用 RGB I/F 类型。
- (3) VIDCON0:配置视频输出格式和显示使能/禁止。
- (4) VIDCON1:RGB I/F 控制信号。
- (5) I80IFCONx: i80 系统 I/F 控制信号。
- (6) ITUIFCON0:ITU(BT.601)接口控制

- (7) VIDTCONx: 配置视频输出时序和显示尺寸。
- (8) WINCONx: 窗口格式设置
- (9) VIDOSDxA , VIDOSDxB:窗口位置设置
- (10) VIDOSDxC: alpha 值设置
- (11) VIDWxxADDx:源图像地址设置
- (12) WxKEYCONx:色键值寄存器
- (13) WINxMAP:窗口颜色控制
- (14) WPALCON:调色板控制寄存器
- (15) WxPDATAxx: 索引窗口调色板数据

## 14.2.2. SFR 内存映射

寄存器	地址	读/写	描述	复位值
VIDCON0	0x77100000	读/写	视频控制 0 寄存器	0x0000_0000
VIDCON1	0x77100004	读/写	视频控制 1 寄存器	0x0000_0000
VIDCON2	0x77100008	读/写	视频控制 2 寄存器	0x0000_0000
VIDTCON0	0x77100010	读/写	视频时序控制 0 寄存器	0x0000_0000
VIDTCON1	0x77100014	读/写	视频时序控制 1 寄存器	0x0000_0000
VIDTCON2	0x77100018	读/写	视频时序控制 2 寄存器	0x0000_0000
WINCON0	0x77100020	读/写	窗口控制 0 寄存器	0x0000_0000
WINCON1	0x77100024	读/写	窗口控制 1 寄存器	0x0000_0000
WINCON2	0x77100028	读/写	窗口控制 2 寄存器	0x0000_0000
WINCON3	0x7710002C	读/写	窗口控制 3 寄存器	0x0000_0000
WINCON4	0x77100030	读/写	窗口控制 4 寄存器	0x0000_0000
VIDOSD0A	0x77100040	读/写	视频窗口 0 的位置控制寄存器	0x0000_0000
VIDOSD0B	0x77100044	读/写	视频窗口 0 的位置控制寄存器	0x0000_0000
VIDOSD0C	0x77100048	读/写	视频窗口 0 的尺寸控制寄存器	0x0000_0000
VIDOSD1A	0x77100050	读/写	视频窗口 1 的位置控制寄存器	0x0000_0000

VIDOSD1B	0x77100054	读/写	视频窗口 1 的位置控制寄存器	0x0000_0000
VIDOSD1C	0x77100058	读/写	视频窗口 1 的 alpha 控制寄存器	0x0000_0000
VIDOSD1D	0x7710005C	读/写	视频窗口 1 的尺寸控制寄存器	0x0000_0000
VIDOSD2A	0x77100060	读/写	视频窗口 2 的位置控制寄存器	0x0000_0000
VIDOSD2B	0x77100064	读/写	视频窗口 2 的位置控制寄存器	0x0000_0000
VIDOSD2C	0x77100068	读/写	视频窗口 2 的 alpha 控制寄存器	0x0000_0000
VIDOSD2D	0x7710006C	读/写	视频窗口 2 的尺寸控制寄存器	0x0000_0000
VIDOSD3A	0x77100070	读/写	视频窗口 3 的位置控制寄存器	0x0000_0000
VIDOSD3B	0x77100074	读/写	视频窗口 3 的位置控制寄存器	0x0000_0000
VIDOSD3C	0x77100078	读/写	视频窗口 3 的 alpha 控制寄存器	0x0000_0000
VIDOSD4A	0x77100080	读/写	视频窗口 4 的位置控制寄存器	0x0000_0000
VIDOSD4B	0x77100084	读/写	视频窗口 4 的位置控制寄存器	0x0000_0000
VIDOSD4C	0x77100088	读/写	视频窗口 4 的 alpha 控制寄存器	0x0000_0000
VIDW00ADD0B0	0x771000A0	读/写	窗口 0 的缓冲区开始地址寄存器，缓冲区 0	0x0000_0000
VIDW00ADD0B1	0x771000A4	读/写	窗口 0 的缓冲区开始地址寄存器，缓冲区 1	0x0000_0000
VIDW01ADD0B0	0x771000A8	读/写	窗口 1 的缓冲区开始地址寄存器，缓冲区 0	0x0000_0000
VIDW01ADD0B1	0x771000AC	读/写	窗口 1 的缓冲区开始地址寄存器，缓冲区 1	0x0000_0000
VIDW02ADD0	0x771000B0	读/写	窗口 2 的缓冲区开始地址寄存器	0x0000_0000
VIDW03ADD0	0x771000B8	读/写	窗口 3 的缓冲区开始地址寄存器	0x0000_0000
VIDW04ADD0	0x771000C0	读/写	窗口 4 的缓冲区开始地址寄存器	0x0000_0000
VIDW00ADD1B0	0x771000D0	读/写	窗口 0 的缓冲区开始地址寄存器，缓冲区 0	0x0000_0000
VIDW00ADD1B1	0x771000D4	读/写	窗口 0 的缓冲区开始地址寄存器，缓冲区 1	0x0000_0000
VIDW01ADD1B0	0x771000D8	读/写	窗口 1 的缓冲区开始地址寄存	0x0000_0000



			器，缓冲区 0	
VIDW01ADD1B1	0x771000DC	读/写	窗口 1 的缓冲区开始地址寄存器，缓冲区 1	0x0000_0000
VIDW02ADD1	0x771000E0	读/写	窗口 2 的缓冲区开始地址寄存器	0x0000_0000
VIDW03ADD1	0x771000E8	读/写	窗口 3 的缓冲区开始地址寄存器	0x0000_0000
VIDW04ADD1	0x771000F0	读/写	窗口 4 的缓冲区开始地址寄存器	0x0000_0000
VIDW00ADD2	0x77100100	读/写	窗口 0 的缓冲区开始地址寄存器	0x0000_0000
VIDW01ADD2	0x77100104	读/写	窗口 1 的缓冲区开始地址寄存器	0x0000_0000
VIDW02ADD2	0x77100108	读/写	窗口 2 的缓冲区开始地址寄存器	0x0000_0000
VIDW03ADD2	0x7710010C	读/写	窗口 3 的缓冲区开始地址寄存器	0x0000_0000
VIDW04ADD2	0x77100110	读/写	窗口 4 的缓冲区开始地址寄存器	0x0000_0000
VIDINTCON0	0x77100130	读/写	指明视屏中断控制寄存器	0x03F0_0000
VIDINTCON1	0x77100134	读/写	视频中断悬挂控制寄存器	0x0000_0000
W1KEYCON0	0x77100140	读/写	Color key 控制寄存器	0x0000_0000
W1KEYCON1	0x77100144	读/写	Color key 值寄存器	0x0000_0000
W2KEYCON0	0x77100148	读/写	Color key 控制寄存器	0x0000_0000
W2KEYCON1	0x7710014C	读/写	Color key 值寄存器	0x0000_0000
W3KEYCON0	0x77100150	读/写	Color key 控制寄存器	0x0000_0000
W3KEYCON1	0x77100154	读/写	Color key 值寄存器	0x0000_0000
W4KEYCON0	0x77100158	读/写	Color key 控制寄存器	0x0000_0000
W4KEYCON1	0x7710015C	读/写	Color key 值寄存器	0x0000_0000
DITHMODE	0x77100170	读/写	抖动模式寄存器	0x0000_0000
WINOMAP	0x77100180	读/写	窗口颜色控制	0x0000_0000
WIN1MAP	0x77100184	读/写	窗口颜色控制	0x0000_0000
WIN2MAP	0x77100188	读/写	窗口颜色控制	0x0000_0000
WIN3MAP	0x7710018C	读/写	窗口颜色控制	0x0000_0000
WIN4MAP	0x77100190	读/写	窗口颜色控制	0x0000_0000
WPALCON	0x771001A0	读/写	窗口调色板控制寄存器	0x0000_0000

TRIGCON	0x771001A4	读/写	I80/RGB 触发控制寄存器	0x0000_0000
ITUIFCON0	0x771001A8	读/写	ITU(BT. 601) 接口控制	0x0000_0000
I80IFCONA0	0x771001B0	读/写	主 LDI 的 I80 接口控制 0	0x0000_0000
I80IFCONA1	0x771001B4	读/写	子 LDI 的 I80 接口控制 0	0x0000_0000
I80IFCONB0	0x771001B8	读/写	主 LDI 的 I80 接口控制 1	0x0000_0000
I80IFCONB1	0x771001BC	读/写	子 LDI 的 I80 接口控制 1	0x0000_0000
LDI_CMDCON0	0x771001D0	读/写	I80 接口 LDI 命令控制 0	0x0000_0000
LDI_CMDCON1	0x771001D4	读/写	I80 接口 LDI 命令控制 1	0x0000_0000
SIFCCON0	0x771001E0	读/写	LCD I80 系统接口手册命令控制	0x0000_0000
SIFCCON1	0x771001E4	读/写	LCD I80 系统接口手册命令数据 写控制	0x0000_0000
SIFCCON2	0x771001E8	读/写	LCD I80 系统接口手册命令数据 读控制 2	未定义
LDI_CMD0	0x77100280	读/写	I80 接口 LDI 命令 0	0x0000_0000
LDI_CMD1	0x77100284	读/写	I80 接口 LDI 命令 1	0x0000_0000
LDI_CMD2	0x77100288	读/写	I80 接口 LDI 命令 2	0x0000_0000
LDI_CMD3	0x7710028C	读/写	I80 接口 LDI 命令 3	0x0000_0000
LDI_CMD4	0x77100290	读/写	I80 接口 LDI 命令 4	0x0000_0000
LDI_CMD5	0x77100294	读/写	I80 接口 LDI 命令 5	0x0000_0000
LDI_CMD6	0x77100298	读/写	I80 接口 LDI 命令 6	0x0000_0000
LDI_CMD7	0x7710029C	读/写	I80 接口 LDI 命令 7	0x0000_0000
LDI_CMD8	0x771002A0	读/写	I80 接口 LDI 命令 8	0x0000_0000
LDI_CMD9	0x771002A4	读/写	I80 接口 LDI 命令 9	0x0000_0000
LDI_CMD10	0x771002A8	读/写	I80 接口 LDI 命令 10	0x0000_0000
LDI_CMD11	0x771002AC	读/写	I80 接口 LDI 命令 11	0x0000_0000
W2PDATA01	0x77100300	读/写	索引 0, 1 的窗口 2 调色板数据	0x0000_0000
W2PDATA23	0x77100304	读/写	索引 2, 3 的窗口 2 调色板数据	0x0000_0000
W2PDATA45	0x77100308	读/写	索引 4, 5 的窗口 2 调色板数据	0x0000_0000

W2PDATA67	0x7710030C	读/写	索引 6, 7 的窗口 2 调色板数据	0x0000_0000
W2PDATA89	0x77100310	读/写	索引 8, 9 的窗口 2 调色板数据	0x0000_0000
W2PDATAAB	0x77100314	读/写	索引 A, B 的窗口 2 调色板数据	0x0000_0000
W2PDATAACD	0x77100318	读/写	索引 C, D 的窗口 2 调色板数据	0x0000_0000
W2PDATAEF	0x7710031C	读/写	索引 E, F 的窗口 2 调色板数据	0x0000_0000
W3PDATA01	0x77100320	读/写	索引 0, 1 的窗口 3 调色板数据	0x0000_0000
W3PDATA23	0x77100324	读/写	索引 2, 3 的窗口 3 调色板数据	0x0000_0000
W3PDATA45	0x77100328	读/写	索引 4, 5 的窗口 3 调色板数据	0x0000_0000
W3PDATA67	0x7710032C	读/写	索引 6, 7 的窗口 3 调色板数据	0x0000_0000
W3PDATA89	0x77100330	读/写	索引 8, 9 的窗口 3 调色板数据	0x0000_0000
W3PDATAAB	0x77100334	读/写	索引 A, B 的窗口 3 调色板数据	0x0000_0000
W3PDATAACD	0x77100338	读/写	索引 C, D 的窗口 3 调色板数据	0x0000_0000
W3PDATAEF	0x7710033C	读/写	索引 E, F 的窗口 3 调色板数据	0x0000_0000
W4PDATA01	0x77100340	读/写	索引 0, 1 的窗口 4 调色板数据	0x0000_0000
W4PDATA23	0x77100344	读/写	索引 2, 3 的窗口 4 调色板数据	0x0000_0000

## 14.4 INDIVIDUAL 寄存器描述

### 14.4.1. 视频主控制 0 寄存器

寄存器	地址	读/写	描述	复位值
VIDCON0	0x77100000	读/写	视频控制 0 寄存器	0x0000_0000

VIDCON0	位	描述	初始状态
Reserved	[31:30]	保留	0
INTERLACE_F	[29]	交错或渐进 0: 渐进	0

		1: 交错	
Reserved	[28]	保留	0
VIDOUT	[27:26]	定义显示控制器的输出格式  00: RGB I/F  01:TV 接口  10: LDI0 的 I80 I/F  11: LDI1 的 I80 I/F	00
L1_DATA16	[25:23]	选择 I80 I/F (LDI1) 输出数据格式的方式 (VIDOUT[1:0]==2' b00)  000=16 位模式 (16BPP)  001=16+2 位模式 (18BPP)  010=9+9 位模式 (18BPP)  011=16+8 位模式 (24BPP)  100=18 位模式 (18BPP)  101=8+8 位模式 (16BPP)	000
LO_DATA16	[22:20]	选择 I80 CPU I/F (LDI0) 输出数据格式的方式 (VIDOUT[1:0]==2' b00)  000=16 位模式 (16BPP)  001=16+2 位模式 (18BPP)  010=9+9 位模式 (18BPP)  011=16+8 位模式 (24BPP)  100=18 位模式 (18BPP)  101=8+8 位模式 (16BPP)	000
Reserved	[19]	保留	0
PNRMODE	[18:17]	选择显示模式(VIDOUT[1:0]==2' b00)  00=RGB 调色板格式 (RGB)  01= RGB 调色板格式 (RGB)  10=串行格式 (R->G->B)  11=串行格式 (R->G->B)	00

		选择显示模式 (VIDOUT[1:0] != 2'b00)  00=RGB 调色板格式 (RGB)	
CLKVALUP	[16]	选择 CLKVAL_F 更新时序控制  0=always  1=真开始时控制	0
Reserved	[15: 14]	保留	
CLKVAL_F	[13:6]	确定 VCLK 和 CLKVAL[7:0]的速率  VCLK=视频时钟源/ (CLKVAL+1)      CLKVAL>=1  注释： 1. VCLK 的最大频率是 66MHz  2. 通过 CLKSEL_F 寄存器选择视频时钟源	0
VCLKFREE	[5]	VCLK 自由运行控制（只有在 RGB IF 模式下有效）  0=常规模式（由 ENVID 控制）  1=自由运行模式	0
CLKDIR	[4]	直接选择时钟源或用 CLKVAL_F 寄存器划分时钟源  0=直接选择时钟源  1=用 CLKVAL_F 寄存器划分时钟源	0
CLKSEL_F	[3:2]	选择视频时钟源  00=HCLK  01=LCD 视频时钟  10=保留  11=27MHz Ext 时钟输入	0
ENVID	[1]	视频输出和逻辑瞬态使能/禁止  0=视频输出及显示控制信号使能  1=视频输出和显示控制信号禁止	0
ENVID_F	[0]	在当前帧末尾视频输出和逻辑瞬态使能/禁止  0=视频输出及显示控制信号使能  1=视频输出和显示控制信号禁止  如果此位设置或关闭，用户可以读取“H”，视频控制器一直处于工作状态知道当前帧结束为止。	0

注意:

显示开启: ENVID 和 ENVID\_F 设置为 1

直接关闭: ENVID 和 ENVID\_F 同时设置为 0

帧关闭: ENVID 设置为 0, ENVID\_F 设置为 1

警告 1: 在常规显示模式下, 必须将 SEL\_BYPASS@MIFPCON(0x7410\_800C)寄存器设置为 0

警告 2: 在交错模式下 VIDCON0 设置为 Per Frame off 时, 需要同时将 INTERLACE\_F 值设置为 0.

警告 3: 当用 direct\_off 关闭显示控制器时, 可以不通过复位直接打开显示控制器。

### 14.4.2. 视频主控制 1 寄存器

寄存器	地址	读/写	描述	复位值
VIDCON1	0x77100004	读/写	视频控制 1 寄存器	0x0000_0000

VIDCON1	位	描述	初始状态
LINECNT (只读)	[26:16]	提供行计数器状态 (只读) 从 0 开始向 LINEVAL 加计数。	0
FSTATUS	[15]	区域状态 (只读) 0=ODD 区域                      1=EVEN 区域	0
VSTATUS	[14:13]	垂直状态 (只读) 00= VSYNC      01=BACK Porch 10=ACTIVE      11=FRONT Porch	00
Reserved	[10:8]	保留	
IVCLK	[7]	此位控制 VCLK 活动边沿的极性 0=RGB 类型 LCD 设备在 VCLK 下降沿得到视频数据 1=RGB 类型 LCD 设备在 VCLK 上升沿得到视频数据	0
IHSYNC	[6]	此位指明 HSYNC 脉冲极性	0

		0=常规      1=反向	
IVSYNC	[5]	此位指明 VSYNC 脉冲极性 0=常规      1=反向	0
IVDEN	[4]	此位指明 VDEN 信号极性 0=常规      1=反向	0
Reserved	[3:0]	保留	0x0

### 14.4.3. 视频主控制 2 寄存器

寄存器	地址	读/写	描述	复位值
VIDCON2	0x77100008	读/写	视频控制 2 寄存器	0x0000_0000

VIDCON2	位	描述	初始状态
–	[31:24]	保留	0
EN601	[23]	控制 ITU601 输出使能 0=禁止      1=使能	0
–	[22:15]	保留	0
TVFORMATSELO	[14]	此位指明 YUV 数据格式选择方式 0=硬件选择 1=软件选择	0
TVFORMATSELO	[13:12]	此位指明 YUV 数据输出格式 00=RGB      01=YUV422      1x=YUV444	00
–	[11:9]	保留	0
OrgYCbCr	[8]	此位指明 YUV 数据的顺序 0=Y-CbCr      1= CbCr-Y	0
YUVOrd	[7]	此位指明 Chroma 数据的顺序 0=Cb-Cr      1= Cr-Cb	0
–	[6: 0]	保留	0

#### 14.4.4. 视频时序控制 0 寄存器

寄存器	地址	读/写	描述	复位值
VIDTCON0	0x77100010	读/写	视频时序控制 0 寄存器	0x0000_0000

VIDTCON0	位	描述	初始状态
VBPDE	[31:24]	在帧开始处，垂直后沿是不活动行的数量，在垂直同步过程之后（只对 YVU 接口）	0x0
VBPD	[23:16]	在帧开始处，垂直后沿是不活动行的数量，在垂直同步过程之后	0x0
VFPD	[15:8]	在帧末尾处，垂直前沿是不活动行的数量，在垂直同步过程之前。	0x0
VSPW	[7:0]	通过计算不活动行的数量，垂直同步脉冲宽度决定 VSYNC 脉冲的高点品宽度。	0x0

#### 14.4.5. 视频时序控制 1 寄存器

寄存器	地址	读/写	描述	复位值
VIDTCON1	0x77100014	读/写	视频时序控制 1 寄存器	0x0000_0000

VIDTCON1	位	描述	初始状态
VFPDE	[31:24]	在帧末尾处，垂直前沿是不活动行的数量，在垂直同步过程之前（只对 YVU 接口）	0x0
HBPDP	[23:16]	水平后沿是 HSYNC 下降沿和有效数据开始之间的 VCLK 周期的数量。	0x0
HFPDP	[15:8]	水平前沿是有效数据末端和 HSYNC 上升沿之间的 VCLK 周期的数量。	0x0
HSPW	[7:0]	通过计算不 VCLK 的数量，水平同步脉冲宽度决定 HSYNC 脉冲的高点品宽度。	0x0



WINCON0	位	描述	初始状态
nWide/Narrow	[27:26]	根据输入值范围选择从 YCbCr 到 RGB 的颜色空间转换等式。YCbCr 宽范围是 2' 00, YCbCr 窄范围是 2' 11, 宽范围: Y/Cb/Cr:2514-0 窄范围:Y:2314-16, Cb/Cr:240-16	00
Reserved	[25:23]	保留	0
ENLOCAL	[22]	选择数据访问方式 0: 专用的 DMA 1: 后处理程序的本地路径 注: 此寄存器必须指明 ENWIN_F 为 disable 状态。	0
BUFSTATUS	[21]	缓冲区状态 (只读) 0=缓冲区设置 0 1=缓冲区设置 1	0
BUFSEL	[20]	缓冲区状态 (0/1)	0

		0=缓冲区设置 0                      1=缓冲区设置 1	
BUFAUTOEN	[19]	双缓冲区自动控制位 0=由 BUFSEL 固定    1=通过 H/W 控制信号自动改变	0
BITSWP	[18]	位交换 控制位 0=交换禁止                      1=交换使能	0
BYTSWP	[17]	字节交换 控制位 0=交换禁止                      1=交换使能	0
HAWSWP	[16]	半字交换控制位 0=交换禁止                      1=交换使能	0
Reserved	[15:14]	必须为 0	0
InRGB	[13]	此位指明源图像的输入颜色空间（只对 ENLOCALI 有效）  0: RGB 1: YCbCr	0
Reserved	[12:11]	应该为 0	0
BURSTLEN	[10:9]	DMA 的突发最大长度选择  00: 16 字 01: 8 字 10: 4 字	0
Reserved	[8:6]	保留	0
BPPMODE_F	[5:2]	选择 BPP 模式窗口图像  0000=1BPP 0001=2BPP 0010=4BPP 0011=8BPP(palletized) 0100=保留 0101=16BPP (non-palletized, R: 14-G:6-B:5) 0110=保留 0111=16BPP( non-palletized, I:1-R:14-G:14-B:5)	0

		1000=unpacked 18BPP (non-palletized, R:6-G:6-B:6) 1010=保留 1011= unpacked 24BPP (non-palletized, R:8-G:8-B:8) 11xx=保留	
Reserved	[1]	保留	0
ENWIN_F	[0]	视频输出和逻辑瞬态使能/禁止 0=禁止视频输出和 VIDEO 控制信号 1=使能视频输出和 VIDEO 控制信号	0

### 14.4.8. 窗口 1 控制寄存器

寄存器	地址	读/写	描述	复位值
WINCON1	0x77100024	读/写	窗口 1 控制寄存器	0x0000_0000

WINCON1	位	描述	初始状态
nWide/Narrow	[27:26]	根据输入值范围选择从 YCbCr 到 RGB 的颜色空间转换等式。YCbCr 宽范围是 2' 00, YCbCr 窄范围是 2' 11, 宽范围: Y/Cb/Cr:2514-0 窄范围:Y:2314-16, Cb/Cr:240-16	00
Reserved	[25:24]	保留	00
LOCALSEL	[23]	选择本地路径源 0: TV 缩放 1: 摄像头预览本地 IF 注: 当选择摄像头预览本地 IF, 摄像头预览本地 IF 源必须为内存而不是外部摄像头。可以通过 MS-DM 从存储器向摄像头预览本地 IF 进行数据转换	0
ENLOCAL	[22]	选择数据访问方式	0

		0：专用的 DMA 1：本地路径 注：此寄存器必须指明 ENWIN_F 为 disable 状态。	
BUFSTATUS	[21]	缓冲区状态（只读） 0=缓冲区设置 0                      1=缓冲区设置 1	0
BUFSEL	[20]	缓冲区状态（0/1） 0=缓冲区设置 0                      1=缓冲区设置 1	0
BUFAUTOEN	[19]	双缓冲区自动控制位 0=由 BUFSEL 固定    1=通过 H/W 控制信号自动改变	0
BITSWP	[18]	位交换 控制位 0=交换禁止                      1=交换使能	0
BYTSWP	[17]	字节交换 控制位 0=交换禁止                      1=交换使能	0
HAWSWP	[16]	半字交换控制位 0=交换禁止                      1=交换使能	0
Reserved	[15:14]	必须为 0	0
InRGB	[13]	此位指明源图像的输入颜色空间（只对 ENLOCALI 效） 0：RGB 1：YCbCr	0
Reserved	[12:11]	应该为 0	0
BURSTLEN	[10:9]	DMA 的突发最大长度选择 00：16 字 01：8 字 10：4 字	0
Reserved	[8:7]	保留	0
BLD_PIX	[6]	选择混合类型 0=平面混合 1=像素混合	0

BPPMODE_F	[5:2]	<p>选择 BPP 模式窗口图像</p> <p>0000=1BPP</p> <p>0001=2BPP</p> <p>0010=4BPP</p> <p>0011=8BPP (palletized)</p> <p>0100=8BPP (non-palletized, A: 1-R:2-G:3-B:2)</p> <p>0101=16BPP (non-palletized, R: 14-G:6-B:5)</p> <p>0110=16BPP (non-palletized, A: 1-R:14-G:14-B:5)</p> <p>0111=16BPP (non-palletized, I:1-R:14-G:14-B:5)</p> <p>1000=unpacked 18BPP (non-palletized, R : 6-G:6-B:6)</p> <p>1001= unpacked 18 BPP (non-palletized, A: 1-R: 6-G: 6-B: 5 )</p> <p>1010=unpacked 19BPP (non-palletized, A: 1-R: 6-G: 6-B: 6)</p> <p>1011= unpacked 24 BPP (non-palletized, R : 8-G:8-B:8)</p> <p>1100= unpacked 24 BPP ( non-palletized, A: 1-R : 8-G: 8-B : 7)</p> <p>1101= unpacked 25 BPP ( non-palletized, A: 1-R : 8-G: 8-B : 8)</p> <p>111x=保留</p> <p>注: 在 BLD_PIX=1 时, 1101 也可以支持 unpacked 28 BPP ( non-palletized, A: 1-R : 8-G: 8-B : 8)</p>	0
ALPHA_SEL	[1]	<p>选择 Alpha 值</p> <p>平面混合情况 (BLD_PIX/=0)</p> <p>0=用 ALPHA0_R/G/B 值</p> <p>1=用 ALPHA1_R/G/B 值</p>	0

		像素混合 (BLD_PIX==1) 0=由 AEN 或 chromatic Key 选择 1= 用 DATA[27:24] 数据 (只有当 BPPMODE_F = 4' b1101 )	
ENWIN_F	[0]	视频输出和逻辑瞬态使能/禁止 0=禁止视频输出和 VIDEO 控制信号 1=使能视频输出和 VIDEO 控制信号	0

### 14.4.9. 窗口 2 控制寄存

寄存器	地址	读/写	描述	复位值
WINCON2	0x77100028	读/写	窗口 2 控制寄存器	0x0000_0000

WINCON2	位	描述	初始状态
nWide/Narrow	[27:26]	根据输入值范围选择从 YCbCr 到 RGB 的颜色空间转换等式。YCbCr 宽范围是 2' 00, YCbCr 窄范围是 2' 11, 宽范围: Y/Cb/Cr:2514-0 窄范围:Y:2314-16, Cb/Cr:240-16	00
Reserved	[25:24]	保留	00
LOCALSEL	[23]	选择本地路径源 0: TV 缩放 1: 摄像头预览本地 IF 注: 当选择摄像头预览本地 IF, 摄像头预览本地 IF 源必须为内存而不是外部摄像头。可以通过 MS-DMA 从存储器向摄像头预览本地 IF 进行数据转换	0
ENLOCAL	[22]	选择数据访问方式 0: 专用的 DMA 1: 本地路径 注: 此寄存器必须指明 ENWIN_F 为 disable 状态。	0

Reserved	[21:19]	保留	0
BITSWP	[18]	位交换 控制位 0=交换禁止                      1=交换使能	0
BYTSWP	[17]	字节交换 控制位 0=交换禁止                      1=交换使能	0
HAWSWP	[16]	半字交换控制位 0=交换禁止                      1=交换使能	0
Reserved	[15:14]	必须为 0	0
InRGB	[13]	此位指明源图像的输入颜色空间（只对 ENLOCALI 有效） 0: RGB 1: YCbCr	0
Reserved	[12:11]	应该为 0	0
BURSTLEN	[10:9]	DMA 的突发最大长度选择 00: 16 字 01: 8 字 10: 4 字	0
Reserved	[8:7]	保留	0
BLD_PIX	[6]	选择混合类型 0=平面混合 1=像素混合	0
BPPMODE_F	[5:2]	选择 BPP 模式窗口图像 0000=1BPP 0001=2BPP 0010=4BPP 0011=保留 0100=8BPP (non-palletized, A: 1-R:2-G:3-B:2) 0101=16BPP (non-palletized, R: 14-G:6-B:5) 0110=16BPP (non-palletized, A: 1-R:14-G:14-B:5)	0

		0111=16BPP( non-palletized, I:1-R:14-G:14-B:5) 1000=unpacked 18BPP(non-palletized , R : 6-G:6-B:6) 1001= unpacked 18 BPP (non-palletized, A: 1-R: 6-G: 6-B: 5 ) 1010=unpacked 19BPP (non-palletized, A: 1-R: 6-G: 6-B: 6) 1011= unpacked 24 BPP(non-palletized , R : 8-G:8-B:8) 1100= unpacked 24 BPP ( non-palletized, A: 1-R : 8-G: 8-B : 7) 1101= unpacked 25 BPP ( non-palletized, A: 1-R : 8-G: 8-B : 8) 111x=保留 注: 在 BLD_PIX=1 时, 1101 也可以支持 unpacked 28 BPP( non-palletized, A: 1-R : 8-G: 8-B : 8)	
ALPHA_SEL	[1]	选择 Alpha 值 平面混合情况 (BLD_PIX==0) 0=用 ALPHA0_R/G/B 值 1=用 ALPHA1_R/G/B 值 像素混合 (BLD_PIX==1) 0=由 AEN 或 chromatic Key 选择 1= 用 DATA[27:24] 数据 ( 只有当 BPPMODE_F = 4' b1101)	0
ENWIN_F	[0]	视频输出和逻辑瞬态使能/禁止 0=禁止视频输出和 VIDEO 控制信号 1=使能视频输出和 VIDEO 控制信号	0



14.4.10. 窗口 3 控制寄存器

寄存器	地址	读/写	描述	复位值
WINCON3	0x7710002C	读/写	窗口 3 控制寄存器	0x0000_0000

WINCON3	位	描述	初始状态
BITSWP	[18]	位交换 控制位 0=交换禁止                      1=交换使能	0
BYTSWP	[17]	字节交换 控制位 0=交换禁止                      1=交换使能	0
HAWSWP	[16]	半字交换控制位 0=交换禁止                      1=交换使能	0
Reserved	[15:11]	必须为 0	0
BURSTLEN	[10:9]	DMA 的突发最大长度选择  00: 16 字 01: 8 字 10: 4 字	0
Reserved	[8:7]	保留	0
BLD_PIX	[6]	选择混合类型  0=平面混合 1=像素混合	0
BPPMODE_F	[5:2]	选择 BPP 模式窗口图像  0000=1BPP (LUT) 0001=2BPP (LUT) 0010=4BPP (LUT) 0011=保留 0100=保留 0101=16BPP (non-palletized, R: 14-G:6-B:5)	0

		0110=16BPP (non-palletized, A: 1-R:14-G:14-B:5) 0111=16BPP ( non-palletized, I:1-R:14-G:14-B:5) 1000=unpacked 18BPP(non-palletized , R : 6-G:6-B:6) 1001= unpacked 18 BPP (non-palletized, A: 1-R: 6-G: 6-B: 5 ) 1010=unpacked 19BPP (non-palletized, A: 1-R: 6-G: 6-B: 6) 1011= unpacked 24 BPP(non-palletized , R : 8-G:8-B:8) 1100= unpacked 24 BPP ( non-palletized, A: 1-R : 8-G: 8-B : 7) 1101= unpacked 25 BPP ( non-palletized, A: 1-R : 8-G: 8-B : 8) 111x=保留 注: 在 BLD_PIX=1 时, 1101 也可以支持 unpacked 28 BPP( non-palletized, A:4-R : 8-G: 8-B : 8)	
ALPHA_SEL	[1]	选择 Alpha 值 平面混合情况 (BLD_PIX==0) 0=用 ALPHA0_R/G/B 值 1=用 ALPHA1_R/G/B 值 像素混合 (BLD_PIX==1) 0=由 AEN 或 chromatic Key 选择 1= 用 DATA[27:24] 数据 ( 只有当 BPPMODE_F = 4' b1101)	0
ENWIN_F	[0]	视频输出和逻辑瞬态使能/禁止 0=禁止视频输出和 VIDEO 控制信号 1=使能视频输出和 VIDEO 控制信号	0

14.4.11. 窗口 4 控制寄存器

寄存器	地址	读/写	描述	复位值
WINCON4	0x77100030	读/写	窗口 4 控制寄存器	0x0000_0000

WINCON4	位	描述	初始状态
BITSWP	[18]	位交换 控制位 0=交换禁止                      1=交换使能	0
BYTSWP	[17]	字节交换 控制位 0=交换禁止                      1=交换使能	0
HAWSWP	[16]	半字交换控制位 0=交换禁止                      1=交换使能	0
Reserved	[15:11]	必须为 0	0
BURSTLEN	[10:9]	DMA 的突发最大长度选择  00: 16 字  01: 8 字  10: 4 字	0
Reserved	[8:7]	保留	0
BLD_PIX	[6]	选择混合类型  0=平面混合  1=像素混合	0
BPPMODE_F	[5:2]	选择 BPP 模式窗口图像  0000=1BPP (LUT)  0001=2BPP (LUT)  0010=保留  0011=保留  0100=保留  0101=16BPP (non-palletized, R: 14-G:6-B:5)	0

		<p>0110=16BPP (non-palletized, A: 1-R:14-G:14-B:5)</p> <p>0111=16BPP ( non-palletized, I:1-R:14-G:14-B:5)</p> <p>1000=unpacked 18BPP(non-palletized , R : 6-G:6-B:6)</p> <p>1001= unpacked 18 BPP (non-palletized, A: 1-R: 6-G: 6-B: 5 )</p> <p>1010=unpacked 19BPP (non-palletized, A: 1-R: 6-G: 6-B: 6)</p> <p>1011= unpacked 24 BPP(non-palletized , R : 8-G:8-B:8)</p> <p>1100= unpacked 24 BPP ( non-palletized, A: 1-R : 8-G: 8-B : 7)</p> <p>1101= unpacked 25 BPP ( non-palletized, A: 1-R : 8-G: 8-B : 8)</p> <p>111x=保留</p> <p>注: 在 BLD_PIX=1 时, 1101 也可以支持 unpacked 28 BPP( non-palletized, A:4-R : 8-G: 8-B : 8)</p>	
ALPHA_SEL	[1]	<p>选择 Alpha 值</p> <p>平面混合情况 (BLD_PIX==0)</p> <p>0=用 ALPHA0_R/G/B 值</p> <p>1=用 ALPHA1_R/G/B 值</p> <p>像素混合 (BLD_PIX==1)</p> <p>0=由 AEN 或 chromatic Key 选择</p> <p>1= 用 DATA[27:24] 数据 ( 只有当 BPPMODE_F = 4' b1101)</p>	0
ENWIN_F	[0]	<p>视频输出和逻辑瞬态使能/禁止</p> <p>0=禁止视频输出和 VIDEO 控制信号</p> <p>1=使能视频输出和 VIDEO 控制信号</p>	0

### 14.4.12. 窗口 0 位置控制 A 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD0A	0x77100040	读/写	视频窗口 0 的位置控制寄存器	0x0000_0000

VIDOSD0A	位	描述	初始状态
OSD_LeftTopX_F	[21:11]	左上角的 OSD 图像像素横向屏幕坐标	0
OSD_LeftTopY_F	[10:0]	左上角的 OSD 图像像素纵向屏幕坐标（对于接口 TV 输出而言，这个值必须设置在屏幕 y 坐标值的一半位置。Y 坐标值的初始位置必须为偶数）	0

### 14.4.13. 窗口 0 位置控制 B 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD0B	0x77100044	读/写	视频窗口 0 的位置控制寄存器	0x0000_0000

VIDOSD0B	位	描述	初始状态
OSD_RightBotX_F	[21:11]	右下角的 OSD 图像像素横向屏幕坐标	0
OSD_RightBotY_F	[10:0]	右下角的 OSD 图像像素纵向屏幕坐标（对于接口 TV 输出而言，这个值必须设置在屏幕 y 坐标值的一半位置。Y 坐标值的初始位置必须为奇数）	0

### 14.4.14. 窗口 0 位置控制 C 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD0C	0x77100048	读/写	视频窗口 0 的位置控制寄存器	0x0000_0000

VIDOSDOC	位	描述	初始状态
-	[25:24]	保留	0
OSDSIZE	[23:0]	窗口尺寸=高*宽（字数）	0

#### 14.4.15. 窗口 1 位置控制 A 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD1A	0x77100050	读/写	视频窗口 1 的位置控制 2 寄存器	0x0000_0000

VIDOSD1A	位	描述	初始状态
OSD_LeftTopX_F	[21:11]	左上角的 OSD 图像像素横向屏幕坐标	0
OSD_LeftTopY_F	[10:0]	左上角的 OSD 图像像素纵向屏幕坐标（对于接口 TV 输出而言，这个值必须设置在屏幕 y 坐标值的一半位置。Y 坐标值的初始位置必须为偶数）	0

#### 14.4.16. 窗口 1 位置控制 B 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD1B	0x77100054	读/写	视频窗口 1 的位置控制寄存器	0x0000_0000

VIDOSD1B	位	描述	初始状态
OSD_RightBotX_F	[21:11]	右下角的 OSD 图像像素横向屏幕坐标	0
OSD_RightBotY_F	[10:0]	右下角的 OSD 图像像素纵向屏幕坐标（对于接口 TV 输出而言，这个值必须设置在屏幕 y 坐标值的一半位置。Y 坐标值的初始位置必须为奇数）	0

14.4.17. 窗口 1 位置控制 C 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD1C	0x77100058	读/写	视频窗口 1 的 alpha 控制寄存器	0x0000_0000

VIDOSD1C	位	描述	初始状态
-	[24]	保留	0
ALPHA0_R	[23:20]	红色 Alpha 值 (AEN==0)	0
ALPHA0_G	[19:16]	绿色 Alpha 值 (AEN==0)	0
ALPHA0_B	[15:12]	蓝色 Alpha 值 (AEN==0)	0
ALPHA1_R	[11:8]	红色 Alpha 值 (AEN==1)	0
ALPHA1_G	[7:4]	绿色 Alpha 值 (AEN==1)	0
ALPHA1_B	[3:0]	蓝色 Alpha 值 (AEN==1)	0

14.4.18 窗口 1 位置控制 D 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD1D	0x7710005C	读/写	视频窗口 1 的尺寸控制寄存器	0x0000_0000

VIDOSD1D	位	描述	初始状态
-	[25]	保留	0
-	[24]	保留	0
OSDSIZE	[23:0]	窗口尺寸=高*宽 (字数)	0

### 14.4.19 窗口 2 位置控制 A 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD2A	0x77100060	读/写	视频窗口 2 的位置控制寄存器	0x0000_0000

VIDOSD2A	位	描述	初始状态
OSD_LeftTopX_F	[21:11]	左上角的 OSD 图像像素横向屏幕坐标	0
OSD_LeftTopY_F	[10:0]	左上角的 OSD 图像像素纵向屏幕坐标（对于接口 TV 输出而言，这个值必须设置在屏幕 y 坐标值的一半位置。Y 坐标值的初始位置必须为偶数）	0

### 14.4.20. 窗口 2 位置控制 B 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD2B	0x77100064	读/写	视频窗口 2 的位置控制寄存器	0x0000_0000

VIDOSD2B	位	描述	初始状态
OSD_RightBotX_F	[21:11]	右下角的 OSD 图像像素横向屏幕坐标	0
OSD_RightBotY_F	[10:0]	右下角的 OSD 图像像素纵向屏幕坐标（对于接口 TV 输出而言，这个值必须设置在屏幕 y 坐标值的一半位置。Y 坐标值的初始位置必须为奇数）	0

### 14.4.21. 窗口 2 位置控制 C 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD2C	0x77100068	读/写	视频窗口 2 的 alpha 控制寄存器	0x0000_0000



VIDOSD2C	位	描述	初始状态
-	[24]	保留	0
ALPHA0_R	[23:20]	红色 Alpha 值 (AEN==0)	0
ALPHA0_G	[19:16]	绿色 Alpha 值 (AEN==0)	0
ALPHA0_B	[15:12]	蓝色 Alpha 值 (AEN==0)	0
ALPHA1_R	[11:8]	红色 Alpha 值 (AEN==1)	0
ALPHA1_G	[7:4]	绿色 Alpha 值 (AEN==1)	0
ALPHA1_B	[3:0]	蓝色 Alpha 值 (AEN==1)	0

#### 14.4.22 窗口 2 位置控制 D 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD2D	0x7710006C	读/写	视频窗口 1 的尺寸控制寄存器	0x0000_0000

VIDOSD1D	位	描述	初始状态
-	[25:24]	保留	0
OSDSIZE	[23:0]	窗口尺寸=高*宽 (字数)	0

#### 14.4.23. 窗口 3 位置控制 A 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD3A	0x77100070	读/写	视频窗口 3 的位置控制寄存器	0x0000_0000

VIDOSD3A	位	描述	初始状态
OSD_LeftTopX_F	[21:11]	左上角的 OSD 图像像素横向屏幕坐标	0
OSD_LeftTopY_F	[10:0]	左上角的 OSD 图像像素纵向屏幕坐标 (对于接口 TV 输出而言, 这个值必须设置在屏幕 y 坐标值的一半位置。Y 坐标值的初始位置必须为偶数)	0

14.4.24. 窗口 3 位置控制 B 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD3B	0x77100074	读/写	视频窗口 3 的位置控制寄存器	0x0000_0000

VIDOSD3B	位	描述	初始状态
OSD_RightBotX_F	[21:11]	右下角的 OSD 图像像素横向屏幕坐标	0
OSD_RightBotY_F	[10:0]	右下角的 OSD 图像像素纵向屏幕坐标（对于接口 TV 输出而言，这个值必须设置在屏幕 y 坐标值的一半位置。Y 坐标值的初始位置必须为奇数）	0

14.4.25. 窗口 3 位置控制 C 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD3C	0x77100078	读/写	视频窗口 3 的 alpha 控制寄存器	0x0000_0000

VIDOSD3C	位	描述	初始状态
-	[24]	保留	0
ALPHA0_R	[23:20]	红色 Alpha 值（AEN==0）	0
ALPHA0_G	[19:16]	绿色 Alpha 值（AEN==0）	0
ALPHA0_B	[15:12]	蓝色 Alpha 值（AEN==0）	0
ALPHA1_R	[11:8]	红色 Alpha 值（AEN==1）	0
ALPHA1_G	[7:4]	绿色 Alpha 值（AEN==1）	0
ALPHA1_B	[3:0]	蓝色 Alpha 值（AEN==1）	0

### 14.4.26. 窗口 4 位置控制 A 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD4A	0x77100080	读/写	视频窗口 4 的位置控制寄存器	0x0000_0000

VIDOSD4A	位	描述	初始状态
OSD_LeftTopX_F	[21:11]	左上角的 OSD 图像像素横向屏幕坐标	0
OSD_LeftTopY_F	[10:0]	左上角的 OSD 图像像素纵向屏幕坐标（对于接口 TV 输出而言，这个值必须设置在屏幕 y 坐标值的一半位置。Y 坐标值的初始位置必须为偶数）	0

### 14.4.27. 窗口 4 位置控制 B 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD4B	0x77100084	读/写	视频窗口 4 的位置控制寄存器	0x0000_0000

VIDOSD4B	位	描述	初始状态
OSD_RightBotX_F	[21:11]	右下角的 OSD 图像像素横向屏幕坐标	0
OSD_RightBotY_F	[10:0]	右下角的 OSD 图像像素纵向屏幕坐标（对于接口 TV 输出而言，这个值必须设置在屏幕 y 坐标值的一半位置。Y 坐标值的初始位置必须为奇数）	0

### 14.4.28. 窗口 4 位置控制 C 寄存器

寄存器	地址	读/写	描述	复位值
VIDOSD4C	0x77100088	读/写	视频窗口 4 的 alpha 控制寄存器	0x0000_0000

VIDOSD4C	位	描述	初始状态
-	[24]	保留	0
ALPHA0_R	[23:20]	红色 Alpha 值 (AEN==0)	0
ALPHA0_G	[19:16]	绿色 Alpha 值 (AEN==0)	0
ALPHA0_B	[15:12]	蓝色 Alpha 值 (AEN==0)	0
ALPHA1_R	[11:8]	红色 Alpha 值 (AEN==1)	0
ALPHA1_G	[7:4]	绿色 Alpha 值 (AEN==1)	0
ALPHA1_B	[3:0]	蓝色 Alpha 值 (AEN==1)	0

#### 14.4.29. 帧缓冲地址 0 寄存器

寄存器	地址	读/写	描述	复位值
VIDW00ADD0B0	0x771000A0	读/写	窗口 0 的缓冲区开始地址控制寄存器, 缓冲区 0	0x0000_0000
VIDW00ADD0B1	0x771000A4	读/写	窗口 0 的缓冲区开始地址控制寄存器, 缓冲区 1	0x0000_0000
VIDW01ADD0B0	0x771000A8	读/写	窗口 1 的缓冲区开始地址控制寄存器, 缓冲区 0	0x0000_0000
VIDW01ADD0B1	0x771000AC	读/写	窗口 1 的缓冲区开始地址控制寄存器, 缓冲区 1	0x0000_0000
VIDW02ADD0	0x771000B0	读/写	窗口 2 的缓冲区开始地址控制寄存器	0x0000_0000
VIDW03ADD0	0x771000B8	读/写	窗口 3 的缓冲区开始地址控制寄存器	0x0000_0000
VIDW04ADD0	0x771000C0	读/写	窗口 4 的缓冲区开始地址控制寄存器	0x0000_0000

VIDWxxADD0	位	描述	初始状态
VBANIK_F	[31:24]	这些位指明系统内存内的视频缓冲区的 A[31:24]bank 位置	0
VBASEU_F	[23:0]	这些位指明系统内存内的视频缓冲区的 A[23:0] bank 位置	0

### 14.4.30. 帧缓冲地址 1 寄存器

寄存器	地址	读/写	描述	复位值
VIDW00ADD1B0	0x771000D0	读/写	窗口 0 的缓冲区结束地址控制寄存器, 缓冲区 0	0x0000_0000
VIDW00ADD1B1	0x771000D4	读/写	窗口 0 的缓冲区结束地址控制寄存器, 缓冲区 1	0x0000_0000
VIDW01ADD1B0	0x771000D8	读/写	窗口 1 的缓冲区结束地址控制寄存器, 缓冲区 0	0x0000_0000
VIDW01ADD1B1	0x771000DC	读/写	窗口 1 的缓冲区结束地址控制寄存器, 缓冲区 1	0x0000_0000
VIDW02ADD1	0x771000E0	读/写	窗口 2 的缓冲区结束地址控制寄存器	0x0000_0000
VIDW03ADD1	0x771000E8	读/写	窗口 3 的缓冲区结束地址控制寄存器	0x0000_0000
VIDW04ADD1	0x771000F0	读/写	窗口 4 的缓冲区结束地址控制寄存器	0x0000_0000

VIDWxxADD1	位	描述	初始状态
VBASEL_F	[23:0]	这些位指明视频帧缓冲区的结束地址 A[23:0]。 $VBASEL = VBASEU + (PAGEWIDTH + OFFSIZE) * (LINEVAL + 1)$	0x0

### 14.4.31. 帧缓冲地址 2 寄存器

寄存器	地址	读/写	描述	复位值
VIDW00ADD2	0x77100100	读/写	窗口 0 的缓冲区尺寸寄存器	0x0000_0000
VIDW01ADD2	0x77100104	读/写	窗口 1 的缓冲区尺寸寄存器	0x0000_0000
VIDW02ADD2	0x77100108	读/写	窗口 2 的缓冲区尺寸寄存器	0x0000_0000
VIDW03ADD2	0x7710010C	读/写	窗口 3 的缓冲区尺寸寄存器	0x0000_0000
VIDW04ADD2	0x77100110	读/写	窗口 4 的缓冲区尺寸寄存器	0x0000_0000

VIDWxxADD2	位	描述	初始状态
OFFSIZE_F	[25:13]	虚拟屏幕补偿区尺寸 此值指明了在上一视频行内显示最后字节的地址与在新的视频行内显示的第一个字节地址的不同。	0

		OFFSIZE_F 的值可以是四个字节大小的不同值，或者为 0	
PAGEWIDTH_F	[12:0]	虚拟平面的页面宽度 此值定义了帧内可是端口的宽度。PAGEWIDTH 的值必须比突发尺寸大且在字边界范围内。	0

#### 14.4.32. 视频中断控制 0 寄存器

寄存器	地址	读/写	描述	复位值
VIDINTCON0	0x77100130	读/写	指明视频中断控制 0 寄存器	0x3F00000

VIDINTCON0	位	描述	初始状态
FIFOINTERVAL	[25:20]	这些位控制 FIFO 中断的 间隔时间	0x3F
SYSMAINCON	[19]	向主 LCD 发送完成中断使能位 0=中断禁止 1=中断使能	0
SYSSUBCON	[18]	向子 LCD 发送完成中断使能位 0=中断禁止 1=中断使能	0
I80IFDONE	[17]	I80 接口中断使能控制（只针对 I80 接口模式） 0=中断禁止 1=中断使能	0
FRAMESEL0	[16:15]	视频帧中断 0 开始以： 00=后沿            01=VSYNC 10=ACTIVE        11=前沿	0
FRAMESEL1	[14:13]	视频帧中断 1 开始以： 00=无            01=后沿 10=VSYNC        11=前沿	0
INTFRMEN	[12]	视频帧中断使能控制位	0

		0=视频帧中断 disable 1=视频帧中断 enable	
FIFOSEL	[11:5]	FIFO 中断控制位，每个位代表的意义如下： [11]窗口 4 控制（0: disable, 1: enable） [10]窗口 4 控制（0: disable, 1: enable） [9]窗口 2 控制（0: disable, 1: enable） [8]保留 [7]保留 [6]窗口 1 控制（0: disable, 1: enable） [5]窗口 0 控制（0: disable, 1: enable）	0
FIFOLEVEL	[4:2]	视频 FIFO 中断级别选择 000=0~25% 001=0~50% 010=0~75% 011=0%（空） 100=100%（满）	0
INTFIFOEN	[1]	视频 FIFO 中断使能控制位 0=视频 FIFO 级别中断 disable 1=视频 FIFO 级别中断 enable	0
INTEN	[1]	视频中断使能控制位 0=视频中断禁止 1=视频中断使能	0

#### 14.4.33. 视频中断控制 1 寄存器

寄存器	地址	读/写	描述	复位值
VIDINTCON1	0x77100134	读/写	视频中断悬挂寄存器	0x0000_0000

VIDINTCON1	位	描述	初始状态
Reserved	[4:3]	保留	0
INTI80PEND	[2]	I80 Done 中断。写入'1'清除此位。 0=中断未被请求 1=I80 Done 状态声明中断请求	0
INTFRMPEND	[1]	帧同步中断。写入'1'清除此位。 0=中断未被请求 1=帧同步状态声明中断请求	0
INTFIFOPEND	[2]	FIFO Level 中断。写入'1'清除此位。 0=中断未被请求 1=FIFO 空状态声明中断请求	0

#### 14.4.34. WIN1 color key 0 寄存器

寄存器	地址	读/写	描述	复位值
W1KEYCON0	0x77100140	读/写	Color key 控制寄存器	0x0000_0000

W1KEYCON0	位	描述	初始状态
KEYBLEN	[26]	Color Key 使能控制 0=Disable 1=non_key 区域用 ALPHA0_x 混合,key 区域用 ALPHA1_x 混合(x=R,G,B)	0
KEYEN_F	[25]	Color Key 使能控制 0=Disable 1=non_key 区域用 ALPHA0_x 混合,key 区域用 ALPHA1_x 混合(x=R,G,B)	0
DIRCON	[24]	Color Key 方向控制 0=如果前景图像的索引值与 COLVAL 匹配, 将显示背景图像的索引值(只在 OSD 空间)。 1=如果背景图像的索引值与 COLVAL 匹配, 将显示前景图像的索引值(只在 OSD 空间)。	0



COMPKEY	[23:0]	与 COLVAL[23:0]位一致。如果一些位置的位被设置，那么 COLVAL 中相对应得位在 for-ground 或背景匹配时将被忽略。	0
---------	--------	--	---

### 14.4.35. WIN1 color key 1 寄存器

寄存器	地址	读/写	描述	复位值
W1KEYCON1	0x77100144	读/写	Color key 值寄存器	0x0000_0000

W1KEYCON1	位	描述	初始状态
COLVAL	[23:0]	透明像素效果的 Color key 值。	0

### 14.4.36. WIN2 color key 0 寄存器

寄存器	地址	读/写	描述	复位值
W2KEYCON0	0x77100148	读/写	Color key 控制寄存器	0x0000_0000

W2KEYCON0	位	描述	初始状态
KEYBLEN	[26]	Color Key 使能控制 0=Disable 1=non_key 区域用 ALPHA0_x 混合，key 区域用 ALPHA1_x 混合（x=R,G,B）	0
KEYEN_F	[25]	Color Key 使能控制 0=Disable 1=non_key 区域用 ALPHA0_x 混合，key 区域用 ALPHA1_x 混合（x=R,G,B）	0
DIRCON	[24]	Color Key 方向控制 0=如果前景图像的索引值与 COLVAL 匹配，将显示背景图像的索引值(只在 OSD 空间)。 1=如果背景图像的索引值与 COLVAL 匹配，将显示前景图像的索引值(只在 OSD 空间)。	0

COMPKEY	[23:0]	与 COLVAL[23:0]位一致。如果一些位置的位被设置，那么 COLVAL 中相对应得位在 for-ground 或背景匹配时将被忽略。	0
---------	--------	--	---

### 14.4.37. WIN2 color key 1 寄存器

寄存器	地址	读/写	描述	复位值
W2KEYCON1	0x7710014C	读/写	Color key 值寄存器	0x0000_0000

W2KEYCON1	位	描述	初始状态
COLVAL	[23:0]	透明像素效果的 Color key 值。	0

### 14.4.38. WIN3 color key 0 寄存器

寄存器	地址	读/写	描述	复位值
W3KEYCON0	0x77100150	读/写	Color key 控制寄存器	0x0000_0000

W3KEYCON0	位	描述	初始状态
KEYBLEN	[26]	Color Key 使能控制 0=Disable 1=non_key 区域用 ALPHA0_x 混合，key 区域用 ALPHA1_x 混合（x=R,G,B）	0
KEYEN_F	[25]	Color Key 使能控制 0=Disable 1=non_key 区域用 ALPHA0_x 混合，key 区域用 ALPHA1_x 混合（x=R,G,B）	0
DIRCON	[24]	Color Key 方向控制 0=如果前景图像的索引值与 COLVAL 匹配，将显示背景图像的索引值(只在 OSD 空间)。 1=如果背景图像的索引值与 COLVAL 匹配，将显示	0

		前景图像的索引值(只在 OSD 空间)。	
COMPKEY	[23:0]	与 COLVAL[23:0]位一致。如果一些位置的位被设置，那么 COLVAL 中相对应得位在 for-ground 或背景匹配时将被忽略。	0

#### 14.4.39. WIN3 color key 1 寄存器

寄存器	地址	读/写	描述	复位值
W3KEYCON1	0x77100154	读/写	Color key 值寄存器	0x0000_0000

W3KEYCON1	位	描述	初始状态
COLVAL	[23:0]	透明像素效果的 Color key 值。	0

#### 14.4.40. WIN4 color key 0 寄存器

寄存器	地址	读/写	描述	复位值
W4KEYCON0	0x77100158	读/写	Color key 控制寄存器	0x0000_0000

W4KEYCON0	位	描述	初始状态
KEYBLEN	[26]	Color Key 使能控制 0=Disable 1=non_key 区域用 ALPHA0_x 混合，key 区域用 ALPHA1_x 混合 (x=R,G,B)	0
KEYEN_F	[25]	Color Key 使能控制 0=Disable 1=non_key 区域用 ALPHA0_x 混合，key 区域用 ALPHA1_x 混合 (x=R,G,B)	0
DIRCON	[24]	Color Key 方向控制 0=如果前景图像的索引值与 COLVAL 匹配，将显示	0

		背景图像的索引值(只在 OSD 空间)。 1=如果背景图像的索引值与 COLVAL 匹配，将显示前景图像的索引值(只在 OSD 空间)。	
COMPKEY	[23:0]	与 COLVAL[23:0]位一致。如果一些位置的位被设置，那么 COLVAL 中相对应得位在 for-ground 或背景匹配时将被忽略。	0

#### 14.4.41. WIN4 color key 1 寄存器

寄存器	地址	读/写	描述	复位值
W4KEYCON1	0x7710015C	读/写	Color key 值寄存器	0x0000_0000

W4KEYCON1	位	描述	初始状态
COLVAL	[23:0]	透明像素效果的 Color key 值。	0

#### 14.4.42. 抖动控制 1 寄存器

寄存器	地址	读/写	描述	复位值
DITHMODE	0x77100170	读/写	抖动模式寄存器	0x0000_0000

DITHMODE	位	描述	初始状态
-	[7]	值为 0	0
RDithPos	[6:5]	红色抖动位控制 00: 8 位 01: 6 位 10: 5 位	0
GDithPos	[4:3]	绿色抖动位控制 00: 8 位 01: 6 位	0

		10: 5 位	
BDithPos	[2:1]	蓝色抖动位控制  00: 8 位  01: 6 位  10: 5 位	0
DITHEN_F	[0]	抖动位使能位  0=抖动禁止  1=抖动使能	0

#### 14.4.43. WIN0 色彩映射

寄存器	地址	读/写	描述	复位值
WIN0MAP	0x77100180	读/写	窗口色彩控制	0x000000

WIN0MAP	位	描述	初始状态
MAPCOLEN_F	[24]	窗口色彩映射控制位。  如果此位 enabled，视频 DMA 将停止，同时 MAPCOLOR 将会出现在背景图像上，替换原始图像。  0=disable  1=enable	0
MAPCOLOR	[23:0]	色彩值	0

#### 14.4.44. WIN1 色彩映射

寄存器	地址	读/写	描述	复位值
WIN1MAP	0x77100184	读/写	窗口色彩控制	0x000000

WIN1MAP	位	描述	初始状态
MAPCOLEN_F	[24]	窗口色彩映射控制位。	0

		如果此位 <b>enabled</b> ，视频 DMA 将停止，同时 <b>MAPCOLOR</b> 将会出现在背景图像上，替换原始图像。 0=禁止 1=使能	
MAPCOLOR	[23:0]	色彩值	0

#### 14.4.45. WIN2 色彩映射

寄存器	地址	读/写	描述	复位值
WIN2MAP	0x77100188	读/写	窗口色彩控制	0x000000

WIN2MAP	位	描述	初始状态
MAPCOLEN_F	[24]	窗口色彩映射控制位。 如果此位 <b>enabled</b> ，视频 DMA 将停止，同时 <b>MAPCOLOR</b> 将会出现在背景图像上，替换原始图像。 0=禁止 1=使能	0
MAPCOLOR	[23:0]	色彩值	0

#### 14.4.46. WIN3 色彩映射

寄存器	地址	读/写	描述	复位值
WIN3MAP	0x7710018C	读/写	窗口色彩控制	0x000000

WIN3MAP	位	描述	初始状态
MAPCOLEN_F	[24]	窗口色彩映射控制位。 如果此位 <b>enabled</b> ，视频 DMA 将停止，同时 <b>MAPCOLOR</b> 将会出现在背景图像上，替换原始图像。 0=禁止	0

		1=使能	
MAPCOLOR	[23:0]	色彩值	0

#### 14.4.47. WIN4 色彩映射

寄存器	地址	读/写	描述	复位值
WIN4MAP	0x77100190	读/写	窗口色彩控制	0x0000_0000

WIN4MAP	位	描述	初始状态
MAPCOLEN_F	[24]	窗口色彩映射控制位。 如果此位 enabled，视频 DMA 将停止，同时 MAPCOLOR 将会出现在背景图像上，替换原始图像。 0=禁止 1=使能	0
MAPCOLOR	[23:0]	色彩值	0

#### 14.4.48. 窗口调色板控制器

寄存器	地址	读/写	描述	复位值
WPALCON	0x771001A0	读/写	窗口调色板控制寄存器	0x0000_0000

WPALCON	位	描述	初始状态
PALUPADTEEN	[9]	调色板存储器访问权利控制位。 在访问调色板存储器之前可以设置此位，此时,LCD 控制器不可以访问调色板。更新以后，用户可以清楚此位进行 palletezed LCD 运行。 0:常规模式（LCD 控制器访问） 1: 使能（ARM 访问）	0
W4PAL	[8]	此位决定窗口 4 调色板数据格式的大小	0

		0=16 位 (5:6:5) 1=16 位 (A:5:5:5)	
W4PAL	[8]	此位决定窗口 4 调色板数据格式的大小 0=16 位 (5:6:5) 1=16 位 (A:5:5:5)	0
W3PAL	[7]	此位决定窗口 3 调色板数据格式的大小 0=16 位 (5:6:5) 1=16 位 (A:5:5:5)	0
W2PAL	[6]	此位决定窗口 2 调色板数据格式的大小 0=16 位 (5:6:5) 1=16 位 (A:5:5:5)	0
W1PAL	[5:3]	此位决定窗口 1 调色板数据格式的大小 000=25 位 (A:8:8:8)    001=24 位(8:8:8) 010=19 位 (A:6:6:6)    011=18 位(A: 6:6:5) 100=18 位 (6:6:6)    101= 16 位(A:5:5:5) 110=16 位(5:6:5)	0
W0PAL	[2:0]	此位决定窗口 0 调色板数据格式的大小 000=25 位 (A:8:8:8)    001=24 位(8:8:8) 010=19 位 (A:6:6:6)    011=18 位(A: 6:6:5) 100=18 位 (6:6:6)    101= 16 位(A:5:5:5) 110=16 位(5:6:5)	0

#### 14.4.49. I80/RGB 触发控制寄存器

寄存器	地址	读/写	描述	复位值
TRIGCON	0x771001A4	读/写	I80/RGB 触发控制寄存器	0x0000_0000

TRIGCON	位	描述	初始状态
Reserved	[7:3]	必须为 0	0



SWFRSTATUS	[2]	帧完成 状态[只读] 0:表示未完成 I80 帧转换 1:表示已经完成 I80 帧转换 清除条件：读或新帧开始 只有当 TRGMODE 为 1 时有效	0
SWTRGCMD	[1]	1:软件触发命令（只能写） 只有当 TRGMODE 为 1 时有效	0
TRGMODE	[0]	软件触发使能控制 0=禁止 1=使能	0

#### 14.4.50. ITU 601 接口控制 0

寄存器	地址	读/写	描述	复位值
ITUIFCON0	0x771001A8	读/写	ITU(BT.601)接口控制	0x0000_0000

WPALCON	位	描述	初始状态
Reserved	[26:25]	保留	0
SELVSYNC	[24]	选择 VSYNC 模式 0: 与 HSYCN 前沿相同 1: 延迟的 VSYNC	0
DLYVSYNC	[23:16]	VSYNC 信号延迟的时钟周期数（只有当 SELVSYNC 为 1 时有效）DLYSYNC+1	
Reserved	[15:7]	保留	0
I601HREF	[6]	VEN_HREF 信号极性 0: 常规 1: 被声明的极性	0
I601VSYNC	[5]	VEN_VSYNC 信号极性 0: 常规 1: 被声明的极性	0
I601HSYNC	[4]	VEN_HSYNC 信号极性	0

		0: 常规 1: 被声明的极性	
I601FIELD	[3]	VEN_FIELD 信号极性 0: 常规 1: 被声明的极性	0
I601CLK	[2]	V601_CLK 信号极性 0: 常规 1: 被声明的极性	0
-	[1:0]	保留	0

#### 14.4.51. LCD I80 接口控制 0

寄存器	地址	读/写	描述	复位值
I80IFCONA0	0x771001B0	读/写	主 LDI(LCD) 的 I80 接口控制	0x0
I80IFCONA1	0x771001B4	读/写	子 LDI(LCD) 的 I80 接口控制	0x0

I80IFCONAx	位	描述	初始状态
Reserved	[22:20]	保留	0
LCD_CS_SETUP	[19:16]	地址信号使能向芯片选择使能活动期间的时钟周期数	0
LCD_WR_SETUP	[15:12]	CS 信号使能向写信号使能活动期间的时钟周期数	0
LCD_WR_HOLD	[7:4]	芯片选择使能向写信号使能活动期间的时钟周期数	0
Reserved	[3]	保留	0
RSPOL	[2]	RS 信号极性 0: RS 信号在视频数据转换期间为低电平 1: RS 信号在视频数据转换器件为高电平	0
Reserved	[1]	保留	0
I80IFEN	[0]	LCD I80 接口控制 0=禁止 1=使能	0

### 14.4.52. LCD I80 接口控制 1

寄存器	地址	读/写	描述	复位值
I80IFCONB0	0x771001B8	读/写	主 LDI(LCD) 的 I80 接口控制	0x0
I80IFCONB1	0x771001BC	读/写	子 LDI(LCD) 的 I80 接口控制	0x0

I80IFCONAx	位	描述	初始状态
Reserved	[11:0]	保留	0
NORMAL_CMD_ST	[9]	常规命令开始 传送一个设置命令后将自动清除	0
Reserved	[8:7]	保留	0
FRAME_SKIP	[6:5]	I80 接口输出帧抽取因子 00: 1 01: 2 10: 3	00
Reserved	[4]	保留	0
AUTO_CMD_RATE	[3:0]	0000:自动命令 disable 0001: 每两帧 0010: 每 4 帧 0011: 每 6 帧 ... 1111: 每 30 帧	0000

### 14.4.53. LCD I80 接口命令控制 0

寄存器	地址	读/写	描述	复位值
LDI_CMDCON0	0x771001D0	读/写	I80 系统接口命令控制 0	0x0000_0000

LDI_CMDCON0	位	描述	初始状态
Reserved	[31: 24]	保留	
CMD11_EN	[23:22]	00:禁止            01:常规命令使能 10: 自动命令使能 11: 常规和自动命令使能	00
CMD10_EN	[21:20]	00:禁止            01:常规命令使能 10: 自动命令使能 11: 常规和自动命令使能	00
CMD9_EN	[19:18]	00:禁止            01:常规命令使能 10: 自动命令使能 11: 常规和自动命令使能	00
CMD8_EN	[17:16]	00:禁止            01:常规命令使能 10: 自动命令使能 11: 常规和自动命令使能	00
CMD7_EN	[15:14]	00:禁止            01:常规命令使能 10: 自动命令使能 11: 常规和自动命令使能	00
CMD6_EN	[13:12]	00:禁止            01:常规命令使能 10: 自动命令使能 11: 常规和自动命令使能	00
CMD5_EN	[11:10]	00:禁止            01:常规命令使能 10: 自动命令使能 11: 常规和自动命令使能	00
CMD4_EN	[9:8]	00:禁止            01:常规命令使能 10: 自动命令使能 11: 常规和自动命令使能	00
CMD3_EN	[7:6]	00:禁止            01:常规命令使能 10: 自动命令使能 11: 常规和自动命令使能	00

CMD2_EN	[5:4]	00:禁止 10: 自动命令使能 11: 常规和自动命令使能	01:常规命令使能	00
CMD1_EN	[3:2]	00:禁止 10: 自动命令使能 11: 常规和自动命令使能	01:常规命令使能	00
CMD0_EN	[1:0]	00:禁止 10: 自动命令使能 11: 常规和自动命令使能	01:常规命令使能	00

#### 14.4.54. LCD I80 接口命令控制 1

寄存器	地址	读/写	描述	复位值
LDI_CMDCON1	0x771001D4	读/写	I80 系统接口命令控制 1	0x0000_0000

LDI_CMDCON1	位	描述	初始状态
Reserved	[31:10]	保留	
CMD11_RS	[11]	命令 11 RS 控制	00
CMD10_RS	[10]	命令 10 RS 控制	00
CMD9_RS	[9]	命令 9 RS 控制	00
CMD8_RS	[8]	命令 8 RS 控制	00
CMD7_RS	[7]	命令 7 RS 控制	00
CMD6_RS	[6]	命令 6 RS 控制	00
CMD5_RS	[5]	命令 5 RS 控制	00
CMD4_RS	[4]	命令 4 RS 控制	00
CMD3_RS	[3]	命令 3 RS 控制	00
CMD2_RS	[2]	命令 2 RS 控制	00
CMD1_RS	[1]	命令 1 RS 控制	00
CMD0_RS	[0]	命令 0 RS 控制	00

### 14.4.55. I80 系统接口菜单命令控制 0

寄存器	地址	读/写	描述	复位值
SIFCCON0	0x771001E0	读/写	I80 系统接口菜单命令控制	0x0000_0000

SIFCCON0	位	描述	初始状态
Reserved	[7]	保留	
SYS_ST_CON	[6]	LCD I80 系统接口 ST 信号控制 0: 低电平                      1: 高电平	0
SYS_RS_CON	[5]	LCD I80 系统接口 RS 信号控制 0: 低电平                      1: 高电平	0
SYS_nCS0_CON	[4]	LCD I80 系统接口 nCS0 信号控制 0: 禁止(高电平)      1: 使能(低电平)	0
SYS_nCS1_CON	[3]	LCD I80 系统接口 nCS1 信号控制 0: 禁止(高电平)      1: 使能(低电平)	0
SYS_nOE_CON	[2]	LCD I80 系统接口 nOE 信号控制 0: 禁止(高电平)      1: 使能(低电平)	0
SYS_nWE_CON	[1]	LCD I80 系统接口 nWE 信号控制 0: 禁止(高电平)      1: 使能(低电平)	0
SCOMEN	[0]	LCD I80 系统接口命令模式使能 0: 禁止(常规模式)      1: 使能(菜单命令模式)	0

### 14.4.56. I80 系统接口菜单命令控制 1

寄存器	地址	读/写	描述	复位值
SIFCCON1	0x771001E4	读/写	I80 系统接口菜单命令数据写控制	0x0000_0000

SIFCCON1	位	描述	初始状态
SYS_WDATA	[17:0]	LCD I80 系统接口写数据控制	0

#### 14.4.57. I80 系统接口菜单命令控制 2

寄存器	地址	读/写	描述	复位值
SIFCCON2	0x771001E8	读/写	I80 系统接口菜单命令数据读控制	0x0000_0000

SIFCCON2	位	描述	初始状态
SYS_RDATA	[17:0]	LCD I80 系统接口读数据控制	0

#### 14.4.58. LCD I80 接口命令

寄存器	地址	读/写	描述	复位值
LDI_CMD0	0x77100280	读/写	I80 接口命令 0	0x0000_0000
LDI_CMD1	0x77100284	读/写	I80 接口命令 1	0x0000_0000
LDI_CMD2	0x77100288	读/写	I80 接口命令 2	0x0000_0000
LDI_CMD3	0x7710028c	读/写	I80 接口命令 3	0x0000_0000
LDI_CMD4	0x77100290	读/写	I80 接口命令 4	0x0000_0000
LDI_CMD5	0x77100294	读/写	I80 接口命令 5	0x0000_0000
LDI_CMD6	0x77100298	读/写	I80 接口命令 6	0x0000_0000
LDI_CMD7	0x7710029c	读/写	I80 接口命令 7	0x0000_0000
LDI_CMD8	0x771002A0	读/写	I80 接口命令 8	0x0000_0000
LDI_CMD9	0x771002A4	读/写	I80 接口命令 9	0x0000_0000
LDI_CMD10	0x771002A8	读/写	I80 接口命令 10	0x0000_0000
LDI_CMD11	0x771002AC	读/写	I80 接口命令 11	0x0000_0000

I80IFCONx	位	描述	初始状态
LDI_CMD	[17:0]	LDI 命令	0

#### 14.4.59. 窗口 2 的调色板数据

寄存器	地址	读/写	描述	复位值
W2PDATA01	0x77100300	读/写	索引 0,1 的窗口 2 调色板数据	0x0000_0000
W2PDATA23	0x77100304	读/写	索引 2,3 的窗口 2 调色板数据	0x0000_0000
W2PDATA45	0x77100308	读/写	索引 4,5 的窗口 2 调色板数据	0x0000_0000
W2PDATA67	0x7710030C	读/写	索引 6,7 的窗口 2 调色板数据	0x0000_0000
W2PDATA89	0x77100310	读/写	索引 8,9 的窗口 2 调色板数据	0x0000_0000
W2PDATAAB	0x77100314	读/写	索引 A,B 的窗口 2 调色板数据	0x0000_0000
W2PDATACD	0x77100318	读/写	索引 C,D 的窗口 2 调色板数据	0x0000_0000
W2PDATAEF	0x7710031C	读/写	索引 E,F 的窗口 2 调色板数据	0x0000_0000

W2PDATAxx	位	描述	初始状态
ODD_VAL	[31:16]	LUT 值寄存器	0
EVEN_VAL	[15:0]	Lut 值寄存器	0

#### 14.4.60. 窗口 3 的调色板数据

寄存器	地址	读/写	描述	复位值
W3PDATA01	0x77100320	读/写	索引 0,1 的窗口 3 调色板数据	0x0000_0000
W3PDATA23	0x77100324	读/写	索引 2,3 的窗口 3 调色板数据	0x0000_0000
W3PDATA45	0x77100328	读/写	索引 4,5 的窗口 3 调色板数据	0x0000_0000
W3PDATA67	0x7710032C	读/写	索引 6,7 的窗口 3 调色板数据	0x0000_0000
W3PDATA89	0x77100330	读/写	索引 8,9 的窗口 3 调色板数据	0x0000_0000



W3PDATAAB	0x77100334	读/写	索引 A,B 的窗口 3 调色板数据	0x0000_0000
W3PDATAACD	0x77100338	读/写	索引 C,D 的窗口 3 调色板数据	0x0000_0000
W3PDATAEF	0x7710033C	读/写	索引 E,F 的窗口 3 调色板数据	0x0000_0000

W3PDATAxx	位	描述		初始状态
ODD_VAL	[31:16]	LUT 值寄存器		0
EVEN_VAL	[15:0]	Lut 值寄存器		0

#### 14.4.61. 窗口 4 的调色板数据

寄存器	地址	读/写	描述	复位值
W2PDATA01	0x77100340	读/写	索引 0,1 的窗口 4 调色板数据	0x0000_0000
W2PDATA23	0x77100344	读/写	索引 2,3 的窗口 4 调色板数据	0x0000_0000

W4PDATAxx	位	描述		初始状态
ODD_VAL	[31:16]	LUT 值寄存器		0
EVEN_VAL	[15:0]	Lut 值寄存器		0

#### 14.4.62. WIN0 调色板 RAM 访问地址（不是 SFR）

索引	地址	读/写	描述	复位值
WIN0_PAENTRY0	0x77100400	读/写	窗口 0 调色板进入 0 地址	未定义
WIN0_PAENTRY1	0x77100404	读/写	窗口 0 调色板进入 1 地址	未定义
-	-	-	-	-
WIN0_PAENTRY255	0x771007FC	读/写	窗口 0 调色板进入 255 地址	未定义

**14.4.63. WIN1 调色板 RAM 访问地址（不是 SFR）**

索引	地址	读/写	描述	复位值
WIN1_PAENTRY0	0x77100400	读/写	窗口 1 调色板进入 0 地址	未定义
WIN1_PAENTRY1	0x77100404	读/写	窗口 1 调色板进入 1 地址	未定义
-	-	-	-	-
WIN1_PAENTRY255	0x771007FC	读/写	窗口 1 调色板进入 255 地址	未定义

## 16 TV 定标器（后处理器 VER2.5）

这节主要描述S3C6410X中TV定标器的功能和使用方法。

### 16.1 TV 定标器概述

除了FIFO的大小和输入FIFO模式，TV定标器和后处理器相似。TV定标器执行视频/图像缩放、视频格式转换和色彩空间转换。如图16-1所示，它是由数据通路、DMA控制器和寄存器文件组成。

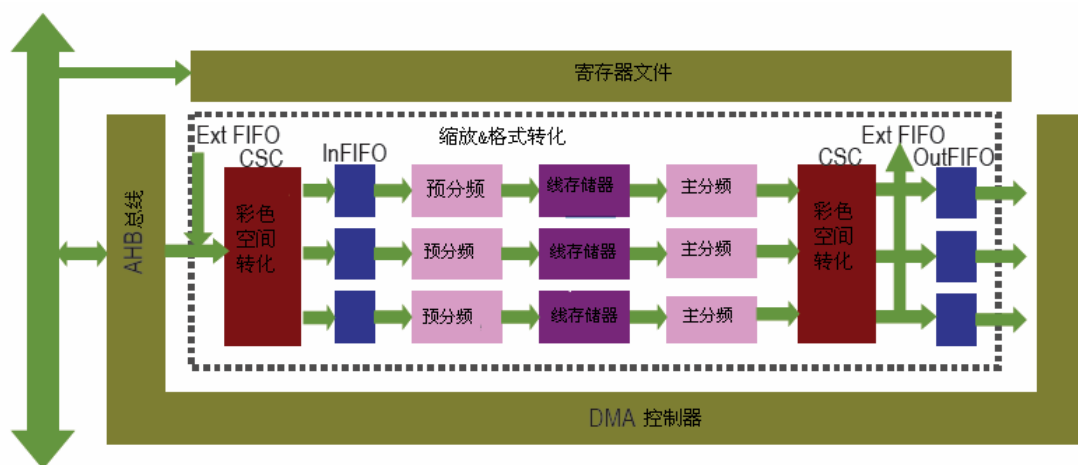


图 16-1 TV 定标器模块图

**TV定标器包含下面特性：**

- 兼容AMBA AHB v2.0的接口。
- 用于视频/图解比例尺放大/缩小或者物象像放大/缩小的3通道缩放管线。
- 视频输入格式：420、422格式。
- 图像输入格式：16位（565格式）或24位。
- 图像输出到存储器的格式：16位（565格式）/24位图形数据（仅逐行扫描）。
- 视频输出到存储器的格式：YCbCr420, YCbCr422。
- 输入到外部FIFO的格式：YCbCr444 / RGB（24位），用于隔行和逐行的扫描。
- 输出到外部FIFO的格式：YCbCr444 / RGB（30位），用于隔行和逐行的扫描。

- 自运行模式操作。
- 可设计的源图像和目标图像可达到1440x1024的分辨率。
- 可设计的缩放比率。
- 视频信号格式转换。
- 从YCbCr到RGB的色彩空间转换。
- 从RGB到YCbCr的色彩空间转换。
- 对于DMA，最大脉冲长度增加到16位。
- 带有AHB接口时钟的独立处理时钟。

## 16.2 源和目标图像数据格式

在FIMV TV定标器的顶部，有两种数据模式，DMA模式和FIFO模式，如图16-2所示。

在FIFO模式（如果LCDPathEnable = 1），目标图像传送到显示控制器中（或者其它带有FIFO接口的IP）的FIFO，没有附加的存储器带宽（比如TV定标器到存储器和存储器到显示控制器）。

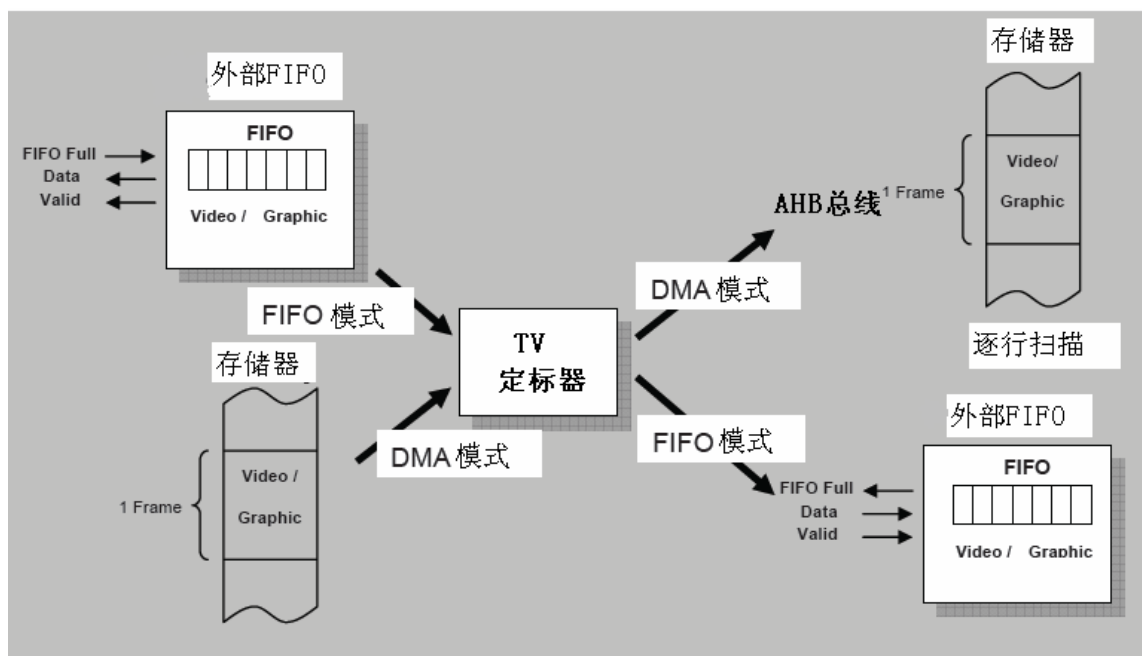


图 16-2 带有两个输入输出模式的 TV 定标器模块图

1 DMA模式操作

如表16-1a所示，通过模式配置能选择各种源和目标图像格式。源图像格式是YCbCr420、YCbCr422、RGB16位（565格式）和RGB 24位格式中的一种。目标图像格式同于源图像格式，是YCbCr420、YCbCr422、RGB 16位（565格式）和 RGB 24位格式中的一种。详细的控制信号如表16-1b所示。

表16-1a 对于视频/图像源格式和相应的数据格式的模式配置

MODE[8]	MODE[3]	MODE[2]	MODE[1]	MODE[15][0]	描述	
					视频/图像格式	图12-30和12-31中的数据格式
SRC420	InRGB	INTERLEAVE	InRGB 格式	InYCbCr 格式		
1	0	0	1	×	420 YCbCr格式	A
0	0	1	1	00/10	422YCbYCr格式	B/B’
0	0	1	1	01/11	422CbYCrY格式	C/C’
0	1	1	1	×	RGB 24位真彩色	D
0	1	1	0	×	RGB 16位格式	E

表16-1b 视频/图像目标格式和相应的数据格式的模式配置

MODE[18]	MODE[17]	MODE[20][19]	MODE[4]	MODE[15][0]	描述	
					视频/图像格式	图12-30和12-31中的数据格式
OutRGB	DST420	OutYCbCr格式	OutRGB 格式	InYCbCr 格式		
0	1	×	×	×	420 YCbCr格式	A
0	0	00/10	×	00/10	422YCbYCr格式	B/B’
0	0	01/11	×	01/11	422CbYCrY格式	C/C’
1	×	×	1	×	RGB 24位真彩色	D
1	×	×	0	×	RGB 16位格式	E

在YCbCr420源和目标图像格式的情况下，Y、Cb和Cr都存储在各自的独立地址空间，没有交叉，如图16-3（a）和图16-4的Case A所示。其它情况下，在统一的地址空间，使用字节或者半字交叉，如图16-3（b）。YCbCr422源图像的字节交叉顺序能从YCbYCr 或 CbYCrY中选择，如图16-3（b）和图16-4的case B和case C。RGB 24位的字节顺序和RGB位的半字顺序如图16-3（b）和16-4所示。

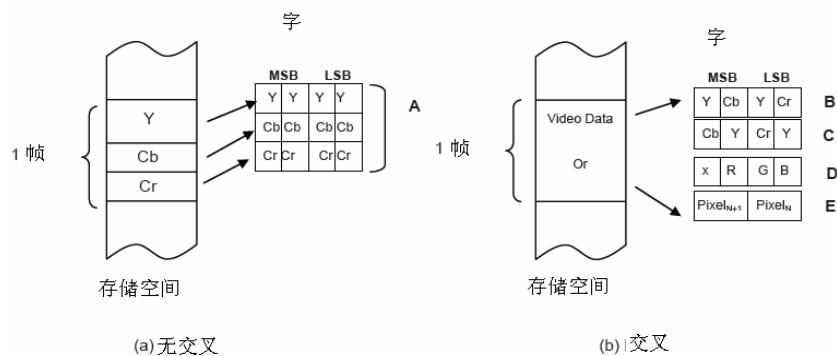


图16-3 数据格式外部存储

YCbCr420和YCbCr422源图像格式下，是选择MPEG4格式还是选择MPEG2/H.263格式，由色度信息的采样位置决定，如图16-4所示。当源和目标图像是YCbCr格式，目标图像和源图像有相同的采样位置。如果源图像是RGB格式，目标图像是YCbCr格式，色度信息的采样位置和亮度位置相同。

所有的源图像和目标图像数据必须存储在存储器系统，以字边界对齐。意思是既不支持字节也不支持半字大小的DMA操作。因此，源和目标图像的宽度必须选择满足字边界大小的。

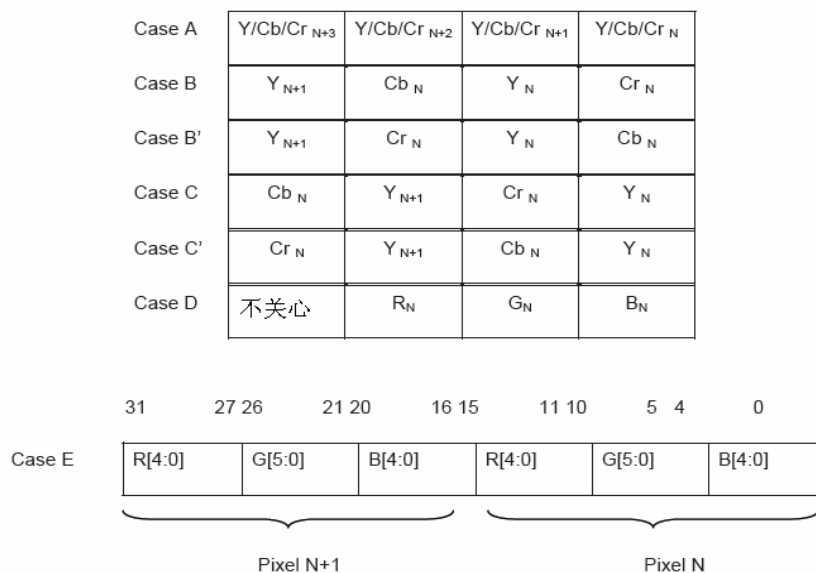


图16-4 字节或半字组织模块图

## 2. FIFO 模式操作

输出数据格式由MODE[18]决定，如表16-1b所示。数据格式被固定到30位数据，每十位构成RGB 或

YCbCr444。如果ExtFIFOIn或LCDPathEnable设置为“1”，表16-1 b中其它特殊模式配置信号被忽略。FIFO模式中，输入输出数据格式如表16-2所示。

表16-2 FIFO模式中，输入输出数据格式

OutRGB (MODE[18])	输出数据格式 (LCDPathEnable = 1) (隔行/逐行)
0	YCbCr 444
1	RGB 30位

输出FIFO模式中，无论是隔行扫描模式还是逐行扫描模式，都是通过“隔行”控制寄存器进行选择的。只有当LCDPathEnable =1 时，“隔行”控制位有效，否则支持逐行扫描模式。

隔行扫描模式有效 (LCDPathEnable = 1和Interlace = 1)，由奇数场和偶数场组成的帧管理自动运行。因此，该帧管理对于隔行扫描和逐行扫描模式是一样的。它也兼容FIMV TV定标器。

16.3 图像尺寸和比率

RGB图像源大小由沿水平方向和垂直方向上的像素数决定。YCbCr420 和 YCbCr422 源图像大小只由Y采样沿水平和垂直方向决定。目标图像大小由最终RGB图解的图像尺度决定。如果源图像是YCbCr 图像，经色度空间转换后，RGB图解的图像最终尺度定下来。

像前面所说的，SRC\_Width和DST\_Width受到字边界的限制，所以水平像素数能用kn表示，其中n=1、2、3、...，对于24bppRGB /16bppRGB / YCbCr420图像,k分别为1 / 2 / 8。而且SRC\_Width必须是PreScale\_H\_Ratio的4倍，并且SRC\_Height必须是PreScale\_V\_Ratio的倍数。源图像和目标图像模块图如图16-5所示。

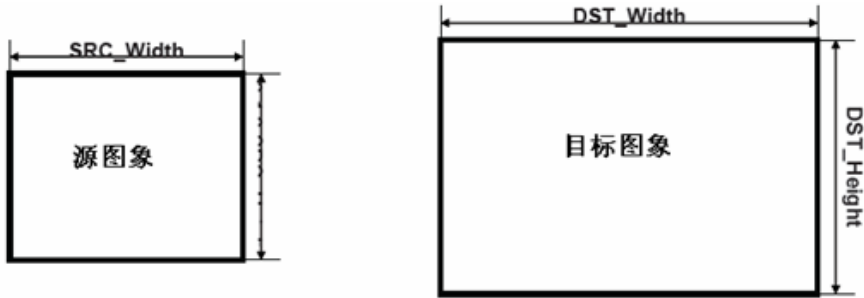


图16-5 源和目标图像大小模块图

其它的控制寄存器，预缩放图像大小、预缩放比率、预缩放移位比率和主要的预缩放比率由下面的等

式定义：

```
if ( SRC_Width >= 64 × DST_Width ) { Exit(-1); /*超出水平比例尺范围 */ }
else if (SRC_Width >= 32 × DST_Width) { PreScale_H_Ratio = 32; H_Shift = 5; }
else if (SRC_Width >= 16 × DST_Width) { PreScale_H_Ratio = 16; H_Shift = 4; }
else if (SRC_Width >= 8 × DST_Width) { PreScale_H_Ratio = 8; H_Shift = 3; }
else if (SRC_Width >= 4 × DST_Width) { PreScale_H_Ratio = 4; H_Shift = 2; }
else if (SRC_Width >= 2 × DST_Width) { PreScale_H_Ratio = 2; H_Shift = 1; }
else { PreScale_H_Ratio = 1; H_Shift = 0; }

PreScale_DSTWidth = SRC_Width / PreScale_H_Ratio;
dx = ( SRC_Width << 8 ) / ( DST_Width << H_Shift);

If ( SRC_Height >= 64 × DST_Height ) { Exit(-1); /* 超出垂直比例尺范围 */ }
else if (SRC_Height >= 32 × DST_Height) { PreScale_V_Ratio = 32; V_Shift = 5; }
else if (SRC_Height >= 16 × DST_Height) { PreScale_V_Ratio = 16; V_Shift = 4; }
else if (SRC_Height >= 8 × DST_Height) { PreScale_V_Ratio = 8; V_Shift = 3; }
else if (SRC_Height >= 4 × DST_Height) { PreScale_V_Ratio = 4; V_Shift = 2; }
else if (SRC_Height >= 2 × DST_Height) { PreScale_V_Ratio = 2; V_Shift = 1; }
else { PreScale_V_Ratio = 1; V_Shift = 0; }

PreScale_DSTHeight = SRC_Height / PreScale_V_Ratio;
dy = ( SRC_Height << 8 ) / ( DST_Height << V_Shift);
PreScale_SHFactor = 10 - ( H_Shift + V_Shift);
```

## 16.4 源和目标图像的 DMA 操作

DMA操作有三种地址，即起始地址、结束地址和偏移地址。每个地址由Y/Cb/Cr三个源地址和RGB/oCb/oCr三个目标地址组成。如果一个源图像以非交叉格式（像YCbCr420）存储，所有的源地址分量是有效的，如图16-6(a)所示。如果一个源图像以交叉格式（像RGB图像格式或YCbCr422格式）存储，三个



源分量中只有Y是有效的，两个色度地址分量是无效的，如图16-6 (b)所示。如果目标图像以非交叉格式（像YCbCr420）存储，所有的源地址分量RGB/oCb/oCr是有效的，如图16-6 (a)所示。如果一个源图像以交叉格式(像RGB图像格式或YCbCr422格式)存储，三个源分量中只有RGB是有效的， oCb/oCr两个色度地址分量是无效的，如图16-6 (b)所示。

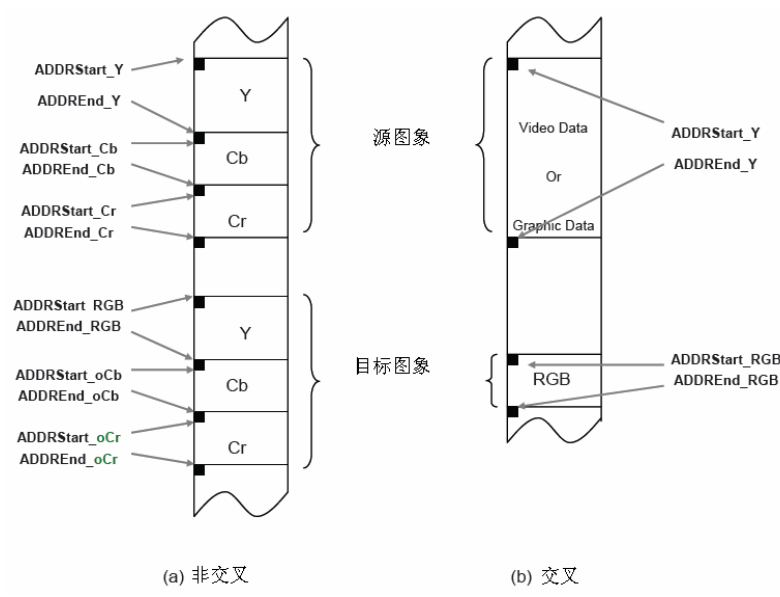


图16-6 根据存储起配置类型，起始和结束地址的设置

1. 起始地址

ADDRStart\_Y/Cb/Cr/RGB/oCb/oCr 的起始地址指向第一个字的地址，Y/Cb/Cr/RGB/oCb/oCr 相应的分量被读取或者写入该地址。每一个必须以字(例如: ADDRStart\_X[1:0] = 00)对齐。只有 YCbCr420 源图像格式，ADDRStart\_Cb 和 ADDRStart\_Cr 是有效的。只有对于 YCbCr420 目标图像格式，ADDRStart\_oCb 和 ADDRStart\_oCr 是有效的。

2. 结束地址

ADDREnd\_Y

= ADDRStart\_Y + 存储器大小 (对于 Y/RGB 分量)

= ADDRStart\_Y + (SRC\_Width × SRC\_Height) × ByteSize\_Per\_Pixel + Offset\_Y × (SRC\_Height-1)

$\text{ADDREnd\_Cb}$  (对于 YCbCr420 源格式有效的)  
 $= \text{ADDRStart\_Cb} + \text{存储器大小 (Cb 分量)}$   
 $= \text{ADDRStart\_Cb} + (\text{SRC\_Width}/2 \times \text{SRC\_Height}/2) \times \text{ByteSize\_Per\_Pixel} + \text{Offset\_Cb} \times (\text{SRC\_Height}/2-1)$

$\text{ADDREnd\_Cr}$  (对于 YCbCr420 源格式有效的)  
 $= \text{ADDRStart\_Cr} + \text{存储器大小 (Cr 分量)}$   
 $= \text{ADDRStart\_Cr} + (\text{SRC\_Width}/2 \times \text{SRC\_Height}/2) \times \text{ByteSize\_Per\_Pixel} + \text{Offset\_Cr} \times (\text{SRC\_Height}/2-1)$

$\text{ADDREnd\_RGB}$   
 $= \text{ADDRStart\_RGB} + \text{存储器大小 (RGB 数据或 Y 分量)}$   
 $= \text{ADDRStart\_RGB} + (\text{DST\_Width} \times \text{DST\_Height}) \times \text{ByteSize\_Per\_Pixel} + \text{Offset\_RGB} \times (\text{DST\_Height}-1)$

$\text{ADDREnd\_oCb}$  (对于 YCbCr420 目标格式有效的)  
 $= \text{ADDRStart\_oCb} + \text{存储器大小 (Cb 分量)}$   
 $= \text{ADDRStart\_oCb} + (\text{SRC\_Width}/2 \times \text{SRC\_Height}/2) \times \text{ByteSize\_Per\_Pixel} + \text{Offset\_Cb} \times (\text{SRC\_Height}/2-1)$

$\text{ADDREnd\_oCr}$  (有效的对于 YCbCr420 目标格式)  
 $= \text{ADDRStart\_oCr} + \text{存储器大小 (Cr 分量) for the component of}$   
 $= \text{ADDRStart\_oCr} + (\text{SRC\_Width}/2 \times \text{SRC\_Height}/2) \times \text{ByteSize\_Per\_Pixel} + \text{Offset\_Cr} \times (\text{SRC\_Height}/2-1)$

其中

$\text{Offset\_Y/Cb/Cr/RGB}$

$= \text{存储器大小 (每一条水平线偏移量)}$

$= \text{水平偏移的像素数(采样)} \times \text{ByteSize\_Per\_Pixel (或采样)}$

ByteSize\_Per\_Pixel = 1 （对于 YCbCr420）

2 （对于 16 位 RGB 和 YcbCr422）

4 （对于 24 位 RGB）

偏移地址用于以下两种情况：一种是为了放大/缩小物象，截取源图像的一些部分，如图 16-7（a）所示；另一种是为了 PIP（picture-in-picture）应用，重新存储目标图像，如图 16-7（b）所示。两种情况都必须满足字边界限制。

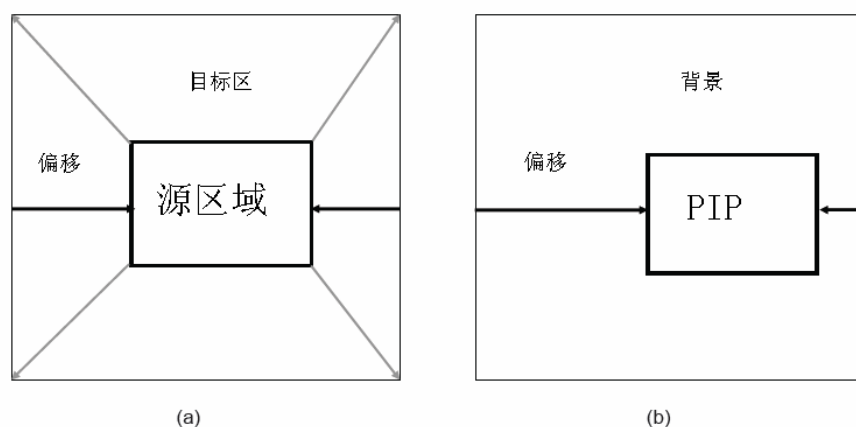


图16-7 (a)源图像放大/缩小操作的偏移地址和(b) PIP应用的目标图像

## 16.5 TV 定标器的帧管理

### 1. 每帧的管理模式

TV定标器的每帧处理被两个控制寄存器控制，像POSTENVID 和POSTINT，如图16-8所示。“POSTENVID”触发TV定标器的操作。当给予的帧完成所有操作，则它自动无效。在“POSTENVID”有效之前，所有控制寄存器必须设置为合适的值。当操作完成，中断等待寄存器有效（POSTINT=1）。如果中断有效信号有效（INTEN=1），POSTINT信号（指向中断控制器），必须通过中断服务程序清除。轮询POSTENVID也用于检测操作的结束。

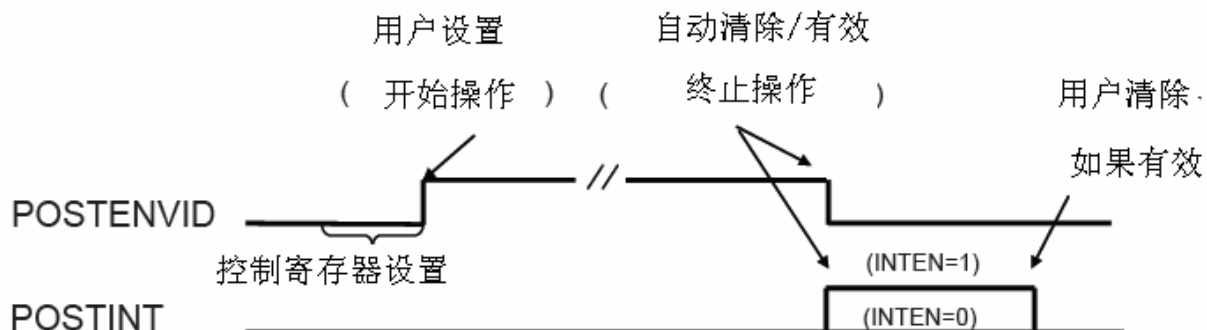


图16-8 TV定标器操作的开始和终止 (AutoLoadEnable = 0)

## 2. 自由运行模式

为了激活新的帧管理方式，即自运行操作，必须设置“AutoLoadEnable”为1。在该模式中，用户能预定义下一相关的帧NxtADDRXXX的地址设置。

当前帧完成时，下面的操作一步步执行：

- (1) 根据INTEN，判断中断信号是否有效；
- (2) NxtADDRXXX的下一帧地址设置，拷贝到ADDRXXX 的当前帧地址设置；
- (3) ENVID 自动有效，并且下一帧的操作开始。

在自运行模式下，除非地址为满，否则其它寄存器的值在当前帧和下一帧必须保持相同的值。

## 16.6 定标器（DMA 到 FIFO）代码实现

下面是 ARM11 中，DMA 到 FIFO 部分代码的具体实现，结合上面对定标器的说明，理解以下代码：

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
DMA to FIFO
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

static void ScalerT_SetCscTypeDmaToFifo(void)
{
    int nSelSrcDataFmt;
    while (1)

```

```

{
    UART_Printf("\n");
    UART_Printf("[1] RGB16\n");
    UART_Printf("[2] RGB24\n");
    UART_Printf("[3] YCbYCr422 Interleave\n");
    UART_Printf("[4] YCrYCb422 Interleave\n");
    UART_Printf("[5] CbYCrY422 Interleave\n");
    UART_Printf("[6] CrYCbY422 Interleave\n");
    UART_Printf("[7] YUV 420 Non-Interleave\n");
    UART_Printf(">> Enter Source Data Format: ");

    nSelSrcDataFmt = UART_GetIntNum();

    if (nSelSrcDataFmt >= 1 && nSelSrcDataFmt <= 7)
    {
        eSrcDataFmt =
            (nSelSrcDataFmt == 1) ? RGB16 :
            (nSelSrcDataFmt == 2) ? RGB24 :
            (nSelSrcDataFmt == 3) ? YCBYCR :
            (nSelSrcDataFmt == 4) ? YCRYCB :
            (nSelSrcDataFmt == 5) ? CBYCRY :
            (nSelSrcDataFmt == 6) ? CRYCBY : YC420;

        break;
    }
    else
        UART_Printf("Invalid Data Format! Retry It!!\n");
}

#if 0 // jihyun
while (true)

```

```

{
    UART_Printf("\n");
    UART_Printf("[0] Exit\n");
    UART_Printf("[1] YUV444 (Y:10, CB:10 CR: 10) Interleave\n");
    UART_Printf("[2] RGB30 (R:10 G:10 B:10)\n");
    UART_Printf(">> Enter Destination Data Format: ");

    nSelDstDataFmt = UART_GetIntNum();

    if (nSelDstDataFmt == 0)
        return;
    else if (nSelDstDataFmt >= 1 && nSelDstDataFmt <= 2)
    {
        ePostFifoIf = (nSelDstDataFmt == 1) ? YUV : RGB;
        break;
    }
    else
        UART_Printf("Invalid Data Format! Retry It!!\n");
}
#endif

if (eSrcDataFmt == RGB16)
{
    uImgHSz = 240, uImgVSz = 320;
    //H-Size : Scale down(280->240), V-Size : Scale down(360->320)
    uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 40, uMidStartY = 80;
    uMidScaledHSz = 280, uMidScaledVSz = 360;
}
else if (eSrcDataFmt == RGB24)
{

```

```

    uImgHSz = 240, uImgVSz = 320;

    //H-Size : Scale up(200->240),    V-Size : Scale up(240->320)

    uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 40, uMidStartY = 80;
    uMidScaledHSz = 200, uMidScaledVSz = 240;
}

else if (eSrcDataFmt == YCBYCR)
{
    uImgHSz = 240, uImgVSz = 320;

    //H-Size : Scale up(200->240),    V-Size : Scale down(360->320)

    uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 160, uMidStartY = 120;
    uMidScaledHSz = 200, uMidScaledVSz = 360;
}

else if (eSrcDataFmt == YCRYCB)
{
    uImgHSz = 240, uImgVSz = 320;

    //H-Size : Scale down(280->240),    V-Size : Scale up(280->320)

    uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 0, uMidStartY = 0;
    uMidScaledHSz = 280, uMidScaledVSz = 280;
}

else if (eSrcDataFmt == CBYCRY)
{
    uImgHSz = 240, uImgVSz = 320;

    //H-Size : Scale up(200->240),    V-Size : Scale up(280->320)

    uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 40, uMidStartY = 80;
    uMidScaledHSz = 200, uMidScaledVSz = 280;
}

else if (eSrcDataFmt == CRYCBY)
{
    uImgHSz = 240, uImgVSz = 320;

```

```

//H-Size : Scale down(320->240),    V-Size : Scale down(440->320)

uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 40, uMidStartY = 40;

uMidScaledHSz = 320, uMidScaledVSz = 440;

}

else if (eSrcDataFmt == YC420)

{

    uImgHSz = 240, uImgVSz = 320;

    //H-Size : Scale up(200->240),    V-Size : Scale down(400->320)

    uMidImgHSz = 360, uMidImgVSz = 480, uMidStartX = 40, uMidStartY = 0;

    uMidScaledHSz = 320, uMidScaledVSz = 400;

}

UART_Printf("\n");

if(uSimpleTest == 0)

{

    UART_Printf("=====    Complex_Test Image Size    =====\n");

    UART_Printf("SrcImgHSz    = %d,          SrcImgVSz    = %d\n",uMidImgHSz,uMidImgVSz);

    UART_Printf("SrcStartX    = %d,          SrcStartY    = %d\n",uMidStartX,uMidStartY);

    UART_Printf("SrcCroppedHSz = %d,          SrcCroppedVSz = %d\n",uMidScaledHSz,uMidScaledVSz);

    UART_Printf("=====\\n");

    UART_Printf("\n");

}

}

```



# 16.7 寄存器文件列表

下面是对各个寄存器的具体介绍。

## 16.7.1. 模式控制寄存器

寄存器	地址	读/写	描述	复位值
MODE	0X76300000	读/写	模式寄存器（[31:0]）。	0x00070B12

MODE	位	描述	初始状态
ExtFIFOIn	[31]	输入 FIFO 模式有效。1对应 FIFO 模式， 0 对应 DMA 模式。	0
CLKVALUP	[30]	CLKVAL_F 更新定时控制。  0 = 一直  1 = 一 帧的开始(一帧只有一次)	0
CLKVAL_F	[29:24]	确定TSCLK 和CLKVAL[5:0]的比率。  TSCLK = 时钟源 / (CLKVAL+1) 其中CLKVAL >= 1  注：VCLK 的最大频率是66MHz 。	0
CLKDIR	[23]	选择指定或者用CLKVAL_F 寄存器划分时钟源。  0 =指定的时钟(TSCLK的频率 = 时钟源频率)  1 = 用CLKVAL_F划分	0
CLKSEL_F	[22:21]	选择视频时钟源。  00 = HCLK  01 = PLL Ext 时钟输入  10 =保留  11 = 27MHz Ext 时钟输出	0
OutYCbCrFormat	[20:19]	当目标图像是交叉的YCbCr 格式，它决定字数据的字节组织。更多信息参考图6-12(b)。	0

OutRGB	[18]	它表示目标图像的输出色彩空间。0对应 YCbCr 或1 对应 RGB。	1
DST420	[17]	0 对应YCbCr422 ， 1 对应 YCbCr420 目标格式。它只对于YCbCr目标图像是有效的（也就是 OutRGB = 0）。	1
R2YSel	[16]	从RGB 到YCbCr选择色彩空间转变等式。1 对应 YCbCr 宽度范围， 0 对应 YCbCr 窄度范围。	1
InYCbCrFormat_ MSB	[15]	当源图像是交叉YCbCr格式，它决定字数据字节组织的MSB。	0
AutoLoadEnable	[14]	自动载入有效。 0 对应逐帧模式， 1 对应自运行模式。	0
LCDPathEnable	[13]	输出FIFO模式有效 。 1 对应 FIFO 模式 ， 0 对应DMA模式。	0
Interlace	[12]	只用在FIFO模式(LCDPathEnable =1)下的，输出浏览方法选择寄存器。 1对应隔行扫描， 0 对应逐行扫描。 在 DMA 模式(LCDPathEnable = 0)下，不论给予何值都执行逐行扫描。	0
Wide/Narrow	[11:10]	根据输入值的范围选择从YCbCr 到RGB 色彩空间转变等式。 2' 10 对应YCbCr 宽度范围， 2' 01 对应YCbCr 窄度范围。	2' b10
SRC420	[8]	0 对应YCbCr422 和1 对应 YCbCr420 源格式。它只有对于 YCbCr 源图像(也就是InRGB = 0) 是有效的。	1
INTEN	[7]	中断有效。当当前帧的处理完成时，由它决定 POSTINT 信号是否有效，。 0:无效 1:有效	0

POSTINT	[6]	中断等待位。如果INTEN有效，当当前帧完成操作，则自动声明正确。它必须被中断服务程序清除。 0: 无效， 1: 有效	0
IRQ_LEVEL	[5]	它决定电平产生方案。1对应电平中断(必须是1)。	0
OutRGBFormat	[4]	它决定目标图像的输出格式。 0对应16位(565格式) RGB 和1对应24位RGB。	1
InRGB	[3]	它指示源图像的输入色彩空间。 0 对应 YCbCr 或 1对应 RGB。	0
INTERLEAVE	[2]	它指示YCbC的数据格式。0 对应非交叉格式(Y、Cb和Cr分量以字存取)。 1 对应交叉格式(Y、Cb和 Cr 混合在单个字内)。当源图像是RGB 数据(或 InRGB =1)时，它应当是1。 。	0
InRGBFormat	[1]	如果源图像在RGB 色彩空间(或InRGB=1)，它表示图解图像的数据格式。0对应16位 (565 格式) 和1 对应24位。 否则(或InRGB=0)，它应当保持为1。	1
InYCbCrFormat_LSB	[0]	当源图像是交叉YCbCr时，它决定字数据字节组织的LSB。	0

注意：[9]位为保留位

内部时钟设计模块图如图16-9所示。

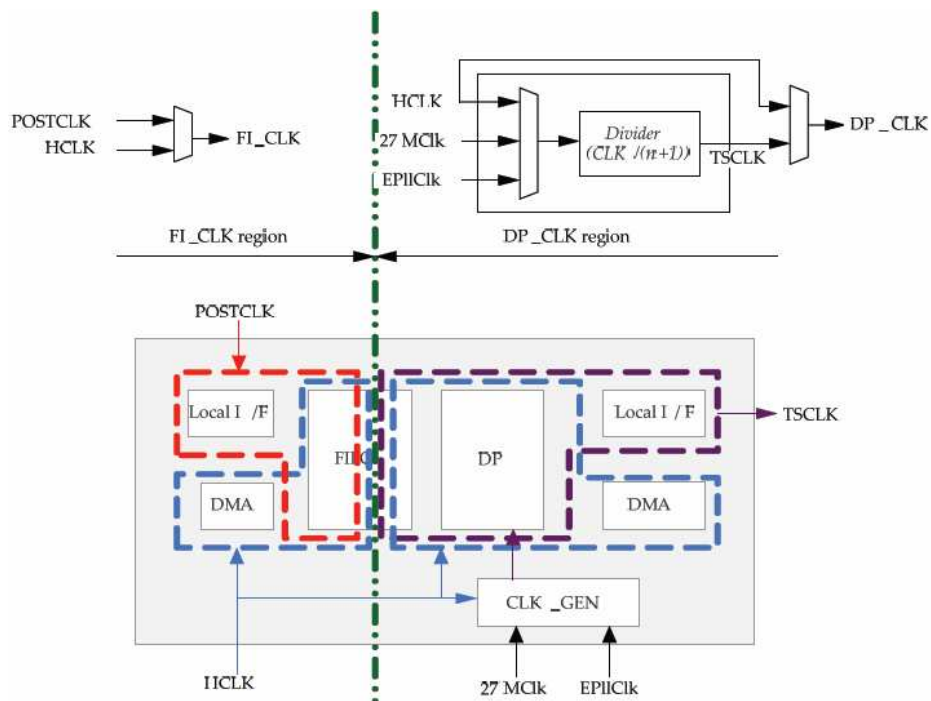


图16-9 内部时钟设计模块图

### 16.7.2. 预缩放比率寄存器

寄存器	地址	读/写	描述	复位值
PreScale_Ratio	0X76300004	读/写	用于垂直和水平方向预缩放比率。	0x0

PreScale_Ratio	位	描述	初始状态
PreScale_V_Ratio	[13:7]	预缩放比率，用于垂直方向。	0x0
PreScale_H_Ratio	[6:0]	预缩放比率，用于水平方向。	0x0

### 16.7.3. 预缩放图像大小寄存器

寄存器	地址	读/写	描述	复位值
PreScaleImgSize	0X76300008	读/写	预缩放图像大小。	0x0

PreScaleImgSize	位	描述	初始状态
PreScale_DSTHeight	[23:12]	预缩放图像高度。	0x0
PreScale_DSTWidth	[11:0]	预缩放图像宽度。	0x0

#### 16.7.4. 源图像大小寄存器

寄存器	地址	读/写	描述	复位值
SRCImgSize	0X7630000C	读/写	源图像大小。	0x0

SRCImgSize	位	描述	初始状态
SRCHeight	[23:12]	源图像高度。	0x0
SRCWidth	[11:0]	源图像宽度。	0x0

#### 16.7.5. 水平主缩放比率寄存器

寄存器	地址	读/写	描述	复位值
MainScale_H_Ratio	0X76300010	读/写	水平方向主缩放比率。	0x0

SRC_Width	位	描述	初始状态
MainScale_H_Ratio	[8:0]	用于水平方向主缩放比率。	0x0

#### 16.7.6. 垂直主缩放比率寄存器

寄存器	地址	读/写	描述	复位值
MainScale_V_Ratio	0X76300014	读/写	垂直方向主缩放比率。	0x0

SRC_Width	位	描述	初始状态
MainScale_V_Ratio	[8:0]	垂直方向主缩放比率。	0x0

16.7.7. 目标图像大小寄存器

寄存器	地址	读/写	描述	复位值
DSTImgSize	0X76300018	读/写	目标图像大小。	0x0

SRCImgSize	位	状态	初始状态
DSTHeight	[23:12]	目标图像高度。	0x0
DSTWidth	[11:0]	目标图像宽度。	0x0

16.7.8. 预缩放移位因子寄存器

寄存器	地址	读/写	描述	复位值
PreScale_SHFactor	0X7630001C	读/写	预缩放移位因子。	0x0

SRC_Width	位	描述	初始状态
PreScale_SHFactor	[3:0]	预缩放移位因子。	0x0

16.7.9. 对于 SW 触发器模式，DMA 起始地址

寄存器	地址	读/写	位	描述	复位值
ADDRStart_Y	0X76300020	读/写	[30:0]	用于源Y或者RGB分量的DMA（缓冲区0）的起始地址。	0x2000_0000

寄存器	地址	读/写	位	描述	复位值
ADDRStart_Cb	0X76300024	读/写	[30:0]	用于源Cb分量的DMA（缓冲区0）的起始地址。	0x2000_0000

寄存器	地址	读/写	位	描述	复位值
ADDRStart_Cr	0X76300028	读/写	[30:0]	用于源Cr分量的DMA（缓冲区0）的起始地址。	0x2000_0000

寄存器	地址	读/写	位	描述	复位值
ADDRStart_RGB	0X7630002C	读/写	[30:0]	用于目标Y或者RGB分量的DMA（缓冲区0）的起始地址。	0x2000_0000

#### 16.7.10. 对于 SW 触发器模式，DMA 结束地址寄存器

寄存器	地址	读/写	位	描述	复位值
ADDREnd_Y	0X76300030	读/写	[30:0]	用于源Y或RGB分量的DMA（缓冲区0）的结束地址。	0x2000_6300

寄存器	地址	读/写	位	描述	复位值
ADDREnd_Cb	0X76300034	读/写	[30:0]	用于源Cb分量的DMA（缓冲区0）的结束地址。	0x2000_6300

寄存器	地址	读/写	位	描述	复位值
ADDREnd_Cr	0X76300038	读/写	[30:0]	用于源Cr分量的DMA（缓冲区0）的结束地址。	0x2000_6300

寄存器	地址	读/写	位	描述	复位值
ADDREnd_RGB	0X7630003C	读/写	[30:0]	用于目标Y或RGB分量的DMA（缓冲区0）的结束地址。	0x2000_6300

16.7.11. 当前帧（缓冲区 0）和下一帧（缓冲区 1）偏移寄存器

寄存器	地址	读/写	位	描述	复位值
Offset_Y	0X76300040	读/写	[23:0]	用于截取源图像的Y和RGB分量的偏移。	0

寄存器	地址	读/写	位	描述	复位值
Offset_Cb	0X76300044	读/写	[23:0]	用于截取源图像的Cb分量的偏移。	0

寄存器	地址	读/写	位	描述	复位值
Offset_Cr	0X76300048	读/写	[23:0]	用于截取源图像的Cr分量的偏移。	0

寄存器	地址	读/写	位	描述	复位值
Offset_RG	0X7630004C	读/写	[23:0]	用于重新存储目标图像的Y和RGB分量的偏移。	0

注：0X76300050被保留

16.7.12. 对于 SW 触发器模式，下一帧 DMA 起始地址

寄存器	地址	读/写	位	描述	复位值
NxtADDRStart_Y	0X76300054	读/写	[30:0]	用于源Y或者RGB分量的下一帧DMA（缓冲区1）的起始地址。	0x2000_0000

寄存器	地址	读/写	位	描述	复位值
NxtADDRStart_Cb	0X76300058	读/写	[30:0]	用于源Cb分量的下一帧DMA（缓冲区1）的起始地址。	0x2000_0000



寄存器	地址	读/写	位	描述	复位值
NxtADDRStart_ Cr	0X7630005C	读/写	[30:0]	用于源Cr分量的下一帧DMA（缓冲区1）的起始地址。	0x2000_0000

寄存器	地址	读/写	位	描述	复位值
NxtADDRstart_ RGB	0X76300060	读/写	[30:0]	用于目标Y或者RGB分量的下一帧DMA（缓冲区1）的起始地址。	0x2000_0000

#### 16.7.13. 对于 SW 触发器模式，下一帧 DMA 结束地址寄存器

寄存器	地址	读/写	位	描述	复位值
NxtADDREnd_Y	0X76300064	读/写	[30:0]	用于源Y或RGB分量的下一帧DMA（缓冲区1）的结束地址。	0x2000_6300

寄存器	地址	读/写	位	描述	复位值
NxtADDREnd_Cb	0X76300068	读/写	[30:0]	对于源Cb分量的下一帧DMA（缓冲区1）的结束地址。	0x2000_6300

寄存器	地址	读/写	位	描述	复位值
NxtADDREnd_Cr	0X7630006C	读/写	[30:0]	用于源Cr分量，下一帧DMA（缓冲区1）的结束地址。	0x2000_6300

寄存器	地址	读/写	位	描述	复位值
NxtADDREnd_R GB	0X76300070	读/写	[30:0]	用于目标Y或RGB分量的下一帧DMA（缓冲区1）结束地址。	0x2000_6300

#### 16. 7. 14. 对于输出 Cb 和 Cr, DMA 起始地址寄存器

寄存器	地址	读/写	位	描述	复位值
ADDRStart_oCb	0X76300074	读/写	[30:0]	用于目标Cb分量的DMA（缓冲区0）的起始地址。	0x2000_0000

寄存器	地址	读/写	位	描述	复位值
ADDRStart_oCr	0X76300078	读/写	[30:0]	用于目标Cr分量的DMA（缓冲区0）的起始地址。	0x2000_0000

#### 16. 7. 15. 对于输出 Cb 和 Cr, DMA 结束地址寄存器

寄存器	地址	读/写	位	描述	复位值
ADDREnd_oCb	0X7630007C	读/写	[30:0]	用于目标Cb分量的DMA（缓冲区0）的结束地址。	0x2000_0000

寄存器	地址	读/写	位	描述	复位值
ADDREnd_oCr	0X76300080	读/写	[30:0]	用于目标Cr分量的DMA（缓冲区0）的结束地址。	0x2000_0000

#### 16. 7. 16. 对于输出 Cb 和 Cr, 当前帧（缓冲区 0）和下一帧（缓冲区 1）偏移寄存器

寄存器	地址	读/写	位	描述	复位值
Offset_oCb	0X76300084	读/写	[23:0]	用于截取目标图像的Cb分量的偏移量。	0

寄存器	地址	读/写	位	描述	复位值
Offset_oCr	0X76300088	读/写	[23:0]	用于截取目标图像的Cr分量的偏移量。	0

#### 16. 7. 17. 对于输出 Cb 和 Cr, 下一帧 DMA 起始地址寄存器

寄存器	地址	读/写	位	描述	复位值
NxtADDRStart_oCb	0X7630008C	读/写	[30:0]	用于目标Cb分量的下一帧DMA（缓冲区1）的起始地址。	0x2000_6300

寄存器	地址	读/写	位	描述	复位值
NxtADDRStart_oCr	0X76300090	读/写	[30:0]	用于目标Cr分量的下一帧DMA（缓冲区1）的起始地址。	0x2000_6300

#### 16. 7. 18. 对于输出 Cb 和 Cr, 下一帧 DMA 结束地址寄存器

寄存器	地址	读/写	位	描述	复位值
NxtADDREnd_oCb	0X76300094	读/写	[30:0]	用于目标Cb分量的下一帧DMA（缓冲区1）的结束地址。	0x2000_6300

寄存器	地址	读/写	位	描述	复位值
NxtADDREnd_oCr	0X76300098	读/写	[30:0]	用于目标Cr分量的下一帧DMA（缓冲区1）的结束地址。	0x2000_6300

#### 16. 7. 19. 用于启动视频处理, POSTENVID 寄存器

寄存器	地址	读/写	位	描述	复位值
POSTENVID	0X7630009C	读/写	[31]	启动视频处理, 打开TV定标器。当前帧操作完成后, 自动禁止。在控制寄存器配置状态下, 应当被禁止 (POSTENVID=0)。在操作过程中, 禁止禁止。但是对于操作TV定标器是只读的情况下, 可以禁止。	0

### 16. 7. 20. 模式控制寄存器 2

寄存器	地址	读/写	描述	复位值
MODE_2	0X763000A0	读/写	模式寄存器2。	0x0

MODE_2	位	描述	初始状态
FIFO_OUT_PA TH	TH [6:5]	FIFO输出路径选择。  00 = TV 编码器输出  01 = FIMD WIN1  1x = FIMD WIN2	00
ADDR_CH_DIS	[4]	自运行模式下，下一个地址改变无效（软件触发器模式）。  0= 地址改变有效  1 = 地址改变无效	0
BC_SEL	[3]	DMA地址改变选择（软件触发器模式）。  0 = EVEN/ODD FIELD 结束，地址改变  1 = FRAME 结束，地址改变	0
Reserved	[2:1]	必须设置为0	0
TRG_MODE	[0]	选择启动DMA处理模式。  0 :软件触发器模式  1 :保留	0

## 17 TV 编码器

该节主要讲述用于 S3C6410X RSIC 处理器的 TV 编码器。

### 17.1 TV 编码器的概述

TV 编码器用于将数字视频数据转换为复合模拟视频信号。S3C6410X 的 TV 编码器有一些特殊的特性。首先，它有影像增强引擎，使图像质量通过特殊功能增强。第二，它支持不同大小的图像显示，有全屏、宽屏和原始三种模式。它也能同时显示 TV 输出和 LCD 两种不同的图像。S3C6410X 的 TV 编码器支持模拟复合输出和 S-video 输出。

#### 1. TV编码器具有下面的特性：

- 在MIE（移动图像增强器）装置建立。
- 黑色和白色信号展宽。
- 蓝色信号展宽和Flesh-Tone修正。
- 动态的水平尖脉冲和LTI。
- 降低黑噪音和白噪音。
- 对比度、锐度、伽玛射线和亮度控制。
- 用LCD显示不同的或者相同的电子束。
- 原始、全屏和宽屏大小视频输出。

#### 2. 模块图

TV编码器器模块图如图17-1所示。

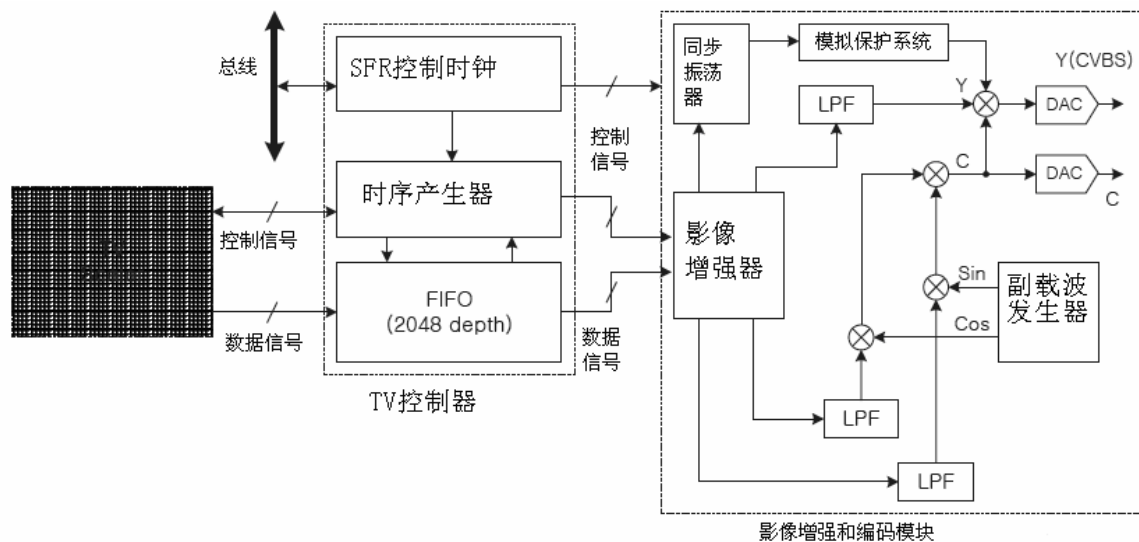


图 17-1 TV 编码器模块图

## 17.2 功能描述

TV编码器将数字的像素数据编码成ITU-R BT. 656格式，其主要包含四部分。

第一部分是增强器和编码器模块。增强器和编码器模块包含编码器和影像增强器。该影像增强器通过控制黑色和白色信号展宽、伽玛射线、亮度、对比度等来管理影像增强，使我们能得到效果加强的图像。编码器产生ITU-R BT. 656格式的TV信号。

### 数据通路

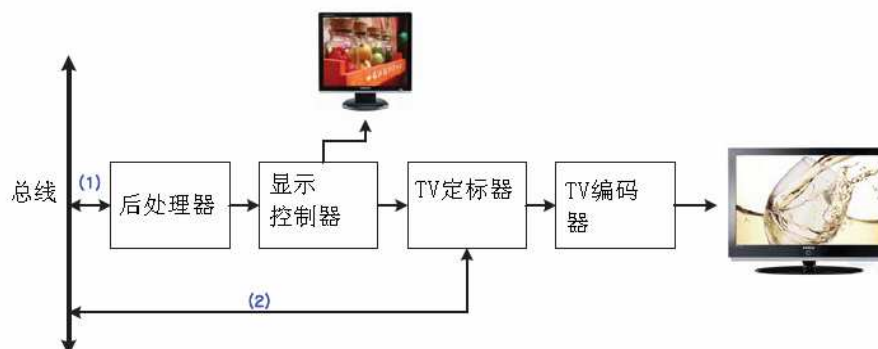


图 17-2 TV 编码器数据通路概念

### （后处理器→）显示控制器→TV定标器→TV编码器，通路（1）

如图17-2所示，数据通过后处理器或者显示控制器直接从存储器载入。显示控制器将数据发送到TV定标器之后，适当地缩放图像。最后，TV定标器将数据发送到TV编码器的图像缓冲区。同时，该通路能在LCD和TV编码器中显示相同的图像。

### TV定标器→TV编码器，通路（2）

如图17-2中，数据通过TV定标器载入，它将数据转换成合适的大小和色彩空间，然后将图像发送到TV编码器。该通路能在LCD和TV编码器之间显示不同的图像。

## 1.复合模拟信号的合成

如图17-3所示，显示水平时序。在X轴方向上，TV合成输出，它分成3个时序部分：后沿、活动、前沿。后沿和前沿是同步信号。活动区包含有效数据。

在TV输出的Y轴方向上，包含亮度和色度分量。图17-3中的DC电平代表亮度分量。色度分量通过突发脉冲频率取样。色度分量如图17-3的正弦波所示。

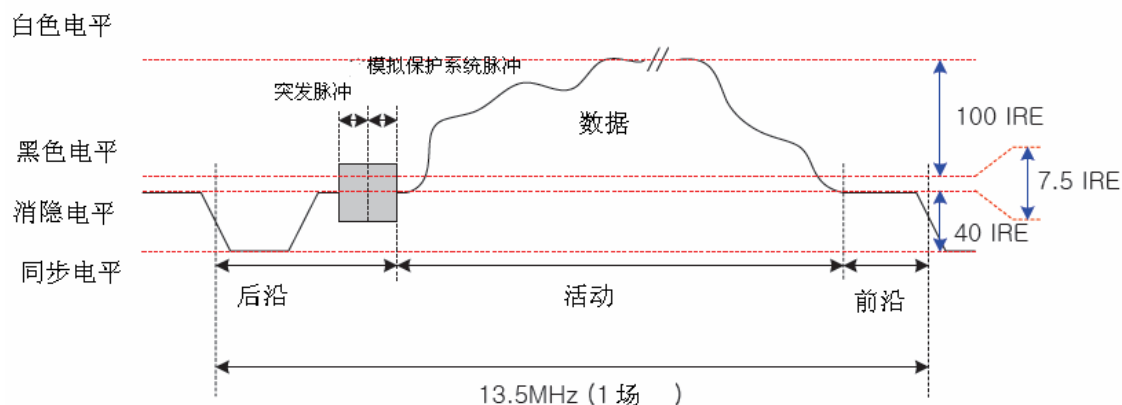


图 17-3 复合模拟信号合成

根据TV系统是NTSC制式还是PAL制式，从消隐电平中辨别出黑色电平。

## 2.公共的 NTSC 制式系统

如图17-4所示，NTSC的类型有NTSC-M、NTSC-J和NTSC 4.43 U. S.。韩国使用NTSC-M，日本使用NTSC-J（在消隐电平方面不同于NTSC-M），一些南亚国家使用NTSC 4.43，NTSC 4.43有4.43361875MHz的副载波

频率。

"M"	"NTSC-J"	"NTSC 4,43"
LINE/FIELD = 525 / 59.94 FH = 15,734 KHZ FV = 59.94 HZ FSC = 3.579545 MHZ  BLANKING SETUP = 7.5 IRE VIDEO BANDWIDTH = 4.2 MHZ AUDIO CARRIER = 4.5 MHZ CHANNEL BANDWIDTH = 6 MHZ	LINE/FIELD = 525 / 59.94 FH = 15,734 KHZ FV = 59.94 HZ FSC = 3.579545 MHZ  BLANKING SETUP = 0 IRE VIDEO BANDWIDTH = 4.2 MHZ AUDIO CARRIER = 4.5 MHZ CHANNEL BANDWIDTH = 6 MHZ	LINE/FIELD = 525 / 59.94 FH = 15,734 KHZ FV = 59.94 HZ FSC = 4.43361875 MHZ  BLANKING SETUP = 7.5 IRE VIDEO BANDWIDTH = 4.2 MHZ AUDIO CARRIER = 4.5 MHZ CHANNEL BANDWIDTH = 6 MHZ

图17-4公共的NTSC制式系统

3.公共的 PAL 制式系统

PAL制式系统的类型如图17-5所示。

"I"	"B, B1, G, H"	"M"
LINE/FIELD = 625 / 50 FH = 15,625 KHZ FV = 50 HZ FSC = 4.43361875 MHZ  BLANKING SETUP = 0 IRE VIDEO BANDWIDTH = 5.5 MHZ AUDIO CARRIER = 5.9996 MHZ CHANNEL BANDWIDTH = 8 MHZ	LINE/FIELD = 625 / 50 FH = 15,625 KHZ FV = 50 HZ FSC = 4.43361875 MHZ  BLANKING SETUP = 0 IRE VIDEO BANDWIDTH = 5.5 MHZ AUDIO CARRIER = 5.5 MHZ CHANNEL BANDWIDTH: B = 7 MHZ B1, G, H = 8 MHZ	LINE/FIELD = 525 / 59.94 FH = 15,734 KHZ FV = 59.94 HZ FSC = 3.57561149 MHZ  BLANKING SETUP = 7.5 IRE VIDEO BANDWIDTH = 4.2 MHZ AUDIO CARRIER = 4.5 MHZ CHANNEL BANDWIDTH = 6 MHZ
"D"	"N"	"Nc"
LINE/FIELD = 625 / 50 FH = 15,625 KHZ FV = 50 HZ FSC = 4.43361875 MHZ  BLANKING SETUP = 0 IRE VIDEO BANDWIDTH = 6.0 MHZ AUDIO CARRIER = 6.5 MHZ CHANNEL BANDWIDTH = 8 MHZ	LINE/FIELD = 625 / 50 FH = 15,625 KHZ FV = 50 HZ FSC = 4.43361875 MHZ  BLANKING SETUP = 7.5 IRE VIDEO BANDWIDTH = 5.0 MHZ AUDIO CARRIER = 5.5 MHZ CHANNEL BANDWIDTH = 6 MHZ	LINE/FIELD = 625 / 50 FH = 15,625 KHZ FV = 50 HZ FSC = 3.58205625 MHZ  BLANKING SETUP = 0 IRE VIDEO BANDWIDTH = 4.2 MHZ AUDIO CARRIER = 4.5 MHZ CHANNEL BANDWIDTH = 6 MHZ

图17-5 公共的PAL制式系统

.4. 屏幕的组成

在60Hz类型中，一帧的大小是858×525。它包括同步和实像区域。实像是720×480。然而图17-6中不是720而是1440。因为嵌入S3C6410X的TV编码器需要两倍的水平数据率，它用来提高图像质量。

在50Hz类型中，一帧的大小是864×625。实像是720x576。它只是在垂直方向上大小不同。



而且，所有的类型都有under-scan（扫描不足）区域。我们能以 $720 \times 480$ 或 $720 \times 576$ 显示图像，但是禁止看到整个区域。

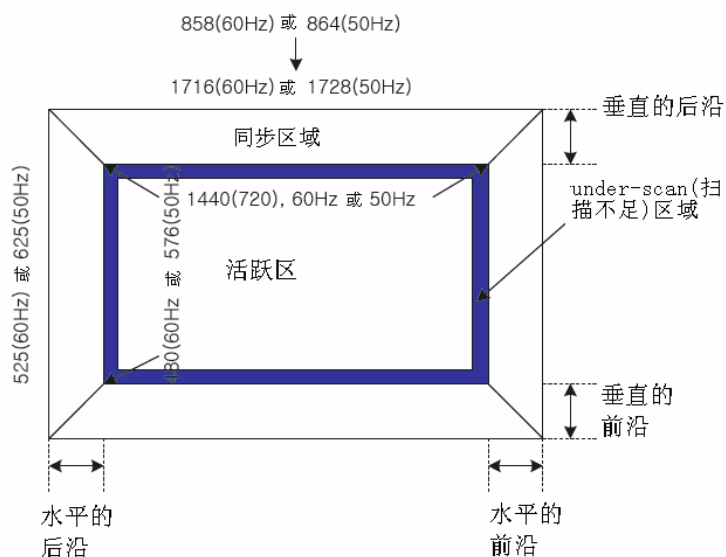


图17-6 TV屏幕的组成

## 5.请求的水平时序

如图17-7所示，描述的与水平时序相关。该水平线由活动和同步区域组成。在60Hz，一条线由活动的1440像素、前沿32像素和后沿（包括同步宽度）244像素组成。它也能配置水平under-scan（扫描不足）区域大小。水平增强器偏移值表示嵌入TV编码器的增强装置需要26个时钟来增强图像。因此，传输数据和控制信号优先于26个时钟来调整时间。

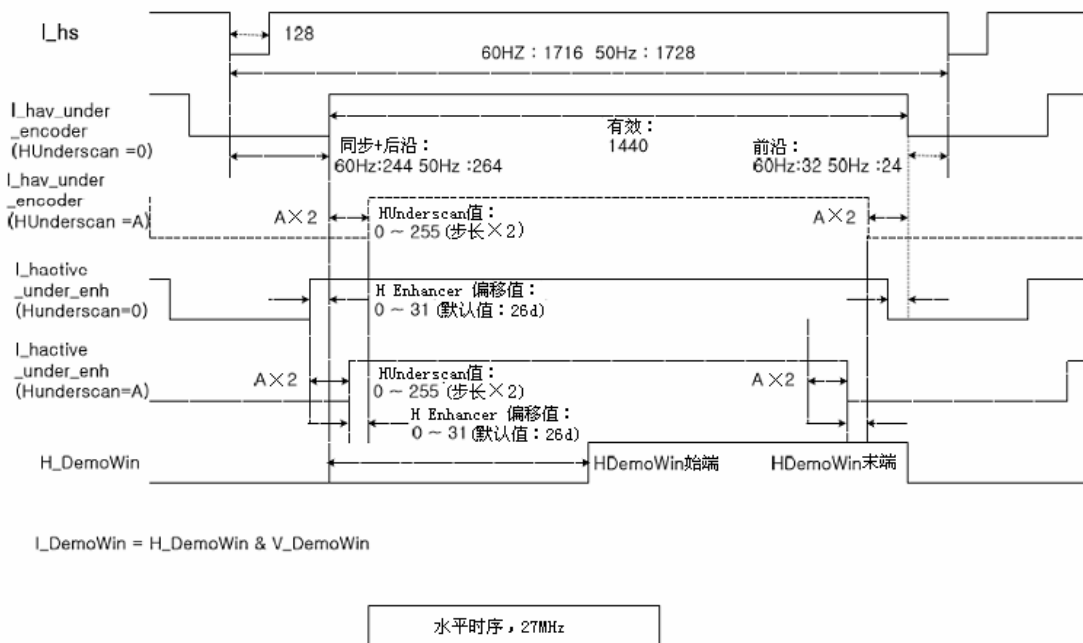


图 17-7 请求的水平时序图

同样，50Hz类型的线，包含活动的1440像素，前沿24像素和后沿264像素。其它的和60Hz的一样。

## 6.请求的垂直时序

请求的垂直时序图，如图17-8（60Hz）、17-9（50Hz）所示。

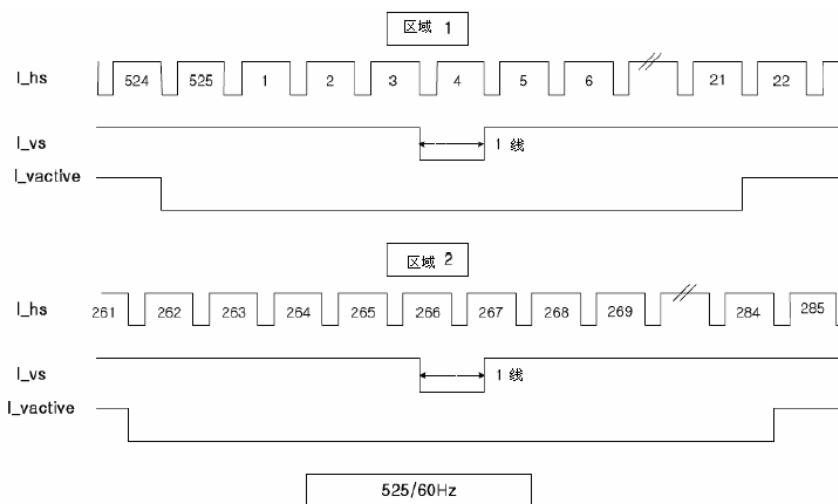


图 17-8 60Hz，请求的垂直时序图

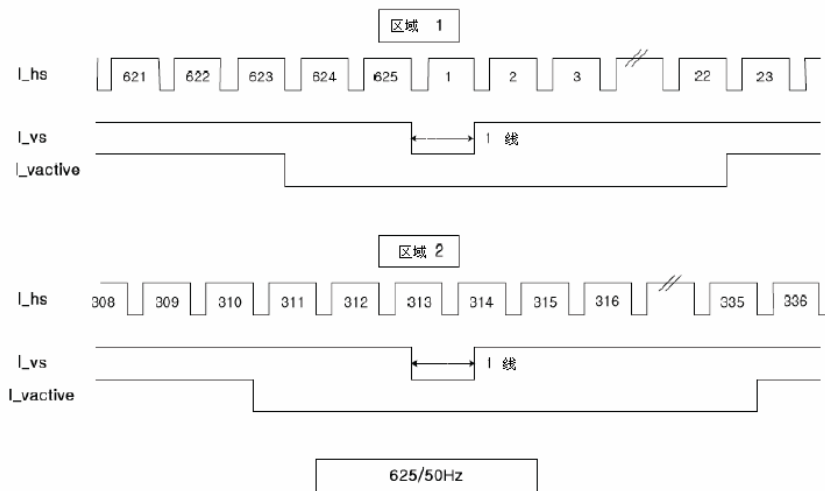


图 17-9 50Hz，请求的垂直时序图

## 7.解释影像增强器的术语

该点在输入/输出能量相等时，是变形点。方程通常是线性的。但是如果在方程式中得出变形点，那么对于输入输出关系，能得出不同的结论。例如，如果宽输出被窄输入掩盖，那么能很容易得到变化的亮度，如图17-10所示。如果能很好利用黑和白倾斜点，则能得到增亮的图像。

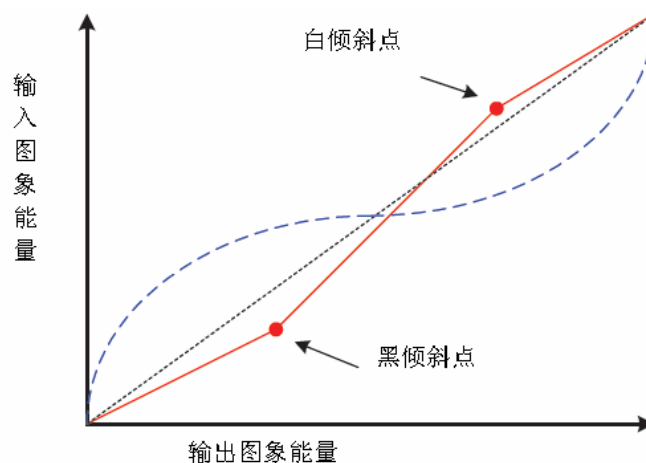


图17-10 在黑色和白色延展中倾斜点的概念

## 8.模拟保护系统（MACROVISION）

模拟保护系统是Macrovision Corporation公司提出的，用于防止录像带的拷贝。

该模拟保护系统具有以下特性：

- 虚同步、AGC脉冲产生。
- 场后沿脉冲结束。
- 彩色条纹。
- 计时周期。

9. 条纹控制寄存器

彩色条纹的寄存器表，如表17-1所示。

表 17-1 彩色条纹的寄存器表

寄存器	描述	备注
N0[7:0]	打开/关闭控制（如果想打开macrovision，设置N0为0）。	
	N0[5]	虚同步和AGC脉冲开/关。
	N0[4]	后沿脉冲开/关。
	N0[3]	彩色条纹处理开/关。
	N0[2]	AGC 脉冲振幅静态/变量。
	N0[1]	外侧，VBI降低同步和消隐。
	N0[0]	内侧，VBI降低同步和消隐的电平。
N1[5:0]	指示彩色条纹出现的第一行（场1中）。	1~64 行（00000 = 1行）
N2[5:0]	第一行和第二行之间的彩色条纹间隔（场1中）。	0~63行
N3[5:0]	指示彩色条纹出现的第一行（场2中）。	264~327/314~377 行
N4[5:0]	第一行和第二行彩色条纹之间的间隔（场2中）。	0~63行
N5[2:0]	第二行彩色条纹后面的彩色条纹间隔。	$N5 + 16 / 2(N2 + 16)$
N6[2:0]	每场中彩色条纹的数量。	$N6 + 6$
N7[1:0]	由每一条彩色条纹组成的水平线的数量。	$N7 + 2$
N16[1]	同步传输禁止后，突发脉冲没有在两个周期的边缘发生。	
N17[3:0]	脉冲zone1持续的时间。	0~2. 22μs
N18[3:0]	脉冲zone2持续的时间。	0~2. 22μs
N19[3:0]	脉冲zone3持续的时间。	0~4. 44μs
N20[2:0]	选择副载波相位带的每一位。	0 = 正常相位
N21[9:0]	选择包含彩色条纹的行的相位，最多5行。	11 = 180 相位倒置

彩色条纹（macrovision control）水平时序图如图17-11。

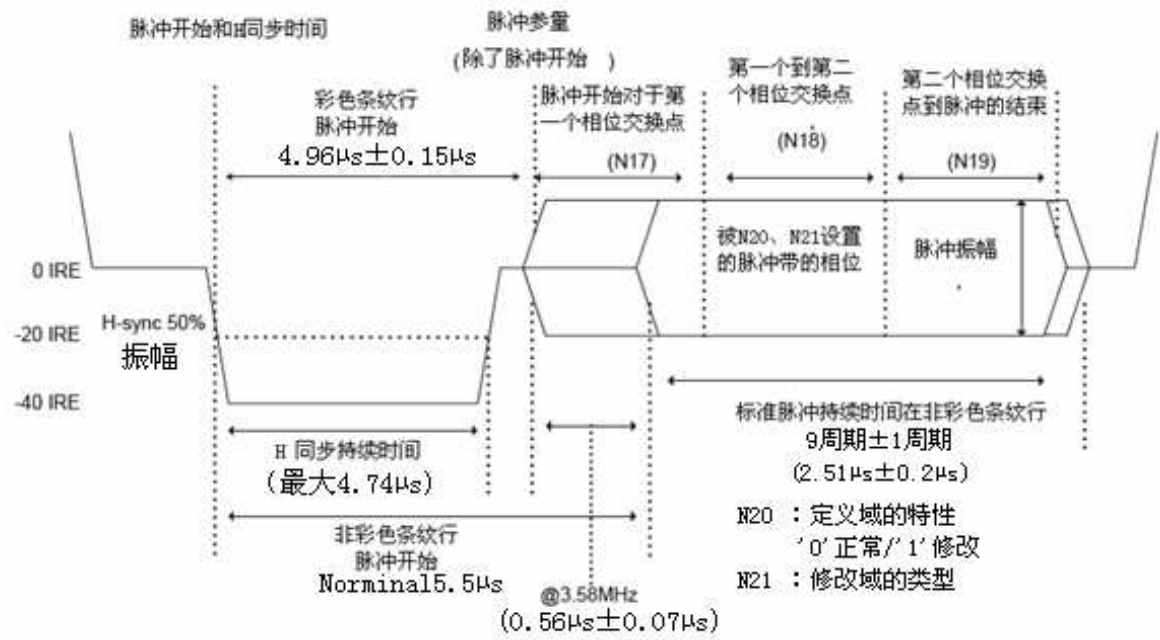


图17-11 （macrovision control）水平时序图

10. 和AGC脉冲控制寄存器

虚同步和AGC脉冲寄存器，如表17-2所示。

表 17-2 虚同步和 AGC 脉冲寄存器表

寄存器	描述	备注
N8[5:3]	在格式A中，虚同步持续的时间。	2(12 + reg) / 13.5M [525] 2(8 + reg) / 13.5M [625]
N8[2:0]	在格式B中，虚同步持续的时间。	
N9[5:3]	格式A中，第一个虚同步脉冲的位置。	8(12 + reg) / 13.5M
N9[2:0]	格式B中，第一个虚同步脉冲的位置。	
N10[5:3]	格式A中，虚同步脉冲的间隔。	8(11 + reg) / 13.5M [525]
N10[2:0]	格式B中，虚同步脉冲的间隔。	8(7 + reg) / 13.5M [625]
N11[14:0]	P. S AGC打开/关闭（每一位）(21~7) (284~270 / 333~319)。	0n = 1
N12[14:0]	格式选择(每一位) (21~7) (284~270 / 333~319)。	Format A = 0
N13[7:0]	格式A的打开/关闭控制。	0n = 1

N14[7:0]	格式B的打开/关闭控制。	On = 1
N15[7:4]	在垂直同步有效前，后沿脉冲的数量。	
N15[3:0]	在垂直同步无效后，后沿脉冲的数量。	

虚同步和AGC脉冲（macrovision control）垂直时序图如图17-12。

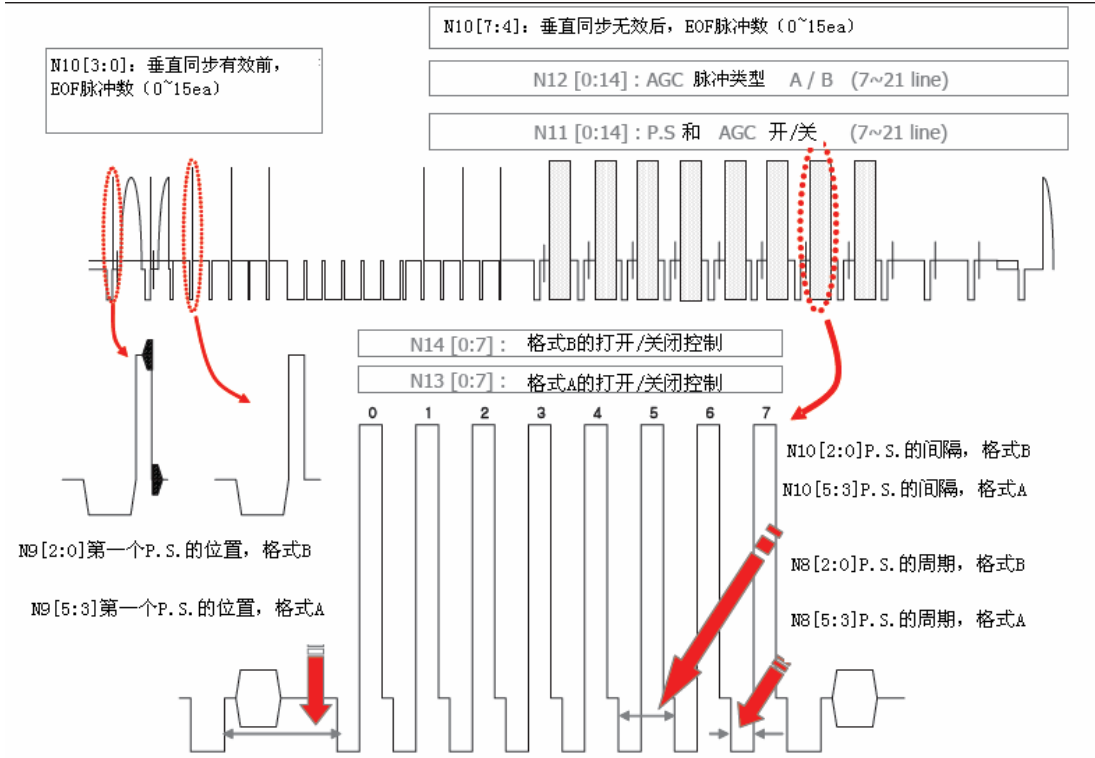


图17-12 同步和AGC脉冲（macrovision control）垂直时序图

11. 初始值实例

Macrovision初始值的例子，如表17-3所示。

表 17-3 Macrovision 初始值的例子

NTSC					
模式	寄存器	值	模式	寄存器	值
AGC 4L	Macrovision0	0x2115_D73E	AGC	Macrovision0	0x2511_1D3E
	Macrovision1	0x0205_0515		Macrovision1	0x0007_0101

	Macrovision2	0x0024_1B1B		Macrovision2	0x0024_1B1B
	Macrovision3	0x0000_07F8		Macrovision3	0x0000_07F8
	Macrovision4	0x0160_0F0F		Macrovision4	0x0160_0F0F
	Macrovision5	0x0405_000A		Macrovision5	0x0405_000A
	Macrovision6	0x0000_03FF		Macrovision6	0x0000_03FF
N01	Macrovision0	0x2115_173E	N02	Macrovision0	0x1A2A_2F3E
	Macrovision1	0x0305_0515		Macrovision1	0x0304_0236
	Macrovision2	0x0023_1C19		Macrovision2	0x001D_2524
	Macrovision3	0x7E07_0FF8		Macrovision3	0x6DCF_36B
	Macrovision4	0x0191_0EF0		Macrovision4	0x0070_1323
	Macrovision5	0x0203_0705		Macrovision5	0x050A_0302
	Macrovision6	0x0000_03C3		Macrovision6	0x0000_03A0
PAL					
模式	寄存器	值	模式	寄存器	值
P01	Macrovision0	0x2A22_1A3E	P02	Macrovision0	0x2A22_1A3E
	Macrovision1	0x0002_0522		Macrovision1	0x0302_0522
	Macrovision2	0x0014_3D1C		Macrovision2	0x002B_1223
	Macrovision3	0x0154_03FE		Macrovision3	0x1F43_78C6
	Macrovision4	0x0060_7EFE		Macrovision4	0x01F0_A353
	Macrovision5	0x0704_0008		Macrovision5	0x0203_0C0B
	Macrovision6	0x0000_0155		Macrovision6	0x0000_0385

TV编码器在ARM11中，Macrovision初始值代码实现如下：

```
// Function Name : TVENC_EnableMacroVision
```

```
// Function Description : Enable Macrovision
```

```
// Input : eTvmode - TV output format
```

```
//          ePattern - Macrovision pattern
```

```
// Output : None
```

```
void TVENC_EnableMacroVision(TV_STANDARDS eTvmode, eMACROPATTERN ePattern)
```

```

{
    switch(ePattern)
    {
        case eAGC4L :    Outp32(MACROVISION0, 0x2115D73E);
                        Outp32(MACROVISION1, 0x02050515);
                        Outp32(MACROVISION2, 0x00241B1B);
                        Outp32(MACROVISION3, 0x000007F8);
                        Outp32(MACROVISION4, 0x01600F0F);
                        Outp32(MACROVISION5, 0x0405000A);
                        Outp32(MACROVISION6, 0x000003FF);
                        break;

        case eAGC2L :    Outp32(MACROVISION0, 0x25111D3E);
                        Outp32(MACROVISION1, 0x00070101);
                        Outp32(MACROVISION2, 0x00241B1B);
                        Outp32(MACROVISION3, 0x000007F8);
                        Outp32(MACROVISION4, 0x01600F0F);
                        Outp32(MACROVISION5, 0x0405000A);
                        Outp32(MACROVISION6, 0x000003FF);
                        break;

        case eN01      :    Outp32(MACROVISION0, 0x2115173E);
                        Outp32(MACROVISION1, 0x03050515);
                        Outp32(MACROVISION2, 0x00231C19);
                        Outp32(MACROVISION3, 0x7E070FF8);
                        Outp32(MACROVISION4, 0x01910EF0);
                        Outp32(MACROVISION5, 0x02030705);
                        Outp32(MACROVISION6, 0x000003C3);
                        break;

        case eN02      :    Outp32(MACROVISION0, 0x1A2A2F3E);
                        Outp32(MACROVISION1, 0x03040236);
    }
}

```



```

        Outp32(MACROVISION2, 0x001D2524);
        Outp32(MACROVISION3, 0x6DCF36B8);
        Outp32(MACROVISION4, 0x00701323);
        Outp32(MACROVISION5, 0x050A0302);
        Outp32(MACROVISION6, 0x000003A0);
        break;
    case eP01 : Outp32(MACROVISION0, 0x2A221A3E);
        Outp32(MACROVISION1, 0x00020522);
        Outp32(MACROVISION2, 0x00143D1C);
        Outp32(MACROVISION3, 0x015403FE);
        Outp32(MACROVISION4, 0x00607EFE);
        Outp32(MACROVISION5, 0x07040008);
        Outp32(MACROVISION6, 0x00000155);
        break;
    case eP02 : Outp32(MACROVISION0, 0x2A221A3E);
        Outp32(MACROVISION1, 0x03020522);
        Outp32(MACROVISION2, 0x002B1223);
        Outp32(MACROVISION3, 0x1F4378C6);
        Outp32(MACROVISION4, 0x01F0A353);
        Outp32(MACROVISION5, 0x02030C0B);
        Outp32(MACROVISION6, 0x00000385);
        break;
    default : break;
}
}
// Function Name : TVENC_DisableMacroVision
// Function Description : Disable Macrovision
// Input : None
// Output : None

```

```
void TVENC_DisableMacroVision(void)
{
    u32 uTemp;
    uTemp = Inp32(MACROVISION0);
    uTemp &= ~(0xFF);
    Outp32(MACROVISION0, uTemp);
}
```

12. 构成的介绍

如图17-13，S3C6410X中是电流驱动DAC，输出电流是6.6mA。因此，它不得不装载150[ohm]电阻器。推荐使用AMP设备，因为它能防止ESD的电损伤。

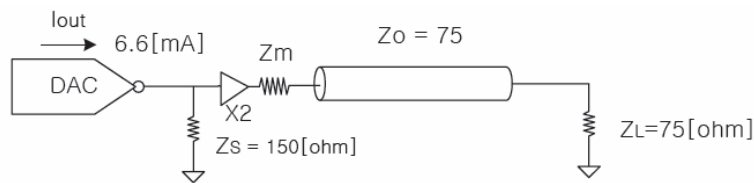


图17-13 AC板介绍

17.3 TV 编码器的寄存器

TV 编码器的寄存器简单介绍，如表 17-4 所示。

表 17-4 TV 编码器的寄存器简介

寄存器	地址	读/写	描述	复位值
TVCTRL	0x76200000	读/写	TC控制器控制SFR设置。	0x0001_0000
VBPORCH	0x76200004	读/写	垂直后沿结束点。	0x011C_0015
HBPORCH	0x76200008	读/写	水平后沿结束点。	0x0080_00F4
HEnhOffset	0x7620000C	读/写	水平增强器偏移。	0x0000_041A
VDemoWinSize	0x76200010	读/写	垂直演示窗口大小。	0x00F0_0000
HDemoWinSize	0x76200014	读/写	水平演示窗口大小。	0x05A0_0000
InImageSize	0x76200018	读/写	输入图像大小。	0x01E0_05A0

PEDCTRL	0x7620001C	读/写	编码器基座控制。	0x0000_0000
YCFILTERBW	0x76200020	读/写	Y/C滤波器带宽控制。	0x0000_0043
HUECTRL	0x76200024	读/写	HUE控制。	0x0000_0000
FscCTRL	0x76200028	读/写	Fsc（副载波频率）控制。	0x0000_0000
FscDTOManCTRL	0x7620002C	读/写	Fsc DTO 手动控制。	0x0000_0000
BGCTRL	0x76200034	读/写	背景控制。	0x0000_0110
BGHVAVCTRL	0x76200038	读/写	背景VAV 和 HAV控制。	0xB400_F000
ContraBright	0x76200044	读/写	对比度和亮度控制。	0x0000_0040
CbCrGainCTRL	0x76200048	读/写	Cb 和 Cr增益控制。	0x0040_0040
DemoWinCTRL	0x7620004C	读/写	演示窗口控制。	0x0000_0010
FTCA	0x76200050	读/写	Flesh tone控制。	0x00D7_008C
BWGAIN	0x76200058	读/写	黑和白色信号展宽增益控制。	0x0000_0034
SharpCTRL	0x76200060	读/写	锐度控制。	0x0304_501F
GammaCTRL	0x76200064	读/写	伽玛控制。	0x0000_0104
FscAuxCTRL	0x76200068	读/写	FSC辅助控制。	0x0000_0000
SyncSizeCTRL	0x7620006C	读/写	同步尺寸控制。	0x0000_003D
BurstCTRL	0x76200070	读/写	突发脉冲信号控制。	0x0069_0049
MacroBurstCTRL	0x76200074	读/写	Macrovision脉冲信号控制。	0x0000_0041
ActVidPoCTRL	0x76200078	读/写	活动视频位置控制。	0x0348_0078
EncCTRL	0x7620007C	读/写	编码器控制。	0x0000_0011
MuteCTRL	0x76200080	读/写	静音控制。	0x8080_1001
Macrovision0	0x76200084	读/写	Macrovision控制0。	0x2115_1700
Macrovision1	0x76200088	读/写	Macrovision控制1。	0x0205_0515
Macrovision2	0x7620008C	读/写	Macrovision控制2。	0x0024_1B1B
Macrovision3	0x76200090	读/写	Macrovision控制3。	0x0000_07F8
Macrovision4	0x76200094	读/写	Macrovision控制4。	0x0160_0F0F
Macrovision5	0x76200098	读/写	Macrovision控制5。	0x0405_000A
Macrovision6	0x7620009C	读/写	Macrovision控制6。	0x0000_03FF

下面分别对个别寄存器进行简单的介绍：

### 17.3.1. TV 控制器

#### 1.TVENC\_REG1

寄存器	地址	读/写	描述	复位值
TVCTRL	0x76200000	读/写	TV控制器控制SFR设置。	0x0001_0000

TVCTRL	位	描述	初始状态
	[31:17]	保留。	0
INTFIFOUR	[16]	FIFO under-run（数据不足）中断控制。 0 :无效 1 :有效	0x1
	[15:13]	保留	0
INTStatus	[12]	FIFO under-run（数据不足）状态寄存器。 如果寄存器是高位，则FIFO under-run（数据不足）中断发生。如果想清除该中断，也得写入高位。	0
	[11:9]	保留	0
TVOUTTYPE	[8]	选择TV输出类型。 0 :复合数据 1 : S-Video 输出	0
	[7]	保留。	0
TVOUTFMT	[6:4]	选择TV输出格式。 0 : NTSC-M 1 : NTSC-J 2 : PAL-B/D/G/H/I 3 : PAL-M 4 : PAL-Nc 其它 : 保留	0
	[3:1]	保留。	0
TVONOFF	[0]	TV编码器开/关控制。 0 : 关 1 : 开	0

## 2.TVENC\_REG2

寄存器	地址	读/写	描述	复位值
VBPORCH	0x76200004	读/写	垂直后沿结束点。	0x011C_0015

VBPORCH	位	描述	初始状态
	[31:25]	保留。	0
VEFBPD	[24:16]	垂直偶数场后沿结束点。 NTSC : 0x11C(284) , PAL : 0x14F(335)	0x11C
	[15:8]	保留。	0
VOFBPD	[7:0]	垂直奇数场后沿结束点。 NTSC : 0x15(21), PAL : 0x16(22)	0x15

## 3.TVENC\_REG3

寄存器	地址	读/写	描述	复位值
HBPORCH	0x76200008	读/写	水平后沿结束点。	0x0080_00F4

HBPORCH	位	描述	初始状态
	[31:24]	保留。	0
HSPW	[23:16]	水平同步脉冲宽度。 默认: 0x80(128) (NTSC, PAL)	0x80
	[15:11]	保留。	0
HBPD	[10:0]	水平后沿结束点。 NTSC : 0xF4(244) PAL : 0x108(264)	0xF4

## 4.TVENC\_REG4

寄存器	地址	读/写	描述	复位值
HEnhOffset	0x7620000C	读/写	水平增强器偏移。	0x0000_041A

HEnhOffset	位	描述	初始状态
	[31:30]	保留。	0
VACTWinCenCTRL	[29:24]	垂直活动窗口中心控制。	0
HACTWinCenCTRL	[23:16]	水平活动窗口中心控制。	0

	[15:11]	保留。	0
DTOffset	[10:8]	数据输入时序偏移值。 NTSC, PAL : 0x4(4)	0x4
	[7:5]	保留。	0
HEOV	[4:0]	水平增强器偏移值。在TV编码器开始(0~31)前,增强器需要26个周期。 默认: 0x1A(26) (NTSC, PAL)	0x1A

## 5.TVENC\_REG5

寄存器	地址	读/写	描述	复位值
VDemoWinSize	0x76200010	读/写	垂直演示窗口大小。	0x00F0_0000

VDemoWinSize	位	描述	初始状态
	[31:25]	保留。	0
VDWS	[24:16]	垂直演示窗口大小。 默认: 0xF0(240)	0xF0
	[15:9]	保留。	0
VDWSP	[8:0]	垂直演示窗口起始点。 默认: 0x0(0)	0

## 6.TVENC\_REG6

寄存器	地址	读/写	描述	复位值
HDemoWinSize	0x76200014	读/写	水平演示窗口大小。	0x05A0_0000

HDemoWinSize	位	描述	初始状态
	[31:27]	保留。	0
HDWEP	[26:16]	水平演示窗口大小。 默认: 0x5A0(1440)	0x5A0
	[15:11]	保留。	0
HDWSP	[10:0]	水平演示窗口起始点。 默认: 0x0(0)	0

7.TVENC\_REG7

寄存器	地址	读/写	描述	复位值
InImageSize	0x76200018	读/写	输入图像大小。	0x01E0_05A0

InImageSize	位	描述	初始状态
	[31:26]	保留。	0
ImageHeight	[25:16]	输入图像高度（最大值：576）。	0x1E0
	[15:11]	保留。	0
ImageWidth	[10:0]	输入图像的宽度。输入值是原始输出图像宽度的两倍。例如，当图像宽度是720时，必须设置为1440（最大值为1440）。	0x5A0

17. 3. 2. 编码器

1.TVENC\_REG8

寄存器	地址	读/写	描述	复位值
PEDCTRL	0x7620001C	读/写	编码器基座控制。	0x0000_0000

PEDCTRL	位	描述	初始状态
	[31:1]	保留。	0
PEDOff	[0]	编码器基座控制。 0：基座打开 (NTSCM与PALM) 1：基座关闭 (偶数 NTSCM与PALM)	0

2.TVENC\_REG9

寄存器	地址	读/写	描述	复位值
YCFilterBW	0x76200020	读/写	Y/C滤波器带宽控制。	0x0000_0043

YCFilterBW	位	描述	初始状态
	[31:7]	保留。	0
YBW	[6:4]	亮度带宽（-3dB）。 0：6.0 MHz (推荐以S-Video输出) 1：3.8 MHz	0x4

		2 : 3.1 MHz 3 : 2.6 MHz (推荐复合输出, PAL4.43MHz槽口) 4 : 2.1 MHz (推荐复合输出, NTSC/PALM 3.58MHz槽口)	
	[3:2]	保留	0
CBW	[1:0]	色度带宽。 0 : 1.2 MHz 1 : 1.0 MHz 2 : 0.8 MHz 3 : 0.6 MHz	0x3

### 3.TVENC\_REG10

寄存器	地址	读/写	描述	复位值
HUECTRL	0x76200024	读/写	HUE控制	0x0000_0000

HUECTRL	位	描述	初始状态
HUE	[31:8]	保留。	0
	[7:0]	彩色HUE控制（有一个1.4063的增量）。 0x00 : 0' 相位移位 ... 0x80 : 180' 相位移位 ... 0xFF : 358.5938' 相位移位	0

### 4.TVENC\_REG11

寄存器	地址	读/写	描述	复位值
FscCTRL	0x76200028	读/写	Fsc(副载波频率)控制。	0x0000_0000

FscCTRL	位	描述	初始状态
	[31:15]	保留。	0
FscCtrl	[14:0]	Fsc 控制(+ / -)。 〈方程式〉: (注) FscCtrl : 2' s) = 当前DT0设置值 + FscCtrl[14:0] × (2^9)	0



**5.TVENC\_REG12**

寄存器	地址	读/写	描述	复位值
FscDTOManCTRL	0x7620002C	读/写	Fsc DT0手动控制。	0x0000_0000

FscTdoManCTRL	位	描述	初始状态
FscMEn	[31]	Fsc DT0手动控制有效。	0
FscDTOManual	[30:0]	Fsc DT0 手动控制。 <方程式>: $\text{FscDTOManual}[30:0] = \text{FSC} * (2^{33}) / \text{Fclk}$ 例如, NTSC $3.5795454545 \times (2^{33}) / 27 = 0x43E0F83E$	0

**6.TVENC\_REG14**

寄存器	地址	读/写	描述	复位值
BGCTRL	0x76200034	读/写	背景控制。	0x0000_0110

BGCTRL	位	描述	初始状态
	[31:9]	保留。	0
SME	[8]	软混合有效。 0 : 无效 1 : 背景边缘的有效软混合	1
	[7]	保留。	0
BGCS	[6:4]	背景色选择。 0 : 黑色 1 : 蓝色 2 : 红色 3 : 紫色 4 : 绿色 5 : 青色 6 : 黄色 7 : 白色	1
BGYOFS	[3:0]	背景亮度的偏移量。	0

7.TVENC\_REG15

寄存器	地址	读/写	描述	复位值
BGHVAVCTRL	0x76200038	读/写	背景VAV和HAV的控制。背景计算方法如图17-14所示。	0xB400_F000

BGHVAVCTRL	位	描述	初始状态
BG_HL	[31:24]	背景 HAV的长度（8倍）。 默认：0xB4(180)	0xB4
BG_HS	[23:16]	背景 HAV的起始位置（8倍）。 默认：0x0(0)	0x00
BG_VL	[15:8]	背景 VAV的长度（1倍）。 默认：0xF0(240)	0xF0
BG_VS	[7:0]	背景 HAV的起始位置（1倍）。 默认：0x0(0)	0x00

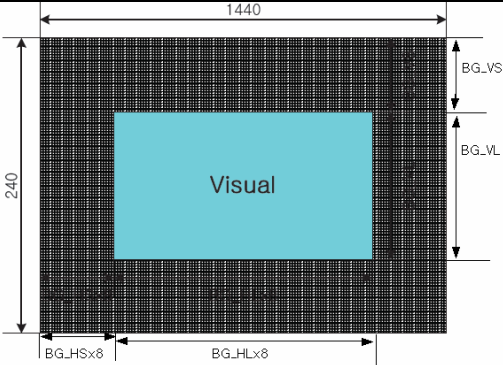


图17-14 方法

17.3.3. 图像增强器

1.TVENC\_REG18

寄存器	地址	读/写	描述	复位值
ContraBright	0x76200044	读/写	对比度和亮度控制。	0x0000_0040

ContraBright	位	描述	初始状态
	[31:24]	保留。	0
BRIGHT	[23:16]	亮度控制（2的补数）。 0x7F：最大亮度 0x80：最小亮度	0x00

	[15:8]	保留。	0
CONTRAST	[7:0]	对比度增益控制：0~4。	0x40

## 2.TVENC\_REG19

寄存器	地址	读/写	描述	复位值
CbCrGainCTRL	0x76200048	读/写	Cb 和 Cr 增益控制。	0x0040_0040

CbCrGainCTRL	位	描述	初始状态
	[31:24]	保留。	0
CR_GAIN	[23:16]	Cr增益控制（增益：0~2）。	0x40
	[15:8]	保留。	0
CB_GAIN	[7:0]	Cb增益控制（增益：0~2）。	0x40

## 3.TVENC\_REG20

寄存器	地址	读/写	描述	复位值
DemoWinCTRL	0x7620004C	读/写	演示窗口控制。	0x0000_0010

DemoWinCTRL	位	描述	初始状态
	[31:25]	保留。	0
MVDemo	[24]	增强器演示窗口开/关。 0: 正常操作 1: 增强器示范窗口模式	0
	[23:17]	保留	0
FreshEn	[16]	Fresh tone修正开/关。 0 : Fresh tone 修正无效 1 : Fresh tone 修正有效	0
	[15:13]	保留	0
BStOff	[12]	黑色信号展宽关闭控制。 0: 黑色信号展宽有效 1: 黑色信号展宽无效	0
	[11:9]	保留	0
WStOff	[8]	白色信号展宽关闭控制。 0: 白色信号展宽有效 1: 白色信号展宽无效	0

	[7:2]	保留	0
BSGn	[1:0]	蓝色信号展宽增益控制。 0：蓝色信号展宽关闭 3：蓝色信号展宽最大增益	0

#### 4.TVENC\_REG21

寄存器	地址	读/写	描述	复位值
FTCA	0x76200050	读/写	Flesh tone控制。	0x00D7_008C

FTCA	位	描述	初始状态
	[31:24]	保留	0
FTCAC	[23:16]	Flesh tone修正角度：余弦值。 〈方程式〉： $FTCAC = \cos(x - 90^\circ) \times (28) \quad (x: 90 \sim 180 \text{ 度})$ 例如， $x = 123 \text{ 度}$ $FTCAC = \cos(123^\circ - 90^\circ) \times (28) = 0xD7$	0xD7
	[15:8]	保留	0
FTCAS	[7:0]	Flesh tone修正角度：正弦值。 〈方程式〉： $FTCAS = \sin(x - 90^\circ) \times (28) \quad (x: 90 \sim 180 \text{ 度})$ 例如， $x = 123 \text{ 度}$ $FTCAS = \sin(123^\circ - 90^\circ) \times (28) = 0x8B$	0x8C

#### 5.TVENC\_REG23

寄存器	地址	读/写	描述	复位值
BWGAIN	0x76200058	读/写	黑色和白色信号展宽增益控制。	0x0000_0034

BWGAIN	位	描述	初始状态
	[31:8]	保留。	0
WGain	[7:4]	白色信号展宽增益。	0x3
BGain	[3:0]	黑色信号展宽增益。	0x4

6.TVENC\_REG25

寄存器	地址	读/写	描述	复位值
SharpCTRL	0x76200060	读/写	锐度控制。	0x0304_501F

SharpCTRL	位	描述	初始状态
	[31:28]	保留。	0
SHARP_T	[27:20]	动态锐度倾斜点。	0x30
	[19:15]	保留。	0
SDhCor	[14:12]	锐度取心控制。 0 : 禁止取心 7 : 最大取心	0x5
	[11:10]	保留。	0
DShpF0	[9:8]	锐度中心频率（建议DShpF0=0x2在VGA(640x480)之上，DShpF0=0在QVGA(320x160)之下） 0 : 低频(2.7MHz、3.4MHz、4.5MHz) 2 : 高频(2.7MHz、3.4MHz、4.5MHz)	0
	[7:6]	保留。	0
DShpGn	[5:0]	动态锐度增益控制。 0x00 : 降低高频 0x0F : 无锐度 0x3F : 最大锐度	0x1F

7.TVENC\_REG26

寄存器	地址	读/写	描述	复位值
GammaCTRL	0x76200064	读/写	伽玛控制。	0x0000_0104

GammaCTRL	位	描述	初始状态
	[31:13]	保留。	
GamEn	[12]	伽玛有效。 0:伽玛无效 1 :伽玛有效	
	[11:10]	保留。	
GamMode	[9:8]	伽玛控制模式。 0 :最小伽玛增益 3 :最大伽玛增益	
	[7:3]	保留。	

DCTRAN	[2:0]	DC模糊增益。 0 : 80% 5 : 100% 7 : 110%	
--------	-------	--	--

## 8.TVENC\_REG27

寄存器	地址	读/写	描述	复位值
FscAuxCTRL	0x76200068	读/写	Fsc辅助控制。	0x0000_0000

FscAuxCTRL	位	描述	初始状态
	[31:5]	保留。	0
Phalt	[4]	副载波倒相控制（对于PAL制式）。 0 : 倒相无效 1 : 倒相有效	0
	[3:1]	保留。	0
Fdrst	[0]	副载波复位有效。 0 : 副载波自运行模式 1 : 副载波复位模式（每四场，副载波复位）	0

## 9.TVENC\_REG28

寄存器	地址	读/写	描述	复位值
SyncSizeCTRL	0x7620006C	读/写	同步尺寸控制。	0x0000_003D

SyncSizeCTRL	位	描述	初始状态
	[31:10]	保留。	0
Sy_Size	[9:0]	同步尺寸。 0x3D : NTSC 0x3E : PAL	0x3D

## 10.TVENC\_REG29

寄存器	地址	读/写	描述	复位值
BurstCTRL	0x76200070	读/写	突发脉冲信号控制。	0x0069_004

BurstCTRL	位	描述	初始状态
	[31:26]	保留。	0
Bu_End	[25:16]	脉冲结束位置。 0x69 : NTSC 0x6A : PAL	0x69
	[15:10]	保留。	0
Bu_St	[9:0]	脉冲起始位置。 0x49 : NTSC 0x4A : PAL	0x49

## 11.TVENC\_REG30

寄存器	地址	读/写	描述	复位值
MacroBurstCTRL	0x76200074	读/写	Macrovision脉冲信号控制。	0x0000_0041

BurstCTRL	位	描述	初始状态
	[31:10]	保留。	0
Bumav_St	[9:0]	Macrovision脉冲起始位置。 0x41 : NTSC 0x42 : PAL	0x41

## 12.TVENC\_REG31

寄存器	地址	读/写	描述	复位值
ActVidPoCTRL	0x76200078	读/写	活动视频位置控制。	0x0348_0078

ActVidPoCTRL	位	描述	初始状态
	[31:26]	保留。	0
Avon_End	[25:16]	活动视频结束位置。 0x348 : NTSC 0x352 : PAL	0x348
	[15:10]	保留。	0
Avon_St	[9:0]	活动视频起始位置。 0x78 : NTSC 0x82 : PAL	0x78

### 13.TVENC\_REG32

寄存器	地址	读/写	描述	复位值
EncCTRL	0x7620007C	读/写	编码器控制。	0x0000_0011

EncCTRL	位	描述	初始状态
	[31:1]	保留。	0
BGEn	[0]	背景有效。 0：无效 1：有效	0x1

### 14.TVENC\_REG33

寄存器	地址	读/写	描述	复位值
MuteCTRL	0x76200080	读/写	静音控制。	0x8080_1001

MuteCTRL	位	描述	初始状态
Mute_Cr	[31:24]	静音Cr元件。	0x80
Mute_Cb	[23:16]	静音Cb元件。	0x80
Mute_Y	[15:8]	静音Y元件。	
	[7:1]	保留。	
MuteOnOff	[0]	视频静音控制。 0：静音有效 1：静音无效	

### 15.TVENC\_REG34

寄存器	地址	读/写	描述	复位值
Macrovision0	0x76200084	读/写	Macrovision控制0。	0x2115_1700

Macrovision0	位	描述	初始状态
	[31:30]	保留。	0
N3	[29:24]	Macrovision控制N3。	0x21
	[23:22]	保留。	0



N2	[21:16]	Macrovision控制N2。	0x15
	[15:14]	保留。	0
N1	[13:8]	Macrovision控制N1。	0x17
N0	[7:0]	Macrovision控制N0。	0

## 16.TVENC\_REG35

寄存器	地址	读/写	描述	复位值
Macrovision1	0x76200088	读/写	Macrovision控制1。	0x0205_0515

Macrovision1	位	描述	初始状态
	[31:26]	保留。	0
N7	[25:24]	Macrovision控制N7。	0x2
	[23:19]	保留。	0
N6	[18:16]	Macrovision控制N6。	0x5
	[15:11]	保留。	0
N5	[10:8]	Macrovision控制N5。	0x5
	[7:6]	保留。	0
N4	[5:0]	Macrovision控制N4。	0x15

## 17.TVENC\_REG36

寄存器	地址	读/写	描述	复位值
Macrovision2	0x7620008C	读/写	Macrovision控制2。	0x0024_1B1B

Macrovision2	位	描述	初始状态
	[31:22]	保留。	0
N10	[21:16]	Macrovision控制N10。	0x24
	[15:14]	保留。	0
N9	[13:8]	Macrovision控制N9。	0x1B
	[7:6]	保留。	0

N8	[5:0]	Macrovision控制N8。	0x1B
----	-------	------------------	------

## 18.TVENC\_REG37

寄存器	地址	读/写	描述	复位值
Macrovision3	0x76200090	读/写	Macrovision控制3。	0x0000_07F8

Macrovision3	位	描述	初始状态
	[31]	保留。	0
N12	[30:16]	Macrovision控制N12。	0
	[15]	保留。	0
N11	[14:0]	Macrovision控制N11。	0x07F8

## 19.TVENC\_REG38

寄存器	地址	读/写	描述	复位值
Macrovision4	0x76200094	读/写	Macrovision控制4。	0x0160_0F0F

Macrovision4	位	描述	初始状态
	[31:25]	保留。	0
N16	[24]	Macrovision控制N16。	0x1
N15	[23:16]	Macrovision控制N15。	0x60
N14	[15:8]	Macrovision控制N14。	0xF
N13	[7:0]	Macrovision控制N13。	0xF

## 20.TVENC\_REG39

寄存器	地址	读/写	描述	复位值
Macrovision5	0x76200098	读/写	Macrovision控制5。	0x0405_000A

Macrovision5	位	描述	初始状态
	[31:27]	保留。	0

N20	[26:24]	Macrovision控制N20。	0x4
	[23:20]	保留。	0
N19	[19:16]	Macrovision控制N19。	0x5
	[15:12]	保留。	0
N18	[11:8]	Macrovision控制N18。	0
	[7:4]	保留。	0
N17	[3:0]	Macrovision控制N17。	0xA

**21.TVENC\_REG40**

寄存器	地址	读/写	描述	复位值
Macrovision6	0x7620009C	读/写	Macrovision控制6。	0x0000_03FF

Macrovision6	位	描述	初始状态
	[31:10]	保留。	0
N21	[9:0]	Macrovision控制N21。	0x3FF

## 18 2D 图形

FIMG-2D是一个2D图形加速器，支持三种原始绘图：线/点绘图、位区块传输（BitBLT）和彩色扩展（文本绘图）。

原始绘图需要两步：

通过设置绘图内容寄存器设定绘制参数，如前景颜色和坐标数据等；

通过设置相关指令寄存器开始绘制过程。

FIMGSE-2D 顶端模块图，如图 18-1 所示。

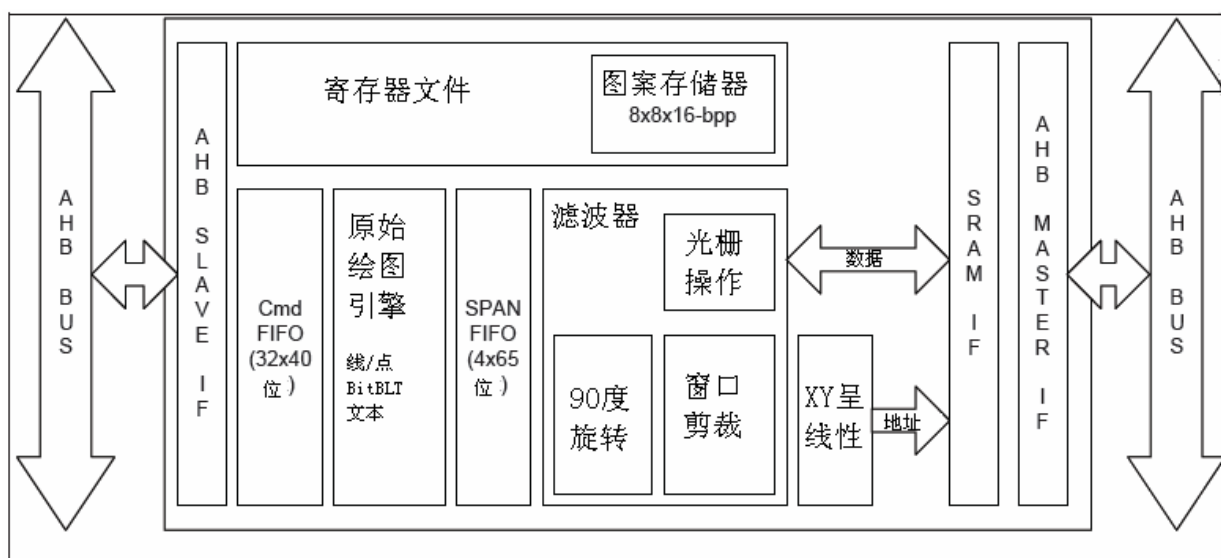


图18-1 FIMGSE-2D顶端模块图

### 18.1 2D 图形特性

#### 1. 原始绘图

（1）线/点绘图。

- DDA（数值微分分析法）算法。
- 支持不用画结点。

（2）位区块传输（BitBLT）。

- 支持延长的BitBLT（最近样点）。
  - 屏到屏。
  - 主机到屏。
- (3) 彩色扩展。
- 存储器到屏。
  - 主机到屏。

### 2. 像素操作

- (1) 最大为2048×2048图形大小。
- (2) 窗口剪裁。
- (3) 90° /180° /270° /X轴/Y轴旋转。
- (4) BitBLT透明模式。
- (5) α 混合。
- 用户指定的256级 α 值的 α 混合。
  - 像素 α 混合。
- (6) 8×8×16bpp图案绘图。

### 3. 数据格式

- (1) 支持15/16/18/24/32bpp色彩格式。
- (2) 支持小/大端。
- (3) 用于坐标数据的11. 11固定点格式。

## 18. 2 2D 图形色彩格式

FIMG2D支持下面的色彩格式：每像素15/16/18/24/32位。每种格式介绍，如图18-2所示。

15-bpp	1位	R (5位 )	G (5位 )	B (5位 )
16-bpp	R (5位 )		G (6位 )	B (5位 )
18-bpp	14 位 保留			R (6位 )    G (6位 )    B (6位 )
24/32-bpp	8 位 保留		R (8位 )    G (8位 )	B (8位 )

图18-2 FIMG2D支持的色彩格式介绍

使用COLOR\_MODE寄存器能配置色彩格式。除了图案数据，所有的色彩数据使用的色彩格式和COLOR\_MODE寄存器指定的一样，图案数据一直使用的是RGB565格式。

ARM11中rG2D\_COLOR\_MODE寄存器图形色彩格式的代码定义如下：

```
void G2D_GetBppMode(CSPACE *eBpp)
{
    u32 uBppVal;

    uBppVal=Inp32(rG2D_COLOR_MODE);
    switch(uBppVal&0xf) {
        case 1:
            *eBpp=ARGB8; //15-bpp
            break;
        case 2:
            *eBpp=RGB16; //16-bpp
            break;
        case 4:
            *eBpp=RGB18; // 18-bpp
            break;
        case 8:
            *eBpp=RGB24; // 24-bpp
            break;
        default:
            *eBpp=RGB16; //16-bpp
            break;
    }
}
```

# 18.3 2D 图形流程

如图18-3所示，介绍了绘制过程，并且在下面给出了每部分的解释。

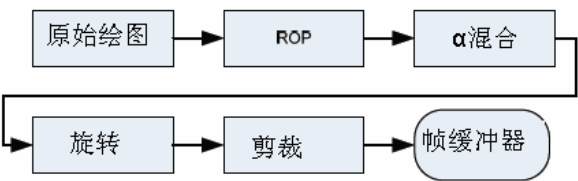


图18-3 绘制过程

## 1. 原始绘图

原始绘图决定像素填充，并且通过它们进一步操作。

FIMG-2D支持三种类型的原始绘图：点/线绘图、位块传送、彩色扩展。

### （1）线/点绘图

线绘图在起始点（sx，sy）和结束点（ex，ey）画出一条线。如果这两点距离沿着Y轴长度大于沿X轴的长度（ $|ey - sy| > |ex - sx|$ ），那么主轴应当设置为Y轴；否则，设置为X轴。如果Y轴是主轴，那么在线上，像素的y坐标是从当前像素以1为单位增加或减少，x坐标则是通过X-INCR（小于1）增加或减少。同样，如果X轴是主轴，x坐标是以1位单位增加或减少，y坐标通过Y-INCR增加或减少。

注意，X-INCR 和 Y-INCR 是以 2 的补数的形式给出的，如图 18-4 所示。



图18-4 X-INCR和Y-INCR

相关的寄存器如表18-1所示。

表18-1 线/点绘图相关寄存器

COORD_0	坐标起始点
COORD_2	坐标结束点
X-INCR	X 增量值（如果 X 轴是主轴则忽略）
Y-INCR	Y 增量值（如果 Y 轴是主轴则忽略）
FG_COLOR	绘制的线/点的颜色
CMDR_0	配置绘制线/点的参数，比如：主轴是 X 轴还是 Y 轴，是画线还是画点等等。写入该寄存器开始绘制过程

(2) 位区块传输

位区块传输是像素矩形块的转化。典型应用包括将图形的一部分复制到另一个位置，通过光栅操作合成位图，改变图形大小等。

FIMG-2D在透明模式下可以绘制图形。在该模式下，像素的颜色和背景颜色相同(BG\_COLOR)，而不是蓝屏的颜色(BS\_COLOR)。如图18-5所示，BG\_COLOR分别设置为白色和BS\_COLOR蓝。

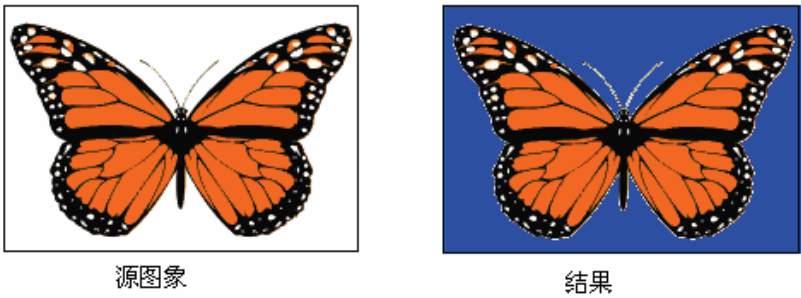


图18-5 位区块传输

FIMG-2D支持BLT的主机到屏模式和存储器到屏模式。

相关寄存器介绍如表18-2所示。

表18-2 位块传送相关寄存器

COORD_0	源图形的最左上端坐标
COORD_1	源图形最右下端的坐标
COORD_2	目标图形最左上端的坐标
COORD_3	目标图形最右下端的坐标
X-INCR	源图形X坐标增量值。如果大于1，则水平压缩；小于1，则扩大。当CMDR_1的S位无效或者使用主机到屏模式，则忽略该值。
Y-INCR	源图形的Y坐标增量值。如果它大于1，该图形垂直压缩；小于1则扩大。当CMDR_1中的S位无效或者使用主机到屏模式则忽略该值。
COLOR_MODE	图形的色彩模式。
BG_COLOR	背景色，在透明模式下使用。
BS_COLOR	蓝屏颜色，在透明模式下使用。
ROP_REG	透明模式有效/无效。
CMDR_1	写入该寄存器来开始一个存储器到屏的位块传送的绘制过程。如果设置了S位，图



	形将被压缩或者扩大，这取决于X-INCR和Y-INCR的值。
CMDR_2 / CMDR_3	主机通过这两个指令寄存器提供源图形数据。当主机将第一个32位数据写入CMDR_2，在主机到屏模式下，第一个绘制过程开始。接下来主机通过不断将数据写入CMDR_3来提供其余的数据。

### （3）彩色扩展（字型绘图）

彩色扩展将单色扩展到任意的背景(BG\_COLOR)色或者前景(FG\_COLOR)色。源数据呈现一个像素的每一位，“1”表示前景色，“0”表示背景色。位的顺序是从MSB到LSB。第一个数据的MSB调整到目标图形的最左上像素。该图形在下面用作彩色扩展的功能和数据类型的参考。例如，前景色是蓝色，背景色是白色，并且目标图形是16像素宽。如图18-6所示。

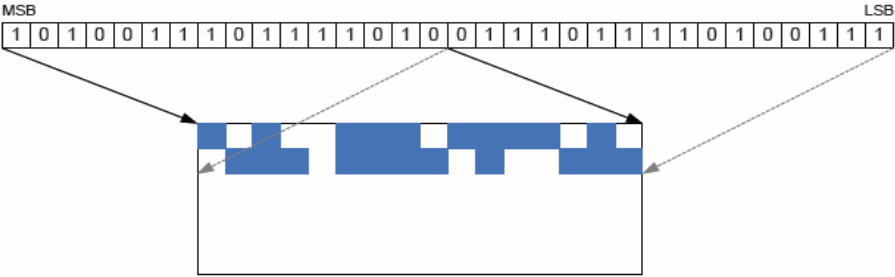


图18-6 彩色扩展（字型绘图）

FIMG-2D支持彩色扩展的主机到屏和存储器到屏模式。

相关寄存器如表18-3所示。

表 18-3 彩色扩展相关寄存器

COORD_0	目标窗口的最左上方坐标
COORD_1	目标窗口的最右下方坐标
FG_COLOR	前景色
BG_COLOR	背景色
CMDR_7	字型数据的基地址。在存储器到屏模式下，写入该寄存器开始绘制过程。
CMDR_4/CMDR_5	主机通过这两个指令寄存器提供字型数据。当主机将第一个32位数据写入CMDR_4，在主机到屏模式下，绘制过程开始。接下来，主机通过不断向CMDR_5写入数据来提供其余的数据

2. 光栅操作

根据用户指定的8位ROP值，光栅操作有三种操作数来执行布尔操作：源、目标和第三操作数。如表18-4所示。

表 18-4 操作数

源	目标	第三操作数	ROP值
1	1	1	位7
1	1	0	位6
1	0	1	位5
1	0	0	位4
0	1	1	位3
0	1	0	位2
0	0	1	位1
0	0	0	位0

第三操作数可以是图案或者前景色，通过配置ROP\_REG寄存器的OS来配置。

图案是用户指定的8×8×16bpp图形；图案数据应当在RGB565格式下给出。

下面的公式用来计算像素（x、y）的图案索引：

索引 = ( ((patternOffsetY + y) & 0x7 ) << 3 ) + ((patternOffsetX + x) & 0x7),

其中patternOffsetY和patternOffsetX是PATOFF\_REG寄存器指定的偏移量。

下面是一些关于怎样使用ROP值来进行操作的例子。

- (1) 最终数据 = 源。只有源数据相关， 因此， ROP 值 = “11110000”。
- (2) 最终数据 = 目标。只有最终数据相关，因此， ROP 值 = “11001100”。
- (3) 最终数据 = 图案。只有图案数据相关，因此， ROP值= “10101010”。
- (4) 最终数据 = 源和目标。ROP 值= “11110000” & “11001100” = “11000000”
- (5) 最终数据 =源或目标。 ROP 值 = “11110000” | “10101010” = “11111010”。

相关寄存器如表18-5所示。

表 18-5 光栅操作相关寄存器

PATTERN_REG[0:31]	图案数据
PATOFF_REG	图案偏移量X, Y
ROP_REG	ROP配置和ROP值

3. 蒙版测试（色度）

当启用蒙版测试时，蒙版测试模块检测DR（最小）和DR（最大）之间的相关图形的每个像素值。如果像素所在的每个区域（A、R、G和B）值在DR（最小）和DR（最大）范围内，则标记通过；如果有一个区域值不在该范围则失败。

如果要检测合格，则A、R、G和B的值必须在所给的值的范围内。例如，16位565、28位RGB和24位RGB格式没有A区域。但是，接受输入数据后，混合器h/w标志A区域为0。所以，如果s/w设置A\_DR(min)为非0值，所有的蒙版结果将失败。为了避免发生这种情况，如果s/w不检测特殊区域，则设置DR（最小）为0，DR（最大）为0xff。

如果DR（最小）和DR（最大）值相同，意味着蒙版测试应当使用固定值的方法。如果DR（最小）和DR（最大）值不同，混合器使用范围匹配的方法来做蒙版测试。反转蒙版操作如6-6表所示。

表 18-6 反转蒙版操作

蒙版启用	反转蒙版结果	动作
1	0	数据用于蒙版测试。如果像素值在DR（最小）和DR（最大）范围内，则混合器标记通过。否则，标记失败
1	1	数据用于蒙版测试。如果像素值在DR（最小）和DR（最大）范围内，则混合器标记失败。否则，标记通过

举例，如图18-7所示。

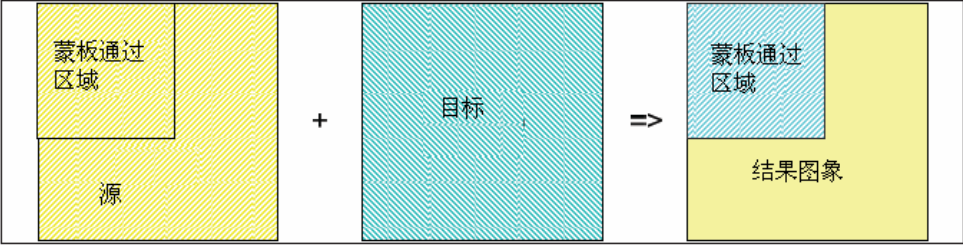


图18-7 蒙版测试 (色度)

4.  $\alpha$  混合

$\alpha$  混合在帧缓冲器合成源颜色和目标颜色来获得新的目标颜色。FIMG-2D支持256级用户指定的  $\alpha$  值和每个像素的  $\alpha$  混合，也支持衰退效应。

用户指定的  $\alpha$  值:  $\alpha$  (0~255)

[  $\alpha$  混合]

数据 = ( 源  $\times$  ( $\alpha + 1$ ) + 目标  $\times$  (256- $\alpha$ ) )  $\gg$  8

[衰退]

数据 = (( 源  $\times$  ( $\alpha + 1$ ) )  $\gg$  8) + 衰退偏移量

每个像素的  $\alpha$  混合:  $\alpha$  (由源图形给出, 0到255)

[  $\alpha$  混合]

数据 = 源  $\times$   $\alpha$  + 目标  $\times$  (1- $\alpha$ )

[衰退]

数据 = (源  $\times$   $\alpha$ ) + 衰退偏移量

相关寄存器如表18-7所示。

表 18-7  $\alpha$  混合相关寄存器

ROP_REG	$\alpha$ 混合配置: $\alpha$ 混合禁止/有效, 每个像素的 $\alpha$ 混合禁止/有效, 衰退禁止/有效
ALPHA_REG	$\alpha$ 值和衰退值

5. 旋转

像素可以围绕相关点 (ox, oy) 顺时针旋转90/180/270度, 或者在 (ox, oy) 线上围绕X轴/Y轴水平

或垂直旋转。旋转操作效果如表18-8和图18-8所示。

图 18-8 旋转效果

	0°	90°	180°	270°	绕X轴旋转	绕Y轴旋转
x	dcx	$-dcy + (ox+oy)$	$-dcx + 2ox$	$dcy + (ox-oy)$	dcx	$-dcx + 2ox$
y	dcy	$dcx - (ox-oy)$	$-dcy + 2oy$	$2oy -dcx +(ox+oy)$	$-dcy + 2oy$	dcy

相关寄存器如表18-9所示。

图 18-9 旋转相关寄存器

ROT_OC_REG	旋转相关点的坐标
ROTATE_REG	旋转模式配置

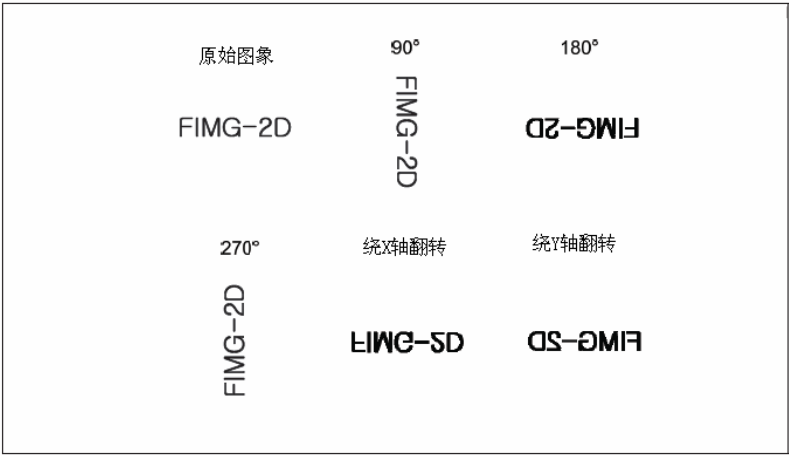


图18-8 旋转

具体的实现，ARM11中2D图形加速器的应用，关于旋转图形的部分代码如下：

```
void G2D_GetRotationOrgXY(u16 usSrcX1, u16 usSrcY1, u16 usSrcX2, u16 usSrcY2, u16 usDestX1, u16 usDestY1, ROT_DEG eRotDegree, u16* usOrgX, u16* usOrgY)
{
    G2D_CheckFifo(17);
    switch(eRotDegree)
    {
```

```

    case ROT_0:
        return;
    case ROT_90:
        *usOrgX = (usDestX1 - usDestY1 + usSrcX1 + usSrcY2)/2;
        *usOrgY = (usDestX1 + usDestY1 - usSrcX1 + usSrcY2)/2;
        break;
    case ROT_180:
        *usOrgX = (usDestX1 + usSrcX2)/2;
        *usOrgY = (usDestY1 + usSrcY2)/2;
        break;
    case ROT_270:
        *usOrgX = (usDestX1 + usDestY1 + usSrcX2 - usSrcY1)/2;
        *usOrgY = (usDestY1 - usDestX1 + usSrcX2 + usSrcY1)/2;
        break;
    default:
        Assert(0); // Unsupported Rotation Degree!
        break;
}

```

```

}

```

//旋转图形设置

```

void G2D_RotateImage(
    u16 usSrcX1, u16 usSrcY1, u16 usSrcX2, u16 usSrcY2,
    u16 usDestX1, u16 usDestY1, ROT_DEG eRotDegree)
{
    u16 usOrgX, usOrgY;
    u32 uRotDegree;

```

```

    G2D_GetRotationOrgXY(usSrcX1, usSrcY1, usSrcX2, usSrcY2, usDestX1, usDestY1, eRotDegree,

```

```
&usOrgX, &usOrgY);
```

```
G2D_CheckFifo(17);
```

```
Outp16(rG2D_ROT_OC_X, usOrgX);
```

```
Outp16(rG2D_ROT_OC_Y, usOrgY);
```

```
uRotDegree =
```

```
    (eRotDegree == ROT_0) ? G2D_ROTATION_0_DEG_BIT :
```

```
    (eRotDegree == ROT_90) ? G2D_ROTATION_90_DEG_BIT :
```

```
    (eRotDegree == ROT_180) ? G2D_ROTATION_180_DEG_BIT : G2D_ROTATION_270_DEG_BIT;
```

```
Outp32(rG2D_ROT_MODE, uRotDegree);
```

```
}
```

```
//BitBlt旋转设置
```

```
void G2D_RotateWithBitBlt(
```

```
    u16 usSrcX1, u16 usSrcY1, u16 usSrcX2, u16 usSrcY2,
```

```
    u16 usDestX1, u16 usDestY1,    ROT_DEG eRotDegree)
```

```
{
```

```
    u16 usOrgX, usOrgY;
```

```
    u32 uRotDegree;
```

```
    G2D_GetRotationOrgXY(usSrcX1, usSrcY1, usSrcX2, usSrcY2, usDestX1, usDestY1, eRotDegree,
```

```
&usOrgX, &usOrgY);
```

```
    G2D_CheckFifo(17);
```

```
    Outp16(rG2D_ROT_OC_X, usOrgX);
```

```
    Outp16(rG2D_ROT_OC_Y, usOrgY);
```

```
    uRotDegree =
```

```
        (eRotDegree == ROT_0) ? G2D_ROTATION_0_DEG_BIT :
```

```
        (eRotDegree == ROT_90) ? G2D_ROTATION_90_DEG_BIT :
```

```
        (eRotDegree == ROT_180) ? G2D_ROTATION_180_DEG_BIT : G2D_ROTATION_270_DEG_BIT ;
```

```
    Outp32(rG2D_ROT_MODE, uRotDegree);
```

```
G2D_BitBlt(usSrcX1, usSrcY1, usSrcX2, usSrcY2, usSrcX1, usSrcY1, usSrcX2, usSrcY2, false);  
}
```

6. 裁剪

裁剪将放弃裁剪窗口以外的像素（旋转后）。放弃的像素的色彩数据不写入帧缓冲器。  
相关寄存器如表18-10所示。

表18-10 裁剪相关寄存器

CW_LT_REG	裁剪窗口最左上方的点的坐标
CW_RB_REG	裁剪窗口最右下方的点的坐标

18. 4 2D 图形寄存器定义

地址映射

寄存器	偏移量	读/写	描述	复位值
通用寄存器				
INTEN_REG	0x76100004	读/写	中断有效寄存器。	0x0000_0000
FIFO_INTC_REG	0x76100008	读/写	中断控制寄存器。	0x0000_0018
INTC_PEND_REG	0x7610000C	读/写	中断控制等待寄存器。	0x0000_0000
FIFO_STAT_REG	0x76100010	读	指令FIFO状态寄存器。	0x0000_0600
FB_BA_REG	0x76100020	读/写	帧缓冲器基址寄存器。	0x0000_0000
指令寄存器				
CMD0_REG	0x76100100	写	用于线/点绘图的指令寄存器。	-
CMD1_REG	0x76100104	写	用于BitBLT的指令寄存器。	-
CMD2_REG	0x76100108	写	用于主机到屏Bitblt传送开始点的指令寄存器。	-
CMD3_REG	0x7610010C	写	用于主机到屏Bitblt传送恢复的指令寄存器。	-
CMD4_REG	0x76100110	写	用于彩色扩展的指令寄存器(主	-



			机到屏，字型开始点）。	
CMD5_REG	0x76100114	写	用于彩色扩展的指令寄存器(主机到屏，字型恢复)。	-
CMD7_REG	0x7610011C	写	用于彩色扩展的指令寄存器(存储器到屏)。	-
指令资源寄存器				
COLOR_MODE_REG	0x76100200	读/写	色彩模式寄存器。	0x0000_0008
HORI_REG_REG	0x76100204	读/写	水平分辨率寄存器。	0x0000_0000
SC_WIN_REG	0x76100210	读/写	屏幕裁剪窗口寄存器。	0x0000_0000
SC_WIN_X_REG	0x76100214	写	屏幕裁剪窗口的最大X寄存器。	0x0000_0000
SC_WIN_Y_REG	0x76100218	写	屏幕裁剪窗口的最大Y寄存器。	0x0000_0000
CW_LT_REG	0x76100220	读/写	裁剪窗口左上方坐标。	0x0000_0000
CW_LT_X_REG	0x76100224	写	裁剪窗口左X坐标。	0x0000_0000-
CW_LT_Y_REG	0x76100228	写	裁剪窗口上Y坐标。	0x0000_0000-
CW_RB_REG	0x76100230	读/写	裁剪窗口右下方坐标。	0x0000_0000
CW_RB_X_REG	0x76100234	写	裁剪窗口右X坐标。	0x0000_0000-
CW_RB_Y_REG	0x76100238	写	裁剪窗口下Y坐标。	0x0000_0000-
COORD0_REG	0x76100300	读/写	坐标0寄存器。	0x0000_0000
COORD0_X_REG	0x76100304	写	坐标0的X坐标。	0x0000_0000-
COORD0_Y_REG	0x76100308	写	坐标0的Y坐标。	0x0000_0000-
COORD1_REG	0x76100310	读/写	坐标1寄存器。	0x0000_0000
COORD1_X_REG	0x76100314	写	坐标1的X坐标。	0x0000_0000-
COORD1_Y_REG	0x76100318	写	坐标1的Y坐标。	0x0000_0000-
COORD2_REG	0x76100320	读/写	坐标2寄存器。	0x0000_0000
COORD2_X_REG	0x76100324	写	坐标2的X坐标。	0x0000_0000-
COORD2_Y_REG	0x76100328	写	坐标2的Y坐标。	0x0000_0000-
COORD3_REG	0x76100330	读/写	坐标3寄存器。	0x0000_0000
COORD3_X_REG	0x76100334	写	坐标3的X坐标。	0x0000_0000-

COORD3_Y_REG	0x76100338	写	坐标3的Y坐标。	0x0000_0000-
ROT_OC_REG	0x76100340	读/写	旋转原点坐标。	0x0000_0000
ROT_OC_X_REG	0x76100344	写	旋转原点坐标的X坐标。	0x0000_0000-
ROT_OC_Y_REG	0x76100348	写	旋转原点坐标的Y坐标。	0x0000_0000-
ROTATE_REG	0x7610034C	读/写	旋转模式寄存器。	0x0000_0001
ENDIA_READSIZE	0x76100350	读/写	大端或小端选择，读取的顺序。	0x0000_0001
X_INCR_REG	0x76100400	读/写	X增量寄存器。	0x0000_0000
Y_INCR_REG	0x76100404	读/写	Y增量寄存器。	0x0000_0000
ROP_REG	0x76100410	读/写	光栅操作寄存器。	0x0000_0000
ALPHA_REG	0x76100420	读/写	$\alpha$ 值，衰退偏移量。	0x0000_0000
FG_COLOR_REG	0x76100500	读/写	前景色/ $\alpha$ 寄存器。	0x0000_0000
BG_COLOR_REG	0x76100504	读/写	背景色寄存器。	0x0000_0000
BS_COLOR_REG	0x76100508	读/写	蓝屏颜色寄存器。	0x0000_0000
PATTERN_REG	0x76100600 ~0x7610067C	读/写	图案寄存器。	-
PATOFF_REG	0x76100700	读/写	图案偏移量XY寄存器。	0x0000_0000
PATOFF_X_REG	0x76100704	写	图案偏移量X寄存器。	0x0000_0000
PATOFF_Y_REG	0x76100708	写	图案偏移量Y寄存器。	0x0000_0000

#### 18. 4. 1. 通用中断有效寄存器（INTEN\_REG）

寄存器	偏移量	读/写	描述	复位值
INTEN_REG	0x76100004	读/写	中断有效寄存器。	0x0000_0000

INTEN_REG	位	描述	初始状态
Reserved	[31:10]		0x0
DF	[10]	绘图引擎完成中断有效。	0x0
F	[9]	所有的指令完成中断有效。当所有指令执行后（指令FIFO中没有	0x0

		指令)，设置该位。	
OV	[8]	溢出中断有效。当指令FIFO溢出，该位被设置。	0x0
Reserved	[7:1]		0x0
E	[0]	FIFO电平中断有效。如果该位设置为1，当FIFO_INT_LEVEL和FIFO_NO_USED相同时，图形引擎将INTREQ信号标志为高位。	0x0

#### 18.4.2. 通用 FIFO 中断控制寄存器(FIFO\_INTC\_REG)

寄存器	偏移量	读/写	描述	复位值
FIFO_INTC_REG	0x76100008	读/写	中断控制寄存器。	0x0000_0018

FIFO_INTC_REG	位	描述	初始值
Reserved	[31:6]		0x0
FIFO_INT_LEVEL	[5:0]	当 FIFO 的数目的使用是 FIFO_INT_LEVEL，中断启动位被设置为 1 时，图形引擎请求中断。	0x18

#### 18.4.3. 通用断控制等待寄存器(INT\_PEND\_REG)

寄存器	偏移量	读/写	描述	复位值
INTC_PEND_REG	0x7610000C	读/写	中断控制等待寄存器。	0x0000_0000

INTC_PEND_REG	位	描述	初始值
CLRSEL	[31]	水平中断&脉冲中断模式选择： 1：水平模式选择（中断清除启动） 0：脉冲中断模式选择	
Reserved	30:11]		
INTP_DE_FIN	[10]	图形绘图引擎完成。	
INTP_FINISH_ALL	[9]	图形引擎空闲状态。	
INTP_OVERFLOW	[8]	溢出中断。	

Reserved	[7:1]		
INTP_FIFO_LEVEL	[0]	当 FIFO_INT_LEVEL 与 FIFO_NO_USED 一样时，这个位将被设置。	

#### 18.4.4. 通用指令 FIFO 状态寄存器(FIFO\_STAT\_REG)

寄存器	偏移量	读/写	描述	复位值
FIFO_STAT_REG	0x76100010	读	指令 FIFO 状态寄存器。	0x0000_0600

FIFO_STAT_REG	位	描述	初始值
Reserved	[31:13]		0x0
DE_FIN	[10]	图形绘图引擎完成。	1
G2D_IDLE	[9]	图形引擎空闲状态。	1
OVR_INT	[8]	溢出中断。	0
Reserved	[7:6]		0x0
FIFO_NO_USED	[5:0]	使用的 FIFO 数目。	0x0

#### 18.4.5. 通用帧缓冲器基址寄存器(FB\_BA\_REG)

寄存器	偏移量	读/写	描述	复位值
FB_BA_REG	0x76100020	读/写	帧缓冲器基址寄存器。	0x0000_0000

FB_BA_REG	位	描述	初始值
FrameBufAddr	[31:10]	帧缓冲器地址上面的 22 位。8 位最高有效位（MSB）决定着帧缓冲器的上限。例如，如果用户设置帧缓冲器地址为 0x60800000，则最大存储器分配帧缓冲器是 [0x60800000, 0x60FFFFFF]。	0x0
Reserved	[9:0]	最低为 10 的帧缓冲器地址的位强制置为 0，这意味帧缓冲器地址应该定位到 1KB。	0x0

18.4.6. 指令线绘图寄存器 (CMD0\_REG)

寄存器	偏移量	读/写	描述	复位值
CMD0_REG	0x76100100	写	用于线/点绘图寄存器。	–

CMD0_REG	位	描述	初始值
保留	[31:10]		–
D	[9]	0: 绘图最后的点 1: 未绘图最后的点	–
M	[8]	0: 主轴是 Y 1: 主轴是 X	–
Reserved	[7:2]		–
L	[1]	0: 无关 1: 线绘图	–
P	[0]	0: 无关 1: 点绘图	–

18.4.7. 指令 BITBLT 寄存器 (CMD1\_REG)

寄存器	偏移量	读/写	描述	复位值
CMD1_REG	0x76100104	写	用于 BitBLT 的指令寄存器。	–

CMD1_REG	位	描述	初始值
Reserved	[31:2]		–
S	[1]	0: 无关 1: 伸展 BitBLT	–
N	[0]	0: 无关 1: 标准 BitBLT	–

18.4.8. 指令主机到屏幕优先 BITBLT 寄存器 (CMD2\_REG)

寄存器	偏移量	读/写	描述	复位值
CMD2_REG	0x76100108	写	用于主机到屏幕 Bitblt 传输优先的指令寄存器。	–

CMD2_REG	位	描述	初始值
Data	[31:0]	BitBLT 优先数据。	–

18.4.9. 指令主机到屏幕连续 BITBLT 寄存器 (CMD3\_REG)

寄存器	偏移量	读/写	描述	复位值
CMD3_REG	0x7610010C	写	用于主机到屏幕 Bitblt 传输连续的指令寄存器。	–

CMD3_REG	位	描述	初始值
Data	[31:0]	BitBLT 连续数据。	–

18.4.10. 指令主机到屏幕优先颜色扩充寄存器 (CMD4\_REG)

寄存器	偏移量	读/写	描述	复位值
CMD4_REG	0x76100110	写	用于颜色扩充的指令寄存器（主机到屏幕，字体优先）。	–

CMD4_REG	位	描述	初始值
Data	[31:0]	压缩格式位图数据。	–

18. 4. 11. 指令主机到屏幕连续颜色扩充寄存器 (CMD5\_REG)

寄存器	偏移量	读/写	描述	复位值
CMD5_REG	0x76100114	写	用于颜色扩充的指令寄存器（主机到屏幕，字体连续）。	-

CMD5_REG	位	描述	初始值
Data	[31:0]	压缩格式位图数据。	-

18. 4. 12. 指令内存到屏幕颜色扩充寄存器 (CMD7\_REG)

寄存器	偏移量	读/写	描述	复位值
CMD7_REG	0x7610011C	写	用于颜色扩充的指令寄存器（内存到屏幕）。	-

CMD7_REG	位	描述	初始值
Reserved	[31:25]		-
Memory Address	[24:2]	字定位地址到点位图数据。	-
Reserved	[1:0]		-

18. 4. 13. 指令资源颜色模式 (COLOR\_MODE\_REG)

寄存器	偏移量	读/写	描述	复位值
COLOR_MODE_REG	0x76100200	读/写	色彩模式寄存器。	0x0000_0008

COLOR_MODE_REG	位	描述	初始值
Reserved	[31:4]		0x0
C3	[3]	24/32	0x1
C2	[2]	18	0x0

C1	[1]	16	0x0
C0	[0]	15	0x0

#### 18.4.14. 指令资源水平分辨率(HORI\_RES\_REG)

寄存器	偏移量	读/写	描述	复位值
HORI_REG_REG	0x76100204	读/写	水平分辨率寄存器。	0x0000_0000

HORI_RES_REG	位	描述	初始值
Reserved	[31: 12]		0x0
HoriRes	[11: 0]	水平分辨率（应为 4 的倍数）。	0x0

#### 18.4.15. 指令资源屏幕剪切窗口(SC\_WIN\_REG)

寄存器	偏移量	读/写	描述	复位值
SC_WIN_REG	0x76100210	读/写	屏幕剪切窗口寄存器。	0x0000_0000

SC_WIN_REG	位	描述	初始值
Reserved	[31:27]		0x0
MaxSY	[26:16]	最大屏幕剪切 Y 窗口。	0x0
Reserved	[15:11]		0x0
MaxSX	[10:0]	最大屏幕剪切 X 窗口。	0x0

#### 18.4.16. 指令资源屏幕剪切最大 X 窗口(SC\_WIN\_X\_REG)

寄存器	偏移量	读/写	描述	复位值
SC_WIN_X_REG	0x76100214	写	屏幕剪切的 X 窗口寄存器。	0x0000_0000



SC_WIN_X_REG	位	描述	初始值
Reserved	[31:11]		0x0
MaxSX	[10:0]	最大屏幕剪切 X 窗口。	0x0

18. 4. 17. 指令资源屏幕剪切最大 X 窗口 (SC\_WIN\_X\_REG)

寄存器	偏移量	读/写	描述	复位值
SC_WIN_Y_REG	0x76100218	写	屏幕剪切的最大的 Y 窗口寄存器。	0x0000_0000

SC_WIN_Y_REG	位	描述	初始值
Reserved	[31:11]		0x0
MaxSY	[10:0]	最大屏幕剪切 Y 窗口。	0x0

18. 4. 18. 指令资源剪切窗口左端 (CW\_LT\_REG)

寄存器	偏移量	读/写	描述	复位值
CW_LT_REG	0x76100220	读/写	剪切窗口的左端坐标。	0x0000_0000

CW_LT_REG	位	描述	初始值
Reserved	[31:27]		0x0
TopCW_Y	[26:16]	Y 端剪切窗口。	0x0
Reserved	[15:11]		0x0
LeftCW_X	[10:0]	X 左端剪切窗口。	0x0

COMMON RESOURCE LEFT X CLIPPING WINDOW

18. 4. 19. 指令资源 X 左端剪切窗口 (CW\_LT\_X\_REG)

寄存器	偏移量	读/写	描述	复位值
CW_LT_X_REG	0x76100224	写	剪切窗口的 X 左端坐标。	0x0000_0000

CW_LT_X_REG	位	描述	初始值
Reserved	[31:11]		0x0
LeftCW_X	[10:0]	X 左端剪切窗口	0x0

18. 4. 20. 指令资源 Y 左端剪切窗口 (CW\_LT\_Y\_REG)

寄存器	偏移量	读/写	描述	复位值
CW_LT_Y_REG	0x76100228	写	剪切窗口的 Y 左端坐标。	0x0000_0000

CW_LT_Y_REG	位	描述	初始值
Reserved	[31:11]		0x0
LeftCW_Y	[10:0]	Y 左端剪切窗口	0x0

18. 4. 21. 指令资源右底部剪切窗口 (CW\_RB\_REG)

寄存器	偏移量	读/写	描述	复位值
CW_RB_REG	0x76100230	读/写	剪切窗口的右底部坐标。	0x0000_0000

CW_RB_REG	位	描述	初始值
Reserved	[31:27]		0x0
BottomCW_Y	[26:16]	底部 Y 剪切窗口。	0x0
Reserved	[15:11]		0x0
RightCW_X	[10:0]	右底部 X 剪切窗口。	0x0

18. 4. 22. 指令资源右端 X 剪切窗口 (CW\_RB\_X\_REG)

寄存器	偏移量	读/写	描述	复位值
CW_RB_X_REG	0x76100234	写	剪切窗口的右 X 坐标。	0x0000_0000

CW_RB_X_REG	位	描述	初始值
Reserved	[31:11]		0x0
RightCW_X	[10:0]	右端 X 剪切窗口。	0x0

18. 4. 23. 指令资源底端 Y 剪切窗口 (CW\_RB\_Y\_REG)

寄存器	偏移量	读/写	描述	复位值
CW_RB_Y_REG	0x76100238	写	剪切窗口的底端 Y 坐标。	0x0000_0000

CW_RB_Y_REG	位	描述	初始值
Reserved	[31:11]		0x0
RightCW_Y	[10:0]	底端 Y 剪切窗口。	0x0

18. 4. 24. 指令资源坐标 0 寄存器 (COORD0\_REG)

寄存器	偏移量	读/写	描述	复位值
COORD0_REG	0x76100300	读/写	坐标 0 寄存器。	0x0000_0000

COORD0_REG	位	描述	初始值
Reserved	[31:27]		0x0
Y	[26:16]	坐标 0 Y。	0x0
Reserved	[15:11]		0x0
X	[10:0]	坐标 0 X。	0x0

18. 4. 25. 指令资源坐标 0 X 寄存器(COORD0\_X\_REG)

寄存器	偏移量	读/写	描述	复位值
COORD0_X_REG	0x76100304	写	坐标 0 的 X 坐标。	0x0000_0000

COORD0_X_REG	位	描述	初始值
Reserved	[31:11]		0x0
COORD0_X	[10:0]	坐标 0 X。	0x0

18. 4. 26. 指令资源坐标 0 Y 寄存器(COORD0\_Y\_REG)

寄存器	偏移量	读/写	描述	复位值
COORD0_Y_REG	0x76100308	写	坐标 0 的 Y 坐标。	0x0000_0000

COORD0_Y_REG	位	描述	初始值
Reserved	[31:11]		0x0
COORD0_Y	[10:0]	坐标 0 Y。	0x0

18. 4. 27. 指令资源坐标 1 寄存器(COORD1\_REG)

寄存器	偏移量	读/写	描述	复位值
COORD1_REG	0x76100310	读/写	坐标 1 寄存器。	0x0000_0000

COORD1_REG	位	描述	初始值
Reserved	[31:27]		0x0
Y	[26:16]	坐标 1 Y。	0x0
Reserved	[15:11]		0x0
X	[10:0]	坐标 1 X。	0x0

18. 4. 28. 指令资源坐标 1 X 寄存器(COORD1\_X\_REG)

寄存器	偏移量	读/写	描述	复位值
COORD1_X_REG	0x76100314	写	坐标 1 的 X 坐标。	0x0000_0000

COORD1_X_REG	位	描述	初始值
Reserved	[31:11]		0x0
COORD1_X	[10:0]	坐标 1 X。	0x0

18. 4. 29. 指令资源坐标 1 Y 寄存器(COORD1\_Y\_REG)

寄存器	偏移量	读/写	描述	复位值
COORD1_Y_REG	0x76100318	写	坐标 1 的 Y 坐标。	0x0000_0000

COORD1_Y_REG	位	描述	初始值
Reserved	[31:11]		0x0
COORD1_Y	[10:0]	坐标 1 Y。	0x0

18. 4. 30. 指令资源坐标 2 寄存器(COORD2\_REG)

寄存器	偏移量	读/写	描述	复位值
COORD2_REG	0x76100320	读/写	坐标 2 寄存器。	0x0000_0000

COORD2_REG	位	描述	初始值
Reserved	[31:27]		0x0
Y	[26:16]	坐标 2 Y。	0x0
Reserved	[15:11]		0x0
X	[10:0]	坐标 2 X。	0x0

18. 4. 31. 指令资源坐标 2 X 寄存器 (COORD2\_X\_REG)

寄存器	偏移量	读/写	描述	复位值
COORD2_X_REG	0x76100324	写	坐标 2 的 X 坐标。	0x0000_0000

COORD2_X_REG	位	描述	初始值
Reserved	[31:11]		0x0
COORD2_X	[10:0]	坐标 2 X	0x0

18. 4. 32. 指令资源坐标 2 Y 寄存器 (COORD2\_Y\_REG)

寄存器	偏移量	读/写	描述	复位值
COORD2_Y_REG	0x76100328	写	坐标 2 的 Y 坐标。	0x0000_0000

COORD2_Y_REG	位	描述	初始值
Reserved	[31:11]		0x0
COORD2_Y	[10:0]	坐标 2 Y。	0x0

18. 4. 33. 指令资源坐标 3 寄存器 (COORD3\_REG)

寄存器	偏移量	读/写	描述	复位值
COORD3_REG	0x76100330	读/写	坐标 3 寄存器。	0x0000_0000

COORD3_REG	位	描述	初始值
Reserved	[31:27]		0x0
Y	[26:16]	坐标 3 Y。	0x0
Reserved	[15:11]		0x0
X	[10:0]	坐标 3 X。	0x0

18. 4. 34. 指令资源坐标 3 X 寄存器(COORD3\_X\_REG)

寄存器	偏移量	读/写	描述	复位值
COORD3_X_REG	0x76100334	写	坐标 3 的 X 坐标。	0x0000_0000

COORD3_X_REG	位	描述	初始值
Reserved	[31:11]		0x0
COORD3_X	[10:0]	坐标 3 X。	0x0

18. 4. 35. 指令资源坐标 3 Y 寄存器(COORD3\_Y\_REG)

寄存器	偏移量	读/写	描述	复位值
COORD3_Y_REG	0x76100338	写	坐标 3 的 Y 坐标。	0x0000_0000

COORD3_Y_REG	位	描述	初始值
Reserved	[31:11]		0x0
COORD3_Y	[10:0]	坐标 3 Y。	0x0

18. 4. 36. 指令资源旋转原点坐标(ROT\_OC\_REG)

寄存器	偏移量	读/写	描述	复位值
ROT_OC_REG	0x76100340	读/写	旋转原点坐标。	0x0000_0000

ROT_OC_REG	位	描述	初始值
Reserved	[31:27]		0x0
Y	[26:16]	坐标 3 Y。	0x0
Reserved	[15:11]		0x0
X	[10:0]	坐标 3 X。	0x0

18. 4. 37. 指令资源旋转坐标 X (ROT\_OC\_X\_REG)

寄存器	偏移量	读/写	描述	复位值
ROT_OC_X_REG	0x76100344	写	旋转原点坐标的 X 坐标。	0x0000_0000

ROT_OC_X_REG	位	描述	初始值
Reserved	31:11]		0x0
ROT_OC_X	[10:0]	旋转原点坐标 X。	0x0

18. 4. 38. 指令资源旋转坐标 Y (ROT\_OC\_Y\_REG)

寄存器	偏移量	读/写	描述	复位值
ROT_OC_Y_REG	0x76100348	写	旋转原点坐标的 Y 坐标。	0x0000_0000

ROT_OC_Y_REG	位	描述	初始值
Reserved	31:11]		0x0
ROT_OC_Y	[10:0]	旋转原点坐标 Y。	0x0

COMMON RESOURCE ROTATION REGISTER

18. 4. 39. 指令资源旋转寄存器 (ROTATE\_REG)

寄存器	偏移量	读/写	描述	复位值
ROTATE_REG	0x7610034C	读/写	旋转模式寄存器。	0x0000_0001

ROTATE_REG	位	描述	初始值
Reserved	31:6]		0x0
FY	[5]	垂直翻转。	0x0
FX	[4]	水平翻转。	0x0



R3	[3]	270 度旋转。	0x0
R2	[2]	180 度旋转。	0x0
R1	[1]	90 度旋转。	0x0
R0	[0]	0 度旋转。	0x0

\*如果两个或两个以上的 Rn 在同一时间被设置为 1，绘图引擎的操作将不可预测。

#### 18.4.40. 指令资源读尺寸(ENDIAN\_READSIZE)

寄存器	偏移量	读/写	描述	复位值
ENDIA_READSIZE	0x76100350	读写	读尺寸选择。	0x0000_0001

ENDIA_READSIZE	位	描述	初始值
Reserved	[31:5]		0x0
Reserved	[4]	可以是 ‘0’ 。	0x0
Reserved	[3]		0x0
SIZE_HW	[2]	1’ b0: h/w 设置读缓冲器大小禁止。 1’ b1: h/w 设置读缓冲器大小启动。	0x0
READ_SIZE	[1:0]	2’ b00: 读脉冲大小 = 1 。 2’ b01: 读脉冲大小 = 4 。 2’ b10: 读脉冲大小 = 8 。 2’ b11: 读脉冲大小 = 16 。	0x0

#### 18.4.41. 指令资源 X 增量寄存器 (X\_INCR\_REG)

寄存器	偏移量	读/写	描述	复位值
X_INCR_REG	0x76100400	读/写	X 增量寄存器。	0x0000_0000

X_INCR_REG	位	描述	初始值
Reserved	[31:22]		0x0

X_INCR	[21:0]	X 增量值。	0x0
--------	--------	--------	-----

#### 18. 4. 42. 指令资源 Y 增量寄存器 (Y\_INCR\_REG)

寄存器	偏移量	读/写	描述	复位值
Y_INCR_REG	0x76100404	读/写	Y 增量寄存器。	0x0000_0000

Y_INCR_REG	位	描述	初始值
Reserved	[31:22]		0x0
Y_INCR	[21:0]	Y 增量值。	0x0

#### 18. 4. 43. 指令资源屏面操作寄存器 (ROP\_REG)

寄存器	偏移量	读/写	描述	复位值
ROP_REG	0x76100410	读/写	屏面操作寄存器。	0x0000_0000

ROP_REG	位	描述	初始值
Reserved	[31:14]		0x0
OS	[13]	第三操作数选择： 1' b0: 图案 1' b1: 前景颜色	0x0
ABM	[12:10]	Alpha 连接模式： 3' b000: 无 Alpha 连接 3' b001: 像素 Alpha 与源位图连接 3' b010: Alpha 与寄存器连接 3' b011: 像素 Alpha 与第三位图连接 3' b100: 衰减 其它: 保留。	0x0
T	[9]	0: 不透明模式	0x0

		1: 透明模式	
Reserved	[8]		0x0
ROP	[7:0]	屏面操作值。	0x0

18. 4. 44. 指令资源 ALPHA 寄存器 (ALPHA\_REG)

寄存器	偏移量	读/写	描述	复位值
ALPHA_REG	0x76100420	读/写	Alpha 值，衰减偏移。	0x0000_0000

ALPHA_REG	位	描述	初始值
Reserved	[31:16]		0x0
Fading	[15:8]	衰减偏移值。	0x0
Alpha	[7:0]	Alpha 值。	0x0

18. 4. 45. 指令资源前景颜色寄存器 (FG\_COLOR\_REG)

寄存器	偏移量	读/写	描述	复位值
FG_COLOR_REG	0x76100500	读/写	前景颜色/Alpha 寄存器。	0x0000_0000

FG_COLOR_REG	位	描述	初始值
Reserved	[31:24]		0x0
ForegroundColor	[23:0]	前景颜色值。	0x0

18. 4. 46. 指令资源背景颜色寄存器 (BG\_COLOR\_REG)

寄存器	偏移量	读/写	描述	复位值
BG_COLOR_REG	0x76100504	读/写	背景颜色/Alpha 寄存器。	0x0000_0000

BG_COLOR_REG	位	描述	初始值
--------------	---	----	-----

Reserved	[31:24]		0x0
BackgroundColor	[23:0]	背景颜色值。	0x0

#### 18. 4. 47. 指令资源蓝色屏幕颜色寄存器 (BS\_COLOR\_REG)

寄存器	偏移量	读/写	描述	复位值
BS_COLOR_REG	0x76100508	读/写	蓝色屏幕颜色寄存器。	0x0000_0000

BS_COLOR_REG	位	描述	初始值
Reserved	[31:24]		0x0
BlueScreenColor	[23:0]	蓝色屏幕颜色值。	0x0

#### 18. 4. 48. 图案存储器 (PATTERN\_REG)

寄存器	偏移量	读/写	描述	复位值
PATTERN_REG	0x76100600 ~0x7610067C	读/写	图案存储器。	–

PATTERN_REG	位	描述	初始值
Pattern memory	[31:0]	图案存储器	

#### 18. 4. 49. 指令资源图案偏移寄存器 (PATOFF\_REG)

寄存器	偏移量	读/写	描述	复位值
PATOFF_REG	0x76100700	读/写	图案偏移 XY 寄存器。	0x0000_0000

PATOFF_REG	位	描述	初始值
Reserved	[31:19]		0x0

POffsetY	[18:16]	图案偏移 Y 值。	0x0
Reserved	[15:3]		0x0
POffsetX	[2:0]	图案偏移 X 值。	0x0

#### 18. 4. 50. 指令资源图案偏移 X 寄存器 (PATOFF\_X\_REG)

寄存器	偏移量	读/写	描述	复位值
PATOFF_X_REG	0x76100704	写	图案偏移 X 寄存器。	0x0000_0000

PATOFF_X_REG	位	描述	初始值
Reserved	[31:3]		0x0
POffsetX	[2:0]	图案偏移 X 值。	0x0

#### 18. 4. 51. 指令资源图案偏移 Y 寄存器 (PATOFF\_Y\_REG)

寄存器	偏移量	读/写	描述	复位值
PATOFF_Y_REG	0x76100708	写	图案偏移 Y 寄存器。	0x0000_0000

PATOFF_Y_REG	位	描述	初始值
Reserved	[31:3]		0x0
POffsetY	[2:0]	图案偏移 Y 值。	0x0

#### 18. 4. 52. 指令资源蒙版控制寄存器 (STENCIL\_CNTL)

寄存器	偏移量	读/写	描述	复位值
STENCIL_CNTL_REG	0x76100720	写	蒙版控制寄存器。	0x0000_0000

STENCIL_CNTL_REG	位	描述	初始值
Stencil_on	[31]	1: 蒙版打开	0x0

Reserved	[30:24]		0x0
stencil _inverse	[23]	1: 反转蒙版	0x0
Reserved	[22:1]		0x0
stencil _swap	[0]	交换输入数据。	0x0

18. 4. 53. 指令资源蒙版确定参考最小寄存器 (STENCIL \_DR\_MIN)

寄存器	偏移量	读/写	描述	复位值
STENCIL_DR_MIN_REG	0x76100724	写	蒙版确定参考最小寄存器。	0x0000_0000

STENCIL_DR_MIN_REG	位	描述	初始值
Reserved	[31:24]		0x0
R_DR (min)	[23:16]	红色 DR 最小值。	0x0
G_DR (min)	[15:8]	绿色色 DR 最小值。	0x0
B_DR (min)	[7:0]	蓝色色 DR 最小值。	0x0

18. 4. 54. 指令资源蒙版确定参考最大寄存器 (STENCIL \_DR\_MAX)

寄存器	偏移量	读/写	描述	复位值
STENCIL_DR_MAX_REG	0x76100728	写	蒙版确定参考最小寄存器。	0xFFFF_FFFF

STENCIL_DR_MAX_REG	位	描述	初始值
Reserved	[31:24]		0xF
R_DR (max)	[23:16]	红色 DR 最大值。	0xF
G_DR (max)	[15:8]	绿色色 DR 最大值。	0xF
B_DR (max)	[7:0]	蓝色色 DR 最大值。	0xF

注：在点/线绘图时支持蒙版操作。

# 19 图像旋转器

## 19.1 概述

图像旋转器运行旋转/翻转图像数据。有旋转 FSM, 旋转缓冲区, AMBA AB 2.0 主/从接口和寄存器。主要性能如下:

- 1.支持图像格式: YCbCr 4:2:2(交错), YCbCr 4:2:0(不交错), RGB565 和 RGB888 (未打包)
- 2.支持选择程度: 90, 180, 270, 垂直翻转和水平翻转。

## 19.2 模块图和图像实例

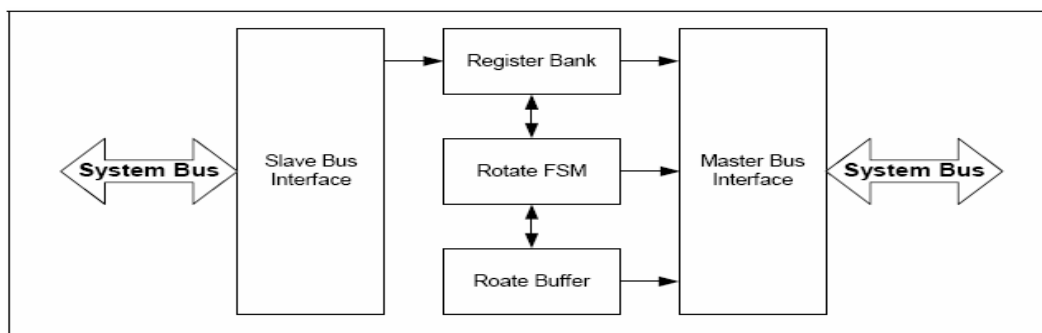


图 19-1 图像旋转器模块图



图 19-2 图像实例

## 19.3 寄存器描述

### 1.内存映射

寄存器	地址	读/写	描述	复位值
CTRLCFG	0x7720_0000	读/写	旋转器控制寄存器	0x0000_0000
SRCADDRREG0	0x7720_0004	读/写	旋转器源图像（RGB 或 Y 组成部分）地址寄存器	0x0000_0000
SRCADDRREG1	0x7720_0008	读/写	旋转器源图像（CB 组成部分）地址寄存器	0x0000_0000
SRCADDRREG2	0x7720_000C	读/写	旋转器源图像（CR 组成部分）地址寄存器	0x0000_0000



SRCsizereg	0x7720_0010	读/写	旋转器源图像尺寸寄存器	0x0000_0000
DESTADDRREG0	0x7720_0018	读/写	旋转器目标图像（RGB 或 Y 成分）地址寄存器	0x0000_0000
DESTADDRREG1	0x7720_001C	读/写	旋转器目标图像（CB 组成部分）地址寄存器	0x0000_0000
DESTADDRREG2	0x7720_0020	读/写	旋转器目标图像（CR 组成部分）地址寄存器	0x0000_0000
STATCFG	0x7720_002C	读	旋转器状态寄存器	0x0000_0000

2.旋转器控制寄存器

寄存器	地址	读/写	描述	复位值
CTRLCFG	0x7720_0000	读/写	旋转器控制寄存器	0x0000_0000

CTRLCFG	位	描述	初始状态
Reserved	[31:25]	保留	000b
Enable int	[24]	中断使能 0:disable 中断                      1:enable 中断	0b
Reserved	[23:16]	保留	0x00
输入图像格式	[15:13]	输入图像格式被旋转 000: YCbCr 4:2:0(不交错) 001: 保留 010: 保留                              011: YCbCr 4:2:2(交错) 100: RBG565                              101:RGB888(未打包)	000b
Reserved	[12:8]	保留	0_0000b
旋转程度	[7:6]	决定旋转程度 00: 不旋转                              01: 90 度 10: 180 度                              11: 270 度 注: 如果翻转程度不为 0, 这些位应该为 0.	00b
翻转方向	[5:4]	决定翻转方向 00: 不翻转	00b

		10: 垂直翻转                      11: 水平翻转 注：如果旋转程度不为 0，这些位应该为 0.	
Reserved	[3:1]	保留	000b
开始旋转	[0]	旋转使能信号，当此位设置时，旋转器开始运行。当选择区开始移动图像时，此位被清零。  0: 不工作                      1: 开始选择运行	0b

**3. 旋转器源图像地址寄存器 0（RGB 或 Y 组成部分）**

寄存器	地址	读/写	描述	复位值
SRCADDRREG0	0x7720_0004	读/写	旋转器源图像地址寄存器	0x0000_0000

SRCADDRREG0	位	描述	初始状态
源地址	[30:0]	源图像地址	000b

**4. 旋转器源图像地址寄存器 1（CB 组成部分）**

寄存器	地址	读/写	描述	复位值
SRCADDRREG1	0x7720_0008	读/写	旋转器源图像地址寄存器	0x0000_0000

SRCADDRREG1	位	描述	初始状态
源地址	[30:0]	源图像地址	000b

**5. 旋转器源图像地址寄存器 2（CR 组成部分）**

寄存器	地址	读/写	描述	复位值
SRCADDRREG2	0x7720_000C	读/写	旋转器源图像地址寄存器	0x0000_0000

SRCADDRREG2	位	描述	初始状态
源地址	[30:0]	源图像地址	000b

**6. 旋转器源图像尺寸寄存器**

寄存器	地址	读/写	描述	复位值
SRCSIZEREG	0x7720_0010	读/写	旋转器源图像尺寸寄存器	0x0000_0000

SRCSIZEREG	位	描述	初始状态
垂直尺寸	[30:16]	源图像的垂直尺寸 垂直图像尺寸=this value	0x0000
水平尺寸	[15: 0]	源图像的水平尺寸 水平图像尺寸=this value	0x0000

**7. 旋转器目标图像地址寄存器 0（RGB 或 Y 组成部分）**

寄存器	地址	读/写	描述	复位值
DESTADDRREG0	0x7720_0018	读/写	旋转器目标图像地址寄存器	0x0000_0000

DESTADDRREG0	位	描述	初始状态
目标地址	[30:0]	目标图像地址	000b

**8. 旋转器源图像地址寄存器 1（CB 组成部分）**

寄存器	地址	读/写	描述	复位值
DESTADDRREG1	0x7720_001C	读/写	旋转器目标图像地址寄存器	0x0000_0000

DESTADDRREG1	位	描述	初始状态
目标地址	[30:0]	目标图像地址	000b

**9. 旋转器目标图像地址寄存器 2（CR 组成部分）**

寄存器	地址	读/写	描述	复位值
DESTADDRREG2	0x7720_0020	读/写	旋转器目标图像地址寄存器	0x0000_0000

DESTADDRREG2	位	描述	初始状态
目标地址	[30:0]	目标图像地址	000b

### 10. 旋转器状态寄存器

寄存器	地址	读/写	描述	复位值
STATCFG	0x7720_002C	读/写	旋转器状态寄存器	0x0000_0000

STATCFG	位	描述	初始状态
当前行数	[31:16]	指明旋转器从哪访问图像。这个值指出已处理图像的行数。	0
Reserved	[15:9]	保留	0x00
中断悬挂	[8]	此位设置什么时候完成一个图像的选择。读入数据将会清零此位。	0
Reserved	[7:2]	保留	0x00
旋转器状态	[1:0]	此位指出旋转器的操作类型。  00: 不工作 (IDLE)      01:保留  10: 旋转一个图像 (BUSY)  11:旋转图像, 并且有更多的工作等待完成	0

## 20 相机接口

### 20.1 概述

本节主要定义了相机接口。S3C6410X 内的相机接口支持 ITU R BT-601/656 YCbCr 8 位标准。最大输入尺寸为 4096x4096 像素。S3C6410X 内的相机接口由不同功能组成。T\_patternMux 是测试样板发生器。测试样板可以用来校准输入同步信号作为 HREF 和 VSYNC。CatchCam 是捕捉 ITU 信号和窗口剪切。视频同步信号和像素时钟极性可以用寄存器设置反接在相机接口部分。相机接口内存在两个定标器，一个是预览定标器，专门用来产生比较小的图像，用于预览。另一个是编解码定标器，专用来产生编解码用途的信号。相机接口内存在两个输出 DMA，一个是预览 DMA，另一个是编解码 DMA。两个 DMA 都专门用来 YCbCr4:2:2, YCbCr4:2:0 和 RGB 输出。S3C6410X 内的相机接口有图像旋转器和图像效果。这些性能在文件夹型手机内非常有用。

#### 20.1.1.性能

相机接口的主要性能有：

- (1) 支持 ITU-R BT 601/656 8 位模式
- (2) 数据缩放能力
- (3) 视频同步信号的可编程极性
- (4) 支持最大的 4096x4096 像素的相机输入
- (5) 编解码/预览图像镜像和旋转（只对预览图像），有 X,Y 翻转，90°，180°，和 270° 旋转功能。
- (6) 编解码/预览输出图像产生（RGB 16/18/24 位格式和 YCbCr4:2:2/YCbCr4:2:0 格式）
- (7) 支持相机图像捕捉帧控制功能
- (8) 支持扫描线消除功能
- (9) 支持 YCbCr4:2:2 图像格式
- (10) 支持 LCD 控制器直接路径

(11) 支持交错相机输入

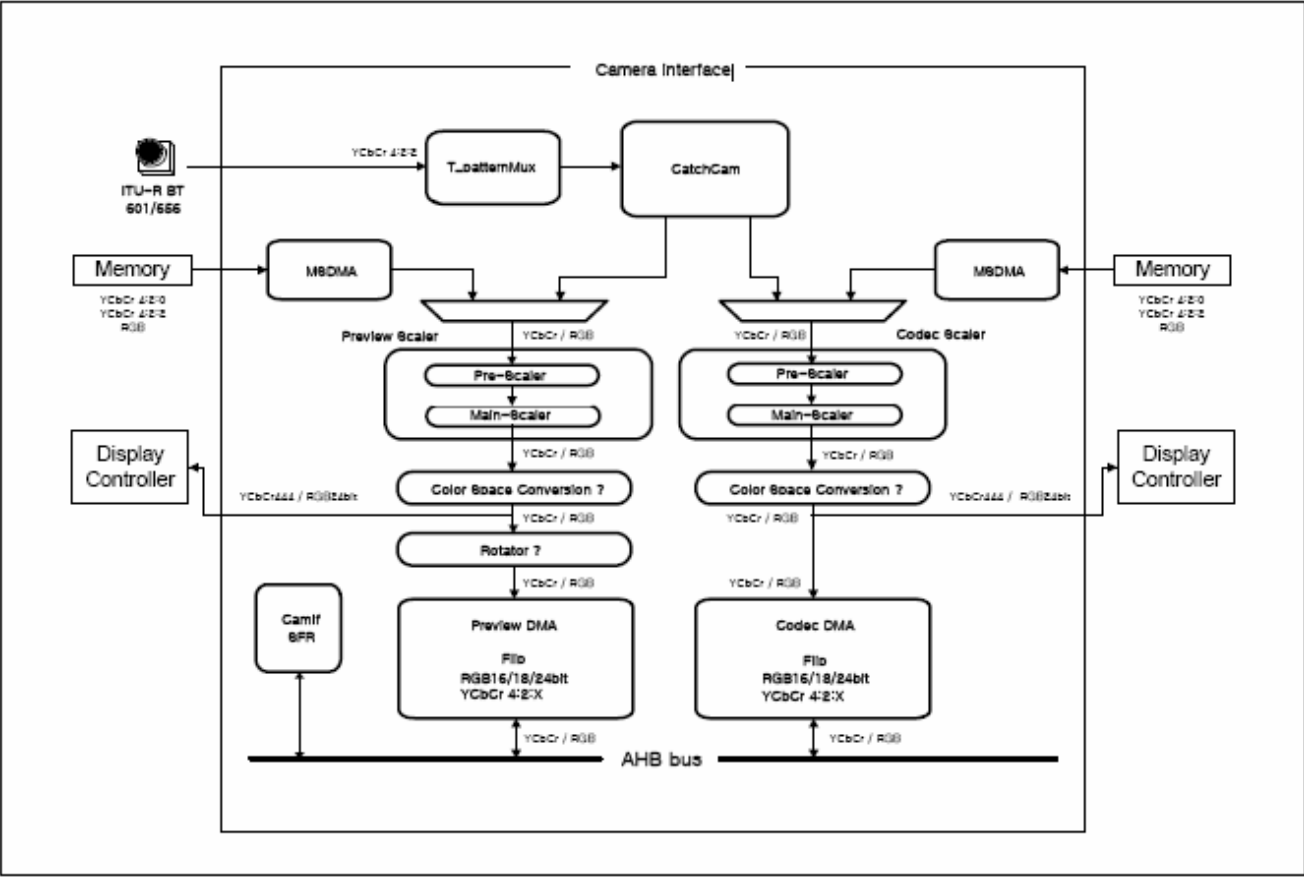


图 20-1 相机接口概述

表 20-1 预览和编解码混合水平尺寸

	预览	编解码
Prescaled input max hsize	720 像素	2048 像素
Scaler bypass	4096	4096 像素
TargetHsize(不旋转)	4096 像素 (bypass YCbCr) 720 像素 (Except Bypass)	4096 像素 (bypass YCbCr) 2048 像素 (Except Bypass)
TargetHsize(旋转)	720 像素 (RGB) 360 像素 (YCbCr)	

20.1.2.外部接口

相机几口支持下面所述视频标准。

- (1) ITU-R 601 YCbCr 8 位模式
- (2) ITU-R BT 656 YCbCr 8 位模式

20.1.3. 信号描述

表 20-2 相机接口信号描述

名称	I/O	描述
外部相机处理器接口信号		
XciPCLK	I	像素时钟，由相机处理器 A 驱动
XciVSYNC	I	帧同步，由相机处理器 A 驱动
XciHREF	I	水平同步，由相机处理器 A 驱动
XciYDATA[7:0]	I	像素数据由相机处理器 A 驱动
XciTSTn	O	相机处理器 A 的软件按复位或电源下载
XciCLK	O	外部 ISP 时钟

20.1.4.时序图

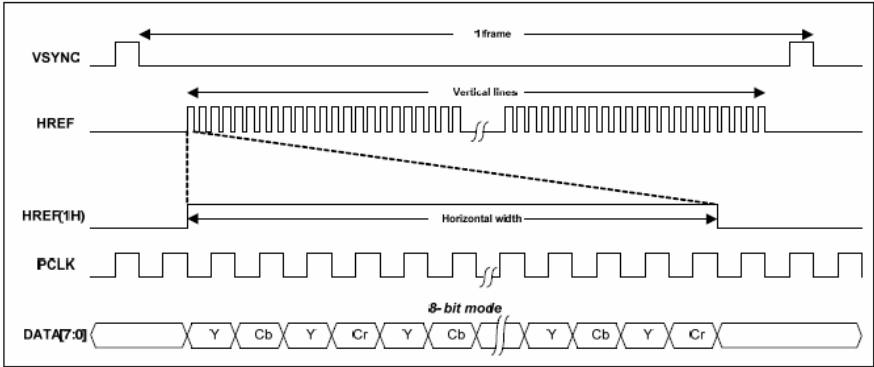


图 20-2 ITU\_R BT 601 输入时序图

ITU-R BT 656 格式内有两个时序参考信号，一个信号在每个视频数据模块的开始，另一个信号在每个视频数据模块的末尾，具体内容如 20-3，和表 20-3.

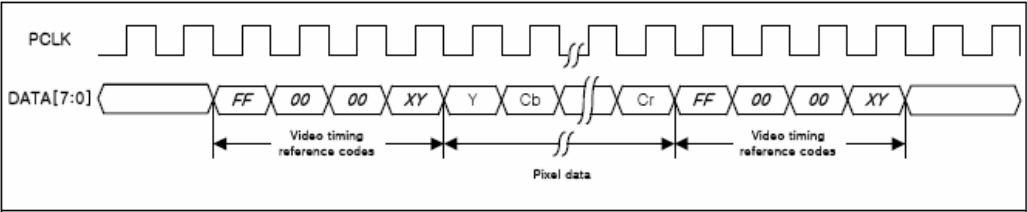


图 20-3 ITU-R BT 656 输入时序图

表 20-3 ITU-656 8 位格式的视频时序参考代码

数据位序号	第一个字	第二个字	第三个字	第四个字
7 (MSB)	1	0	0	1
6	1	0	0	F
5	1	0	0	V
4	1	0	0	H
3	1	0	0	P3
2	1	0	0	P2
1	1	0	0	P1
0	1	0	0	P0

注释：F=0（field 1 期间），1（field 2 期间）

V=0（任何地方），1（blanking field 期间）

H=0（在 SAV:活动视频开端），1（在 EAV:活动视频末端）

P0, P1, P2, P3=保护位

相机接口字保留如“FF-00-00”数据后可以做视频同步位，如 H(SAV,EAV)和 V(帧同步)。

警告：所有的外部相机接口 IO 不许和任何 GPIO 或双向端口连接。为了降低噪音，建议所有的外部相机接口 IO 采用 shimitt-触发类型。



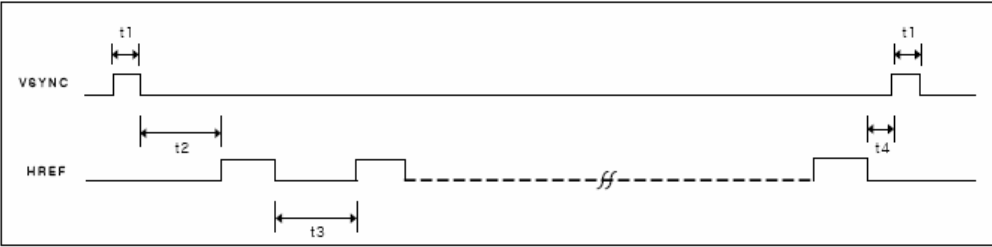


图 20-4 同步信号时序图

表 20-4 同步信号时序要求

	最小	最大
t 1	像素时钟为 12 周期	-
t 2	像素时钟为 12 周期	-
t 3	像素时钟为 2 周期	-
t 4	像素时钟为 12 周期	-

注释：如果旋转器使能，(t 4+1) 必须足够长，可以完成 DMA 处理。这主要是因为旋转器线缓冲区的 DMA 处理延迟 4 或 8 水平线。

## 20.2 相机接口操作

### 20.2.1. 四个 DMA 端口

相机接口有四个 DMA 端口。在 AHB 总线上将进行预览的 MSDMA 输入，进行编解码的 MS | DMA 输入，P-端口和 C 端口都已经分开且各自保持独立。MADMA 读取 YCbCr4:2:2, YCbCr4:2:0 或 RGB 图像。P-端口和 C-端口存储 MSDMA 的输入数据，或相机的输入数据。四个主端口支持各种各样的应用，如 DSC（数码相机）,MPEG-4 视频会议，视频录像等等。P-端口图像可以用来作为预览图像，C-端口图像可以用来在 DSC 应用内作为 JPEG 图像。寄存器可以分别设置四个 DMA 端口的使能。

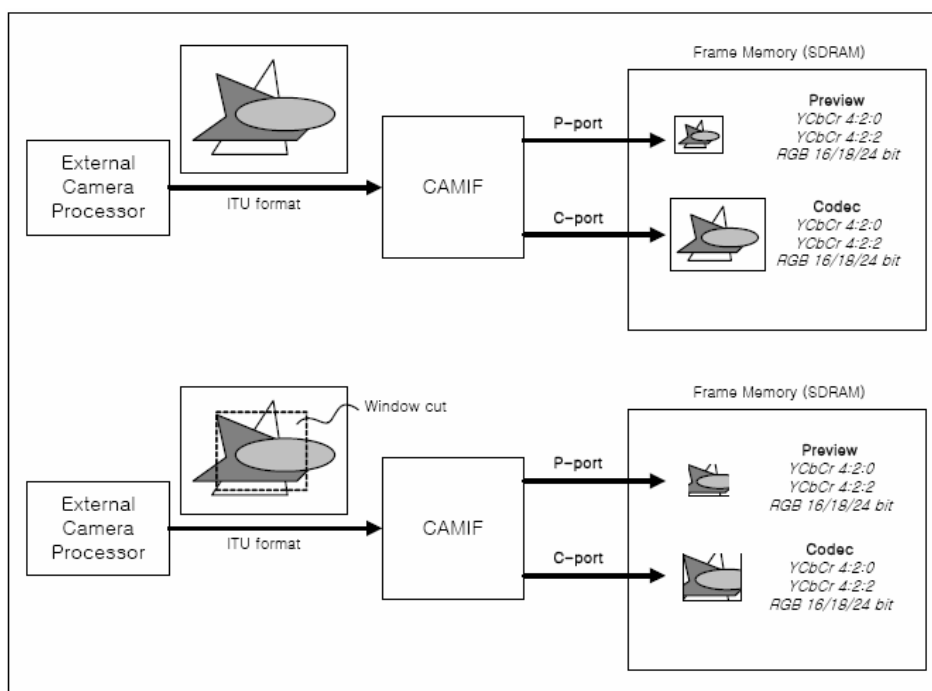


图 20-5 通过相机处理器数据的 DMA 端口

## 20.2.2. 时钟范围

相机接口有两个时钟范围。一个是系统总线时钟 **HCLK**，另一个是像素时钟 **PCLK**。系统时钟必须比像素时钟速度快。如图 20-6 中高亮显示部分，**CAMCLK** 必须从固定频率时钟划分出来，如 **APLL** 或 **MPLL** 时钟。如果使用外部时钟振荡器，**CAMCLK** 必须是浮动的。内部 **Scaler** 时钟是系统时钟。两种时钟范围不要求同步使用。其他信号如 **PCLK** 必须铜 shimitt-触发电平开关连接。

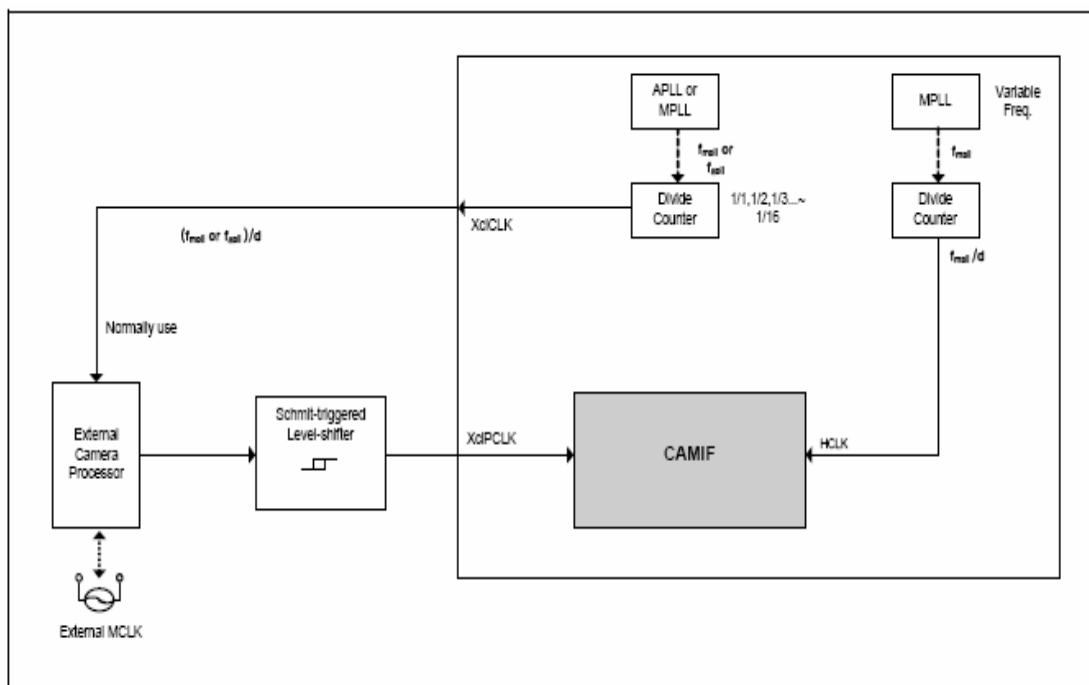


图 20-6 相机接口时钟产生

### 20.2.3.FRAME 内存等级

Frame 存储器的每个 P-端口和 C-端口都有四个 ping-pong 存储器。Ping-pong 存储器有三个元素存储器，分别是亮度 Y，色度 Cb 和色度 Cr。相机接口的仲裁优先级必须比其他任何 matters 高，LCD 控制器除外。建议相机接口优先级最好为固定优先级，而不是旋转优先级。在 Multi-AHB 总线情况下，系统总线优先级包括相机接口优先级必须比其它的优先级高。如果 AHB-总线足够通畅，DMA 操作在一个横向空白期间不会结束，可能会进入错误功能状态。因此，相机接口的优先级必须与其他循环或通知仲裁优先级区别开，并保持独立。在存储器矩阵系统内，包含相机接口的 AHB 总线必须比其它 Multi-AHB 总线的优先级高。相机接口不能为默认的 AMBA AHB 系统的主机。

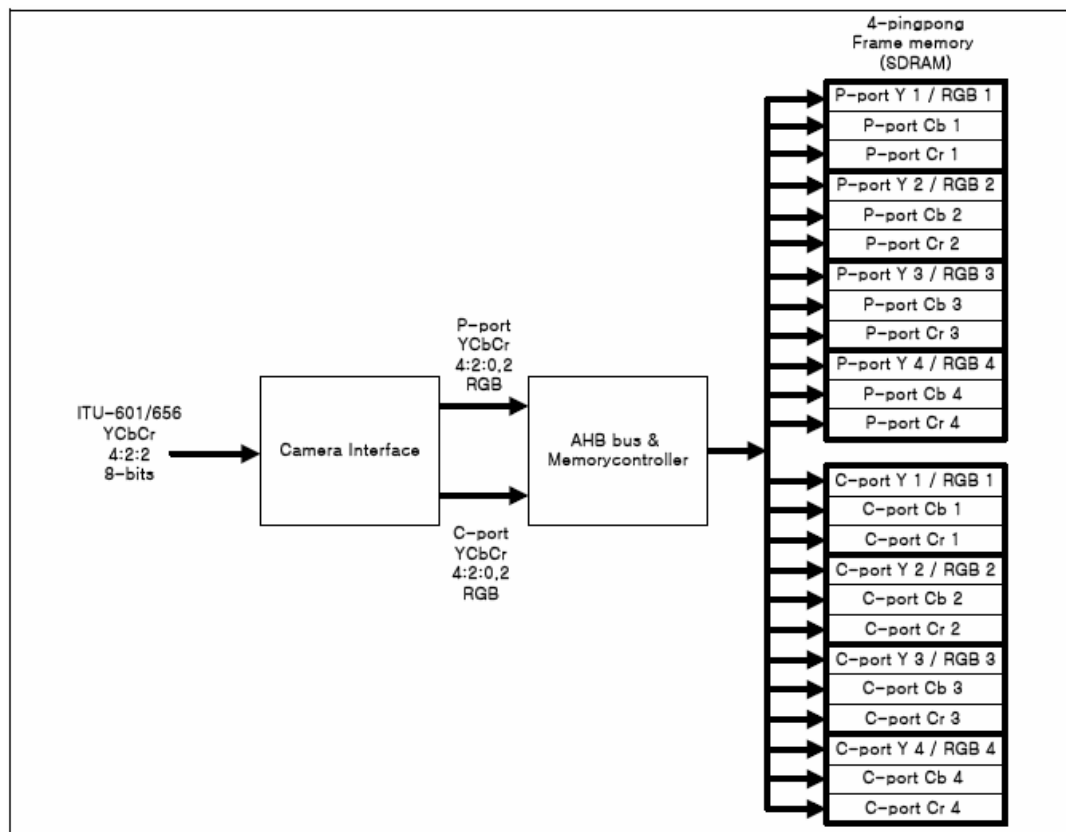


图 20-7 Ping-pong 存储器等级图

## 20.2.4.存储器存储方式

Little-endian 是向帧存储器内储存的方法。第一个进入的像素存储在 LSB 端，最后进入的像素存储在 MSB 端。由 AHB 总线运载的数据是 32 位字。因此相机接口的 Y-Cb-CrZ 字采用 little endian 方式。对于 RGB 24 位/18 位格式，一个像素一个字，对于 16 位格式或 YCbCr4:2:2 格式，两个像素一个字。图 20-8 是存储器的储存形式。

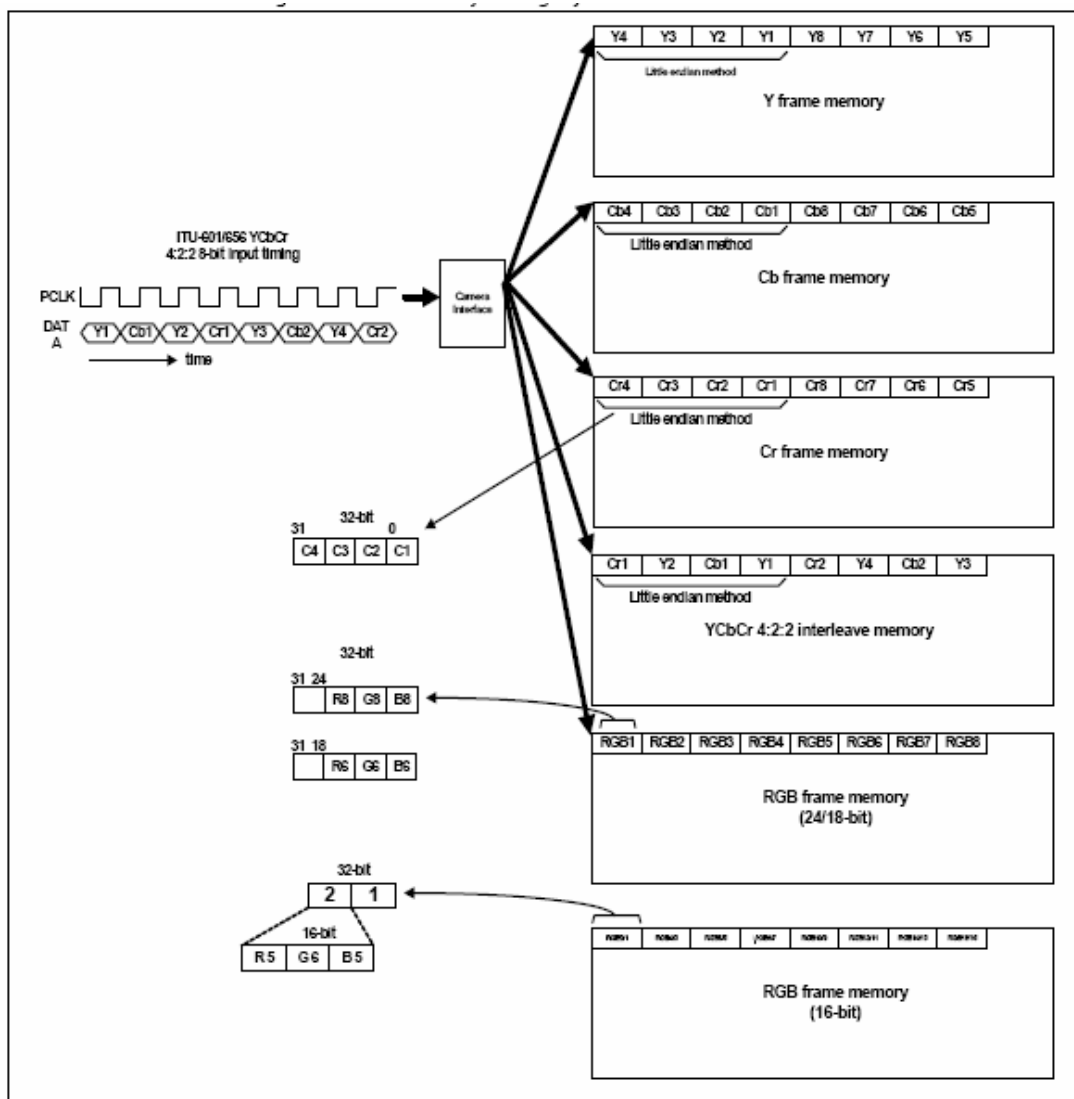


图 20-8 存储器储存方式

### 20.2.5.寄存器设置的时序图

第一个寄存器设置为帧捕捉命令，可以在帧期间的任何部分发生。建议将第一个设置处于 VSYNC“L”状态。VSYNC 信息可以从状态 SFR 读取。具体内容可以参见图 20-9.所有的命令包含 ImgCptEn，并且在 VSYNC 下降沿有效。确定除了第一个 SFR 设置外，所有的命令在 ISR 下都可以被程式化。图像镜像或旋转，加窗和缩放设置可以在捕捉操作时改变。如果一些路径选择 MSDMA 输入模式，在 MSDMA 和 P 端

口或 C 端口的 DMA 操作结束后，所有的命令可程式化。

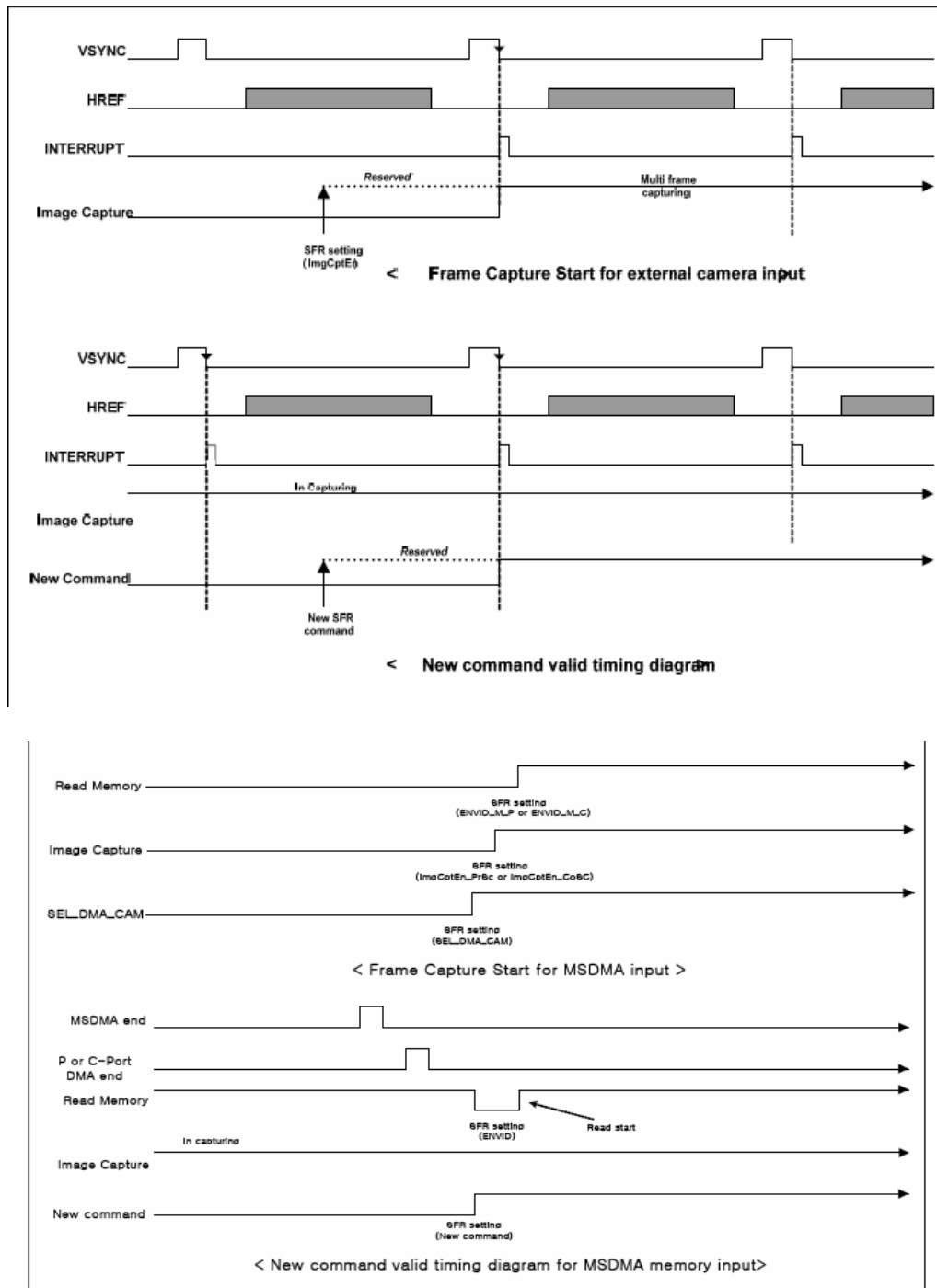


图 20-9 寄存器设置时序图

### 20.2.6.Last IRQ 时序图

IRQ 除了 LastIRQ 外都在图像捕捉前产生。Last IRQ 的意思是相机信号捕捉末尾可以通过下面的时序图进行设置。LastIRQEn 是自动清除的。ISR 内的 SFR 设置是为了下一帧命令。因此，为了有足够的 Last IRQ，必须跟随 LastIRQEn 和 ImgCptEn/ImgCptEn\_CoSc/IngCptEnPrSC 之间的下一串图像。建议在 ISR 内，在相同的时间内设置 ISRImgCptEn/ImgCptEn\_CoSc/IngCptEnPrSC，并且在 SFR 设置的最后进行。FrameCnt 是 ISR 内的读取操作，意思是下一帧计数。在下面所示图内，最后捕捉的真计数为‘1’。这就意味着帧 1 是帧 0-3 之间最后捕捉的帧。FrameCnt 以 IRQ 上升形式增加 1。

(1) 相机输入捕捉路径

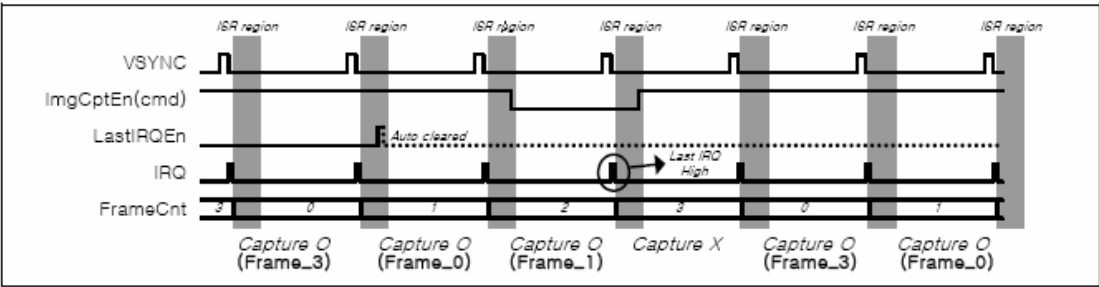


图 20-10 Last IRQ 时序图（LastIRQEn is enabled）

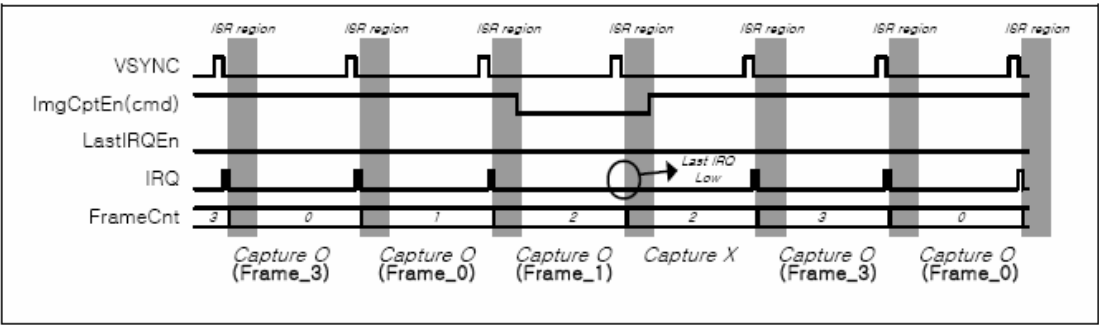


图 20-11 Last IRQ 时序图（LastIRQEn is disabled）

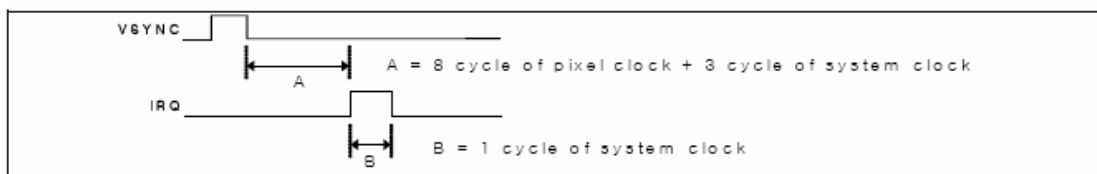


图 20-12 VSYNC&IRQ 信号时序条件

### 20.2.7. IRQ 时序图

MSDMA 输入可以通过 SFR 设置选择。这种情况下，在每帧的 P 端口和 C 端口 DMA 操作完成后将产生 IRQ。这个模式通过用户 SFR 设置认识起点。因此，这个模式不需要 IRQ 起点和 LastIRQ。 FrameCnt 在 ENVID\_M\_P 由低到高和 ImgCptEn\_PrSC= ‘1’ 时增加 1。

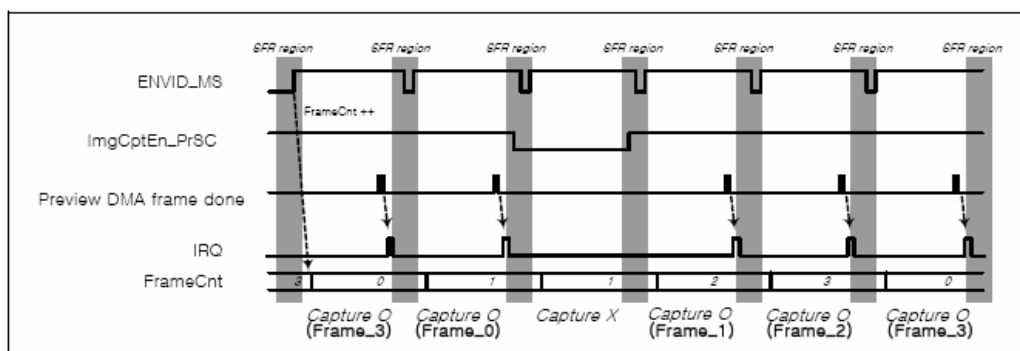


图 20-13 IRQ 时序图

### 20.2.8.MSDMA 性能

MSDMA 支持存储器数据缩放。特别是 PIP 操作要求两个不同的图像数据。第一个图像由一些编解码器保存。第二个图像通过 MSDMA 路径保存。MSDMA 路径有通过缩放/DMA 路径的 YCbCr/RGB 输出格式。LCD 控制器显示和控制两个图像。如果在预览路径或编解码路径内要求 MSDMA, SFR SEL\_DMA\_CAM\_P 信号必须设置为 ‘1’。这个输入路径称为存储缩放 DMA 路径。此路径不允许加窗缩放功能。

注释：MSDMA 输入的纯粋器图像格式有：

YCbCr4:2:0(无交错)



YCbCr4:2:2(无交错)

YCbCr4:2:2(交错)

RGB

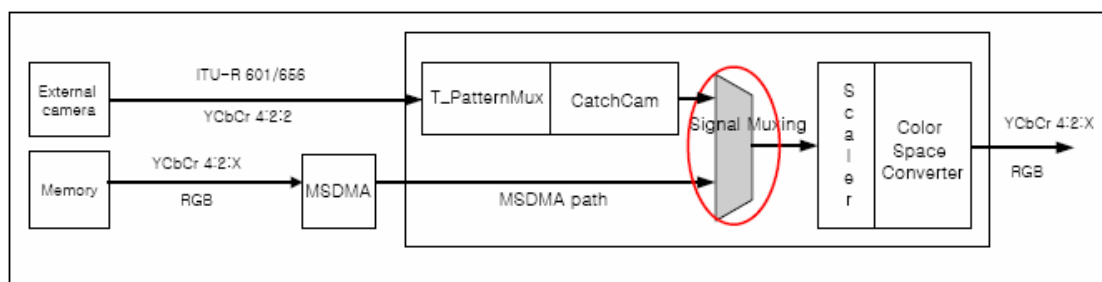


图 20-14 MSDMA 或外部相机接口

## 20.2.9. 相机接口输入支持

为了从外部相机内取得数据，6410 支持 ITU-R BT 601 YCbCr 8/16 位格式和 ITU-R BT 656 YCbCr 8 位格式。6410 不仅支持逐步输入，同时也支持两种模式下的逐步输入。

### 1. 逐步输入

在逐步输入模式下，所有的输入数据将通过帧单元连续的储存在四个缓冲区内。

### 2. 交错模式

在交错模式下，输入数据被储存在四个缓冲区内。这种模式下，偶数领域帧数据和奇数领域帧数据轮流储存。因此偶数领域帧数据储存在第一和第三 pingpong 存储器内而奇数领域帧数据储存在第二和第四 pingpong 存储器内。图像捕捉的时候，开始的帧通常是偶数领域帧。

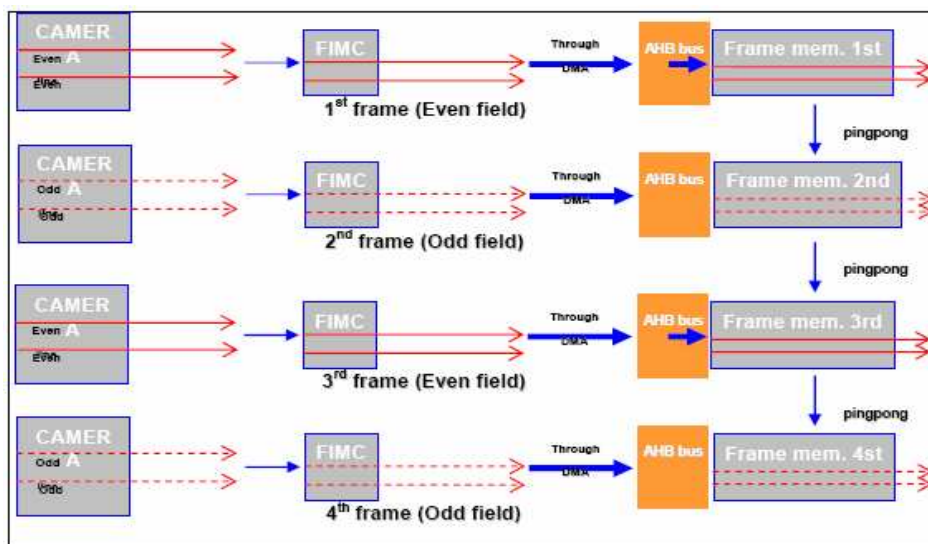


图 20-15 帧缓冲区控制

### 3. 601 接口

为了确定帧是偶数领域或奇数领域，需要用到 FIELD 信号。入股 FIELD 信号位高电平，向奇数帧输入数据，否则向偶数帧输入数据。FIELD 值的意义可以反转。如果设置 InvPolFIELD 为 1，FIELD 信号的高电平意味着当前帧为偶数帧。注意当使用 601 接口模式时，必须设置 FIELDMODE 为 1。

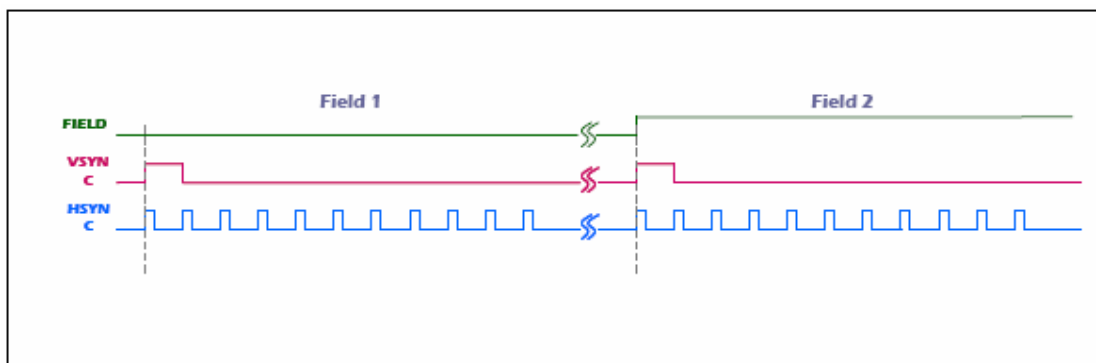


图 20-16 帧缓冲区控制

### 4. 656 接口

在 656 接口内，的当前帧的领域信息是 EAV 和 SAV 内的第四个字。

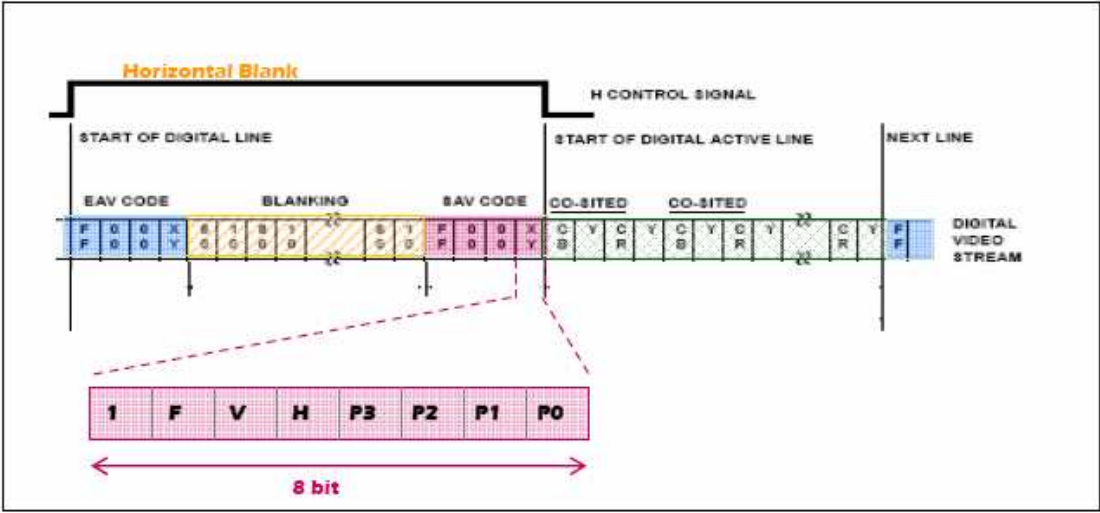


图 20-17 656 区域捕捉控制

表 20-5 视频时序参考代码

数据位序号	第一个字(FF)	第二个字(00)	第三个字(00)	第四个字(XY)
9(MSB)	1	0	0	1
8	1	0	0	F
7	1	0	0	V
6	1	0	0	H
5	1	0	0	P3
4	1	0	0	P2
3	1	0	0	P1
2	1	0	0	P0
1	1	0	0	0
0	1	0	0	0

F=0 FIELD1 期间

F=1 FILED2 期间

V=0 任何时候

V=1 在 field blanking 期间

H=0 SAV 内

H=1 EAV 内

P3, P2, P1, P0:保护位
--------------------

20.3 软件接口（S3C6410 SFR 内的相机接口）

20.3.1. 相机接口特殊功能寄存器

最后的 ‘L’ 列的意思是 SFR 在相机捕捉期间内的 VSYNC 边沿可以改变。（O:可能改变，X：不可能改变）。  
‘M’ 列的意思是在使用 MSDMA 路径时 SFRs 有相关联的捕捉结果。（O： 有关系，X：没关系）

20.3.2. 相机源格式寄存器

寄存器	地址	读/写	描述	复位值
CISRCFMT	0x78000000	读/写	相机输入源格式	0

CISRCFMT	位	描述	初始状态	M	L
ITU601_656n	[31]	1:ITU-R BT. 601 YCbCr 8 位模式使能 0. ITU-R BT. 656 YCbCr 8 位模式使能	0	X	X
UVOffest	[30]	Cb, Cr 值补偿区控制  0:+128  1:+0（常用）	0	X	X
Reserved	[29]		0	X	X
SrcHsize_CAM	[28:16]	相机源横向像素序号（必须为 8 的倍数。  最小值为 8。如果 WinOfsEn 为 0, PreHorRatio 必须为 4 的倍数。	0	X	0
Order411_CAM	[15:14]	8 位模式相机输入 YCbCr 顺序通知 <div>8 位模式</div>	0	X	X

		00: YCbYCr 01: YcrYCb 10: CbYCrY 11: CrYCbY			
Reserved	[13]		0	X	X
SrcHsize_CAM	[12:0]	相机源纵向像素序号（必须为 8 的倍数。 最小值为 8。如果 WinOfsEn 为 0, PreHorRatio 必须为 4 的倍数。	0	X	0

20.3.3.窗口补偿区寄存器

寄存器	地址	读/写	描述	复位值
CIWDOFST	0x78000004	读/写	窗口补偿区寄存器	0

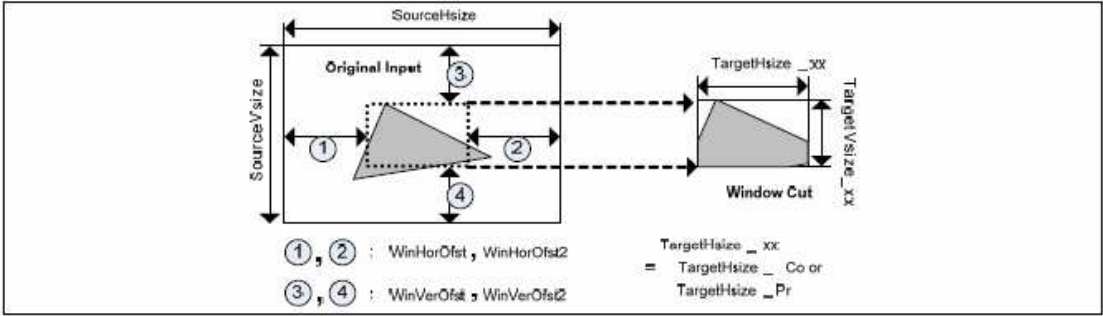


图 20-18 窗口补偿区清单  
 (在 CIWDOFST2 寄存器内 WinHorOfst2 和 WinVerOfst2 被分配)

CIWDOFST	位	描述	初始状态	M	L
WinPfsEn	[31]	1:窗口补偿区使能 0: 没有补偿区	0	X	0

ClrOvCoFiY	[30]	1:清除输入 CODEC FIFO Y 的溢出指示标志 0: 常规	0	X	X
Reserved	[29]		0	X	X
ClrOvRLB_Pr	[28]	清除预览路径内的旋转行缓冲区的溢出指示标志	0	X	X
ClrOvPrFiY	[27]	1:清除输入 PREVIEW FIFO Y 的溢出指示标志 0: 常规	0	X	X
WinHorOfst	[26:16]	窗口横向补偿区像素单元（必须为 2 的倍数）	0	X	0
ClrOvCoFiCb	[15]	1:清除输入 CODEC FIFO Cb 的溢出指示标志 0: 常规	0	X	X
ClrOvCoFiCr	[14]	1:清除输入 CODEC FIFO Cr 的溢出指示标志 0: 常规	0	X	X
ClrOvPrFiCb	[13]	1:清除输入 PREVIEW FIFO Cb 的溢出指示标志 0: 常规	0	X	X
ClrOvPrFiCr	[12]	1:清除输入 PREVIEW FIFO Cr 的溢出指示标志 0: 常规	0	X	X
Reserved	[11]		0	X	X
WinVerOfst	[10:0]	窗口纵向补偿区像素单元（必须为 2 的倍数）	0	X	0

注释：当清除标志以后需要将清除位设置为 0.

Crop Hsize (=SourceHsize-WinHorOfst-WinHorOfst2) 必须为 8 的倍数，PreHorRatio 必须为 4 的倍数

Crop Vsize (=SourceVsize-WinVorOfst-WinVorOfst2) 当缩小时必须为 PreHorRatio 的倍数。如果编解码输入格式 YCbCr4:2:0 时，必须为偶数，且最小值为 8.

例如

Crop Hsize	允许的 Prescal_ratio	PreDstWidth_xx
8n	2	4n
16n	2 或 4	4n
32n	2, 4 或 8	4n

20.3.4. 全局控制寄存器

寄存器	地址	读/写	描述	复位值
CLGCTRL	0x78000008	读/写	全局控制寄存器	2000_0000

CLGCTRL	位	描述	初始状态	M	L
SwRst	[31]	相机接口软件复位。在设置此位以前需要在第一次 SFR 设置时将 CISRCFMT 的 ITU601_656n 位设置为 ‘1’。	0	X	X
CamRst	[30]	外部相机处理器复位或电源运行控制	0	X	X
Reserved	[29]		1	X	X
TestPattern	[28:27]	这个寄存器必须设置在 ITU-T 601 8 位模式下。不允许 16 位输入模式或 ITU-T 656 模式。（最大值是 1280x1024） 00: 外部相机处理器输入（常规） 01: 彩色条测试模式 10: 横向增量测试模式 11: 垂直增量测试模式	0	X	X
InvPolPCLK	[26]	1:PCLK 极性反相    0: 常规	0	X	X
InvPolVSYNC	[25]	1:VSYNC 极性反相    0: 常规	0	X	X
InvPolHREF	[24]	1:HREF 极性反相    0: 常规	0	X	X
Reserved	[23]		0	X	X
IRQ_Ovfen	[22]	1:溢出中断使能（发生溢出时产生中	0	X	X

		断) 0: 溢出中断 disable (常规)			
Href_mask	[21]	1:Vsync 高电平时平布 SHUCHU Href	0	X	X
IRQ_LEVEL	[20]	1:电平中断 0:边沿触发中断 (默认)	0	X	X
IRQ_CLR_c	[19]	此位只与电平中断有关。当向 IRQ_CLR_c 写入 '1' 时编解码路径中断清除。此位自动清除	0	X	X
IRQ_CLR_p	[18]	此位只与电平中断有关。当向 IRQ_CLR_p 写入 '1' 时编解码路径中断清除。此位自动清除	0	X	X
Reserved	[17:3]				
FIELDMODE	[2]	ITU601 交错区域模式 (在 ITU656 模式内不用考虑此位) 1: 采用 FIELD 端口模式 0: 保留	0	X	X
InvPloFIELD	[1]	1: FIELD 极性反相 0: 常规	0	X	X
Cam_Interface	[0]	外部相机浏览方式 1: 反相 0: 逐步	0	X	X



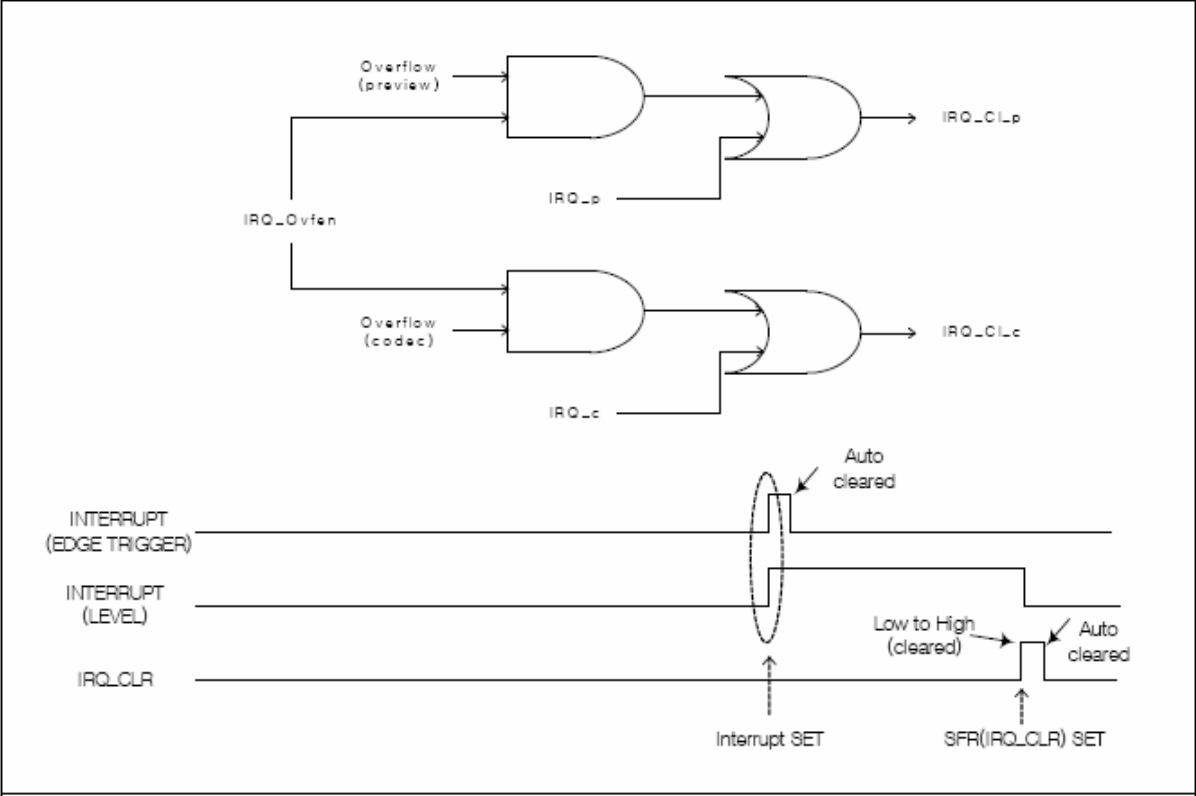


图 20-19 中断产生清单

### 20.3.5. 窗口补偿区寄存器 2

寄存器	地址	读/写	描述	复位值
CIWDOFST2	0x78000014	读/写	窗口补偿区寄存器 2	0

CIWDOFST2	位	描述	初始状态	M	L
Reserved	[31:27]		0	X	X
WinHorOfst2	[26: 16]	窗口横向补偿区 2 像素单元（必须为 2 的倍数）	0	X	0
Reserved	[15: 11]		0	X	X
WinVerOfst	[10:0]	窗口纵向补偿区 2 像素单元（必须为 2 的倍数）	0	X	0

### 20.3.6.编解码器输出 Y1 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CICOYSA1	0x78000018	读/写	编解码器 DMA 第一帧开始地址	0

CICOYSA1	位	描述	初始状态	M	L
CICOYSA1	[31:0]	无交错 Y, 交错 YCbCr, RGB: 第一帧开始地址	0	0	X

### 20.3.7.编解码器输出 Y2 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CICOYSA2	0x7800001C	读/写	编解码器 DMA 第二帧开始地址	0

CICOYSA2	位	描述	初始状态	M	L
CICOYSA2	[31:0]	无交错 Y, 交错 YCbCr, RGB: 第二帧开始地址	0	0	X

### 20.3.8.编解码器输出 Y3 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CICOYSA3	0x78000020	读/写	编解码器 DMA 第三帧开始地址	0

CICOYSA3	位	描述	初始状态	M	L
CICOYSA3	[31:0]	无交错 Y, 交错 YCbCr, RGB: 第三帧开始地址	0	0	X

### 20.3.9.编解码器输出 Y4 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CICOYSA4	0x78000024	读/写	编解码器 DMA 第四帧开始地址	0

CICOYSA4	位	描述	初始状态	M	L
CICOYSA4	[31:0]	无交错 Y, 交错 YCbCr, RGB: 第四帧开始地址	0	0	X

### 20.3.10.编解码器输出 CB 1 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CICOCBSA1	0x78000028	读/写	编解码器 DMA 的 Cb 第一帧开始地址	0

CICOCBSA1	位	描述	初始状态	M	L
CICOCBSA1	[31:0]	编解码器 DMA 的 Cb 第一帧开始地址	0	0	X

### 20.3.11.编解码器输出 CB2 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CICOCBSA2	0x7800002C	读/写	编解码器 DMA 的 Cb 第二帧开始地址	0

CICOCBSA2	位	描述	初始状态	M	L
CICOCBSA2	[31:0]	编解码器 DMA 的 Cb 第二帧开始地址	0	0	X

### 20.3.12 编解码器输出 CB 3 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CICOCBSA3	0x78000030	读/写	编解码器 DMA 的 Cb 第三帧开始地址	0

CICOCBSA3	位	描述	初始状态	M	L
CICOCBSA3	[31:0]	编解码器 DMA 的 Cb 第三帧开始地址	0	0	X

### 20.3.13.编解码器输出 CB4 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CICOCBASA4	0x78000034	读/写	编解码器 DMA 的 Cb 第四帧开始地址	0

CICOCBASA4	位	描述	初始状态	M	L
CICOCBASA4	[31:0]	编解码器 DMA 的 Cb 第四帧开始地址	0	0	X

### 20.3.14.编解码器输出 CR1 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CICOCRSA1	0x78000038	读/写	编解码器 DMA 的 CR 第一帧开始地址	0

CICOCRSA1	位	描述	初始状态	M	L
CICOCRSA1	[31:0]	编解码器 DMA 的 CR 第一帧开始地址	0	0	X

### 20.3.15.编解码器输出 CR2 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CICOCRASA2	0x7800003C	读/写	编解码器 DMA 的 CR 第二帧开始地址	0

CICOCRASA2	位	描述	初始状态	M	L
CICOCRASA2	[31:0]	编解码器 DMA 的 CR 第二帧开始地址	0	0	X

### 20.3.16 编解码器输出 CR 3 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CICOCRASA3	0x78000040	读/写	编解码器 DMA 的 CR 第三帧开始地址	0

CICOCRASA3	位	描述	初始状态	M	L
CICOCRASA3	[31:0]	编解码器 DMA 的 CR 第三帧开始地址	0	0	X

20.3.17.编解码器输出 CR4 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CICOCRASA4	0x78000044	读/写	编解码器 DMA 的 CR 第四帧开始地址	0

CICOCRASA4	位	描述	初始状态	M	L
CICOCRASA4	[31:0]	编解码器 DMA 的 CR 第四帧开始地址	0	0	X

20.3.18.编解码器目标格式寄存器

寄存器	地址	读/写	描述	复位值
CICOTRGFMT	0x78000048	读/写	编解码器 DMA 的的目标图像格式	0

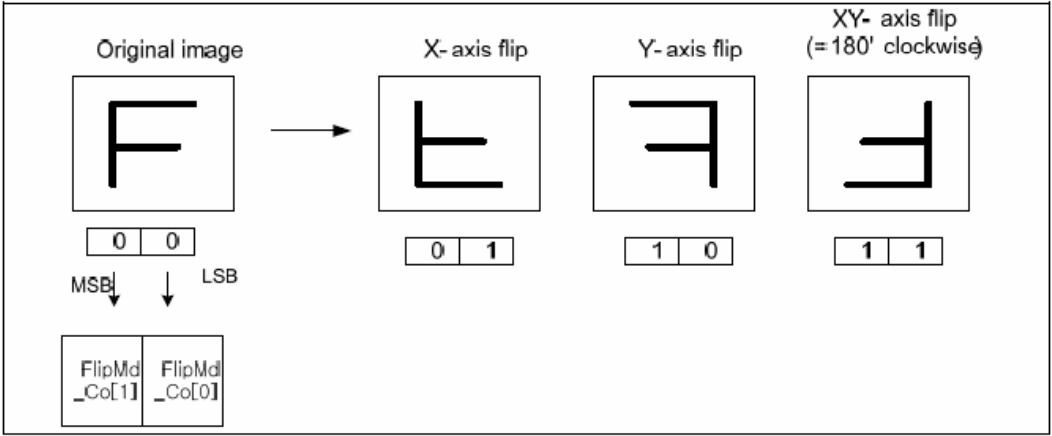


图 20-20 编解码器图像镜像

CICOTRGFMT	位	描述	初始状态	M	L
Reserved	[31]		0	X	X
OutFormat_Co	[30:29]	00:YCbCr4:2:0 编解码器输出图像格式（无交错） 01:YCbCr4:2:2 编解码器输出图像格式（无交错） 10:YCbCr4:2:2 编解码器输出图像格式（交错） 11:RGB 编解码器输出图像格式	0	0	0
TargetHsize_Co	[28:16]	编解码器 DMA 目标图像的横向像素数（16 的倍数，最小值为 16）	0	0	0
FlipMd_Co	[15:14]	编解码器 DMA 图像镜像和旋转 00: 常规 01: X 轴镜像 10: Y 轴镜像 11: 180° 旋转	0	0	0
Reserved	[13]				
TargetVsize_Co	[12:0]	编解码器 DMA 目标图像的纵向像素数，最小值为 4.	0	0	0

TargetHsize\_Co 和 TargetVsize\_Co 不能大于相机 SourceHsize 和 SourceVsize。DMA 输入源尺寸可以忽略。

20.3.19.编解码器 DMA 控制寄存器

寄存器	地址	读/写	描述	复位值
CICOCTRL	0x7800004C	读/写	编解码器 DMA 控制相关寄存器	0

CICOCTRL	位	描述	初始状态	M	L
Reserved	[31:24]	编解码器 DMA 的 CR 第四帧开始地址	0	X	X
Yburst1_Co	[23:19]	编解码器 Y 帧的主突发长度	0	0	0
Yburst2_Co	[18:14]	编解码器 Y 帧的 Remained burst 长度	0	0	0
Cburst1_Co	[13:9]	编解码器 Cb/Cr 帧的主突发长度	0	0	0
Cburst2_Co	[8:4]	编解码器 Cb/Cr 帧的 Remained burst 长度	0	0	0
Reserved	[3]		0	X	X
LastIRQEn_Co	[2]	1:在帧捕捉末尾 Last IRQ enabled 0: 常规	0	X	X
Order422_Co	[1:0]	使 YCbCr4:2:2 输出规则寄存器储存形式交错	0	0	0
		00			
		01			
		10			
		11			

交错突发长度（交错 YCbCr4:2:2）

Y 突发长度	2, 4, 8
C 突发长度（C 突发长度=Y 突发长度/2）	1, 2, 4
希望的突发长度（=Y+2C）	4, 8, 16

注释：当编解码器输出格式是 YCbCr4:2:2 交错格式时，ScalerBypass\_Co=0，ScaleUp\_V\_Co=1，不允许希望的主突发长度=16，希望的保持突发长度≠16

20.3.20.编解码定标器和预览定标器的寄存器设置指南

SRC\_Width 和 DST\_Width 符合字边界规定参数，横向像素的数值可以用 kn 表示，其中 n=1,2,3,.....，k=1/2/8 for 24bppRGB/16bppRGB /YCbCr420 图像。TargetHsize 不能比相机的 SourceHsize 大，同理

TargetVsize 不能比相机的 SourceVsize 大.DMA 输入源尺寸可以忽略。

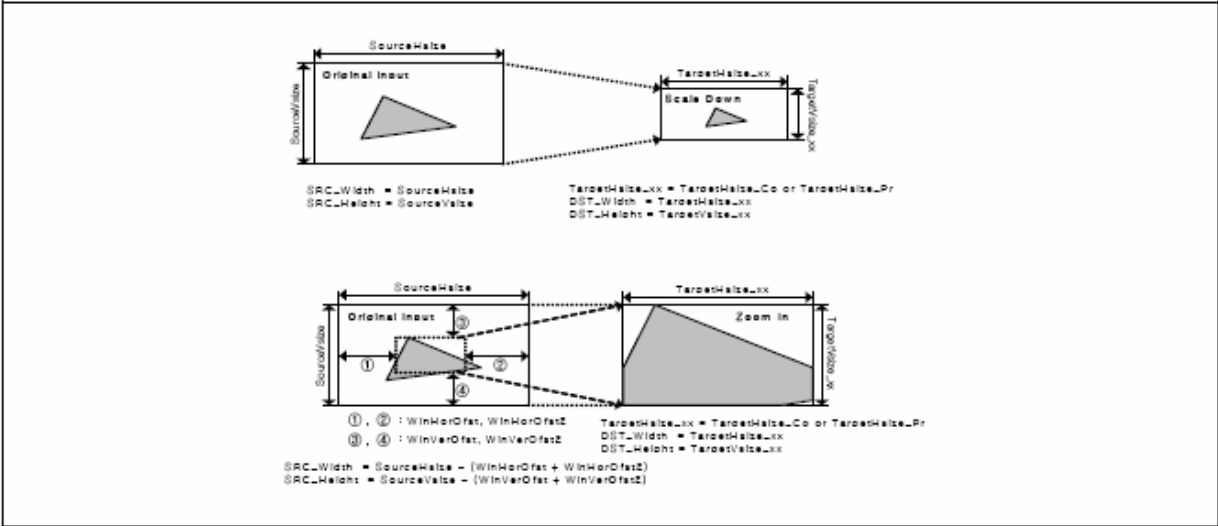


图 20-21 缩放清单

### 20.3.21.编解码器预览缩放控制寄存器 1

寄存器	地址	读/写	描述	复位值
CICOSPRERATIO	0x78000050	读/写	编解码器预先缩放比例控制	0

CICOSPRERATIO	位	描述	初始状态	M	L
SHfactor_Co	[31:28]	编解码器预览缩放的转移因子	0	0	0
Reserved	[27:23]		0	X	X
PreHorRatio_Co	[22:16]	编解码器预览缩放的横向比例	0	0	0
Reserved	[15:7]		0	X	X
PreVorRatio_Co	[6:0]	编解码器预览缩放的纵向比例	0	0	0

### 20.3.22.编解码器预览缩放控制寄存器 2

寄存器	地址	读/写	描述	复位值
CICOSCPREDST	0x78000054	读/写	编解码器预先缩放目标格式	0



CICOSCPRERATIO	位	描述	初始状态	M	L
Reserved	[31:28]		0	X	X
PreDstWidth_Co	[27:16]	编解码器预览缩放的目标宽度	0	X	0
Reserved	[15:12]		0	X	X
PreDstHeight_Co	[11:0]	编解码器预览缩放的目标高度	0	X	0

### 20.3.23. 编解码器主缩放控制寄存器

寄存器	地址	读/写	描述	复位值
CICOSCTRL	0x78000058	读/写	编解码器主缩放控制	0x18000000

CICOSCTRL	位	描述	初始状态	M	L
ScalerBypass_Co	[31]	<p>编解码器缩放旁路。这种情况下，ImgCptEn_CoSC 必须为 0，ImgCptEn 必须为 1。</p> <p>通常这种模式使用大型图片进行最大尺寸缩放。因此，不建议捕捉预览图像。这种模式适合用在 DSC 应用上，进行捕捉 JPEG 输入图像。</p> <p>这种情况下，输入像素缓冲区仅有输入 FIFO 决定，因此系统总线不会很忙。</p> <p>ScalerBypass 有一些约束，不可以进行尺寸缩放，不可以进行颜色空间转换，旋转，和 MSDMA 内存输入图像。只允许 YCbCr 非交错 4:2:0 和交错的 4:2:2 的输入输出格式。</p>	0	0	0
ScaleUp_H_Co	[30]	<p>编解码器缩放的横向缩放大小标志。</p> <p>1：大，0：小</p>	0	0	0

ScaleUp_V_Co	[29]	编解码器缩放的纵向缩放大小标志。 1: 大, 0: 小	0	0	0
CSCR2Y_c	[28]	颜色空间 RGB 向 YCbCr 转换的 YCbCr 数据动态范围选择 1: 大范围=>Y/Cb/Cr(0~255):默认范围为 大范围 0: 小范围=>Y(16~235), Cb/Cr(16~240)	1	0	0
CSCY2Y_c	[27]	颜色空间 YCbCr 向 RGB 转换的 YCbCr 数据动态范围选择 1: 大范围=>Y/Cb/Cr(0~255):默认范围为 大范围 0: 小范围=>Y(16~235), Cb/Cr(16~240)	1	0	0
LCDPathEn_Co	[26]	FIFO 模式使能。 1: FIFO 模式    0:DMA 模式	0	0	0
Interlace_Co	[25]	FIFO 模式下的输出浏览方式选择寄存器。1: 交错浏览, 0: 逐步浏览。 在 DMA 模式下, 无论此位设置什么值都 进行逐步浏览。当从相机处理器输入图像 数据时不允许采用此模式。	0	0	0
MainHorRatio_Co	[24:16]	编解码器主缩放的横向缩放比例	0	0	0
CoScalerStart	[15]	编解码器缩放开始 1: 开始缩放            0: 停止缩放	0	0	0
InRGB_FMT_Co	[14:13]	向专用的编解码器路径输入 RGB 格式的 MSDMA 00:RGB565,    01:RGB666,    10:RGB888, 11: 保留	0	0	0
OutRGB_FMT_Co	[12:11]	编解码器写入 DMA 的输出 RGB 格式 00:RGB565,    01:RGB666,    10:RGB888, 11: 保留	0	0	0

Ext_RGB_Co	[10]	RGB565/666 模式向 RGB888 模式转换的编解码器路径的输入 RGB 数据扩展使能位  1: 扩展 0: 常规  1) 输入 R=5 位, 在 RGB565 模式下 10100->10100101(扩展) 10100->10100000(常规)  2) 输入 R=6 位, 在 RGB666 模式下 10100->10110010(扩展) 10100->10110000(常规)	0	0	0
One2One_Co	[9:0]	非插值数据复制 (警告: 应该设置寄存器位 1: 相同输入输出格式的缩放尺寸, 图像效果不支持 RGB 格式和 One2One 格式)	0	0	0
MainVerRatio_Co	[8:0]	编解码器主缩放的纵向缩放尺寸	0	0	0

## 1. DMA 模式操作（常规模式）

源图像格式可以是 YCbCr420, YCbCr422 和 RGB16/18/24 位格式的一种。目标图像格式是 YCbCr420, YCbCr422 和 RGB16/18/24 位格式的一种。

所有源和目标像素具都以字边界对齐方式储存在存储器系统内。意思就是支持字节或半字大小的 DMA 操作。因此，源图像和目标图像的宽度要符合字边界条件。

## 2.FIFO 模式操作

在 FIFO 模式下，可以实现两种类型的色彩空间转换，如 RGB2YCbCr 和 YCbCr2RGB 等，像在 DMA 模式下操作一样。在没有如 FIMC -to-memory 和 Memory-to-显示控制器等额外存储器带宽的情况下，可以将目标图像传输到显示控制器内 FIFO。输出数据格式只由输出格式寄存器决定，OutFormat\_xx=RGB 格式（24 位 RGB）huo, OutFormat\_xx=YCbCr 格式（YCbCr444）。下面的表格将描绘出源图像格式和目标图像格式的约束条件。

输入图像格式（逐步输入）		输出图像格式（逐步输出或交错输出）
YCbCr	420 YCbCr 格式	YCbCr 444 或 RGB24 位

	422 YCbCr（非交错）	
	422 YCbCr（交错）	
RGB	RGB 16/18/24 位	

在 FIFO 模式下，可以根据交错控制寄存器选择逐步浏览方式和交错浏览方式。只有当 LCDPathEn=1 时，交错控制位才可用，否则交错控制位的值对 DMA 模式操作没有影响，只支持逐步浏览模式。

如果交错模式使能（LCDPathEn=1 或 Interlace=1），帧管理自动分为奇数区和偶数区。意思就是在同一帧内，不再需要用户中断完成内部领域转换。因此，逐步浏览模式和交错浏览模式的帧管理清单是相同的。如果选择用相机处理器输入数据，将不支持交错浏览。

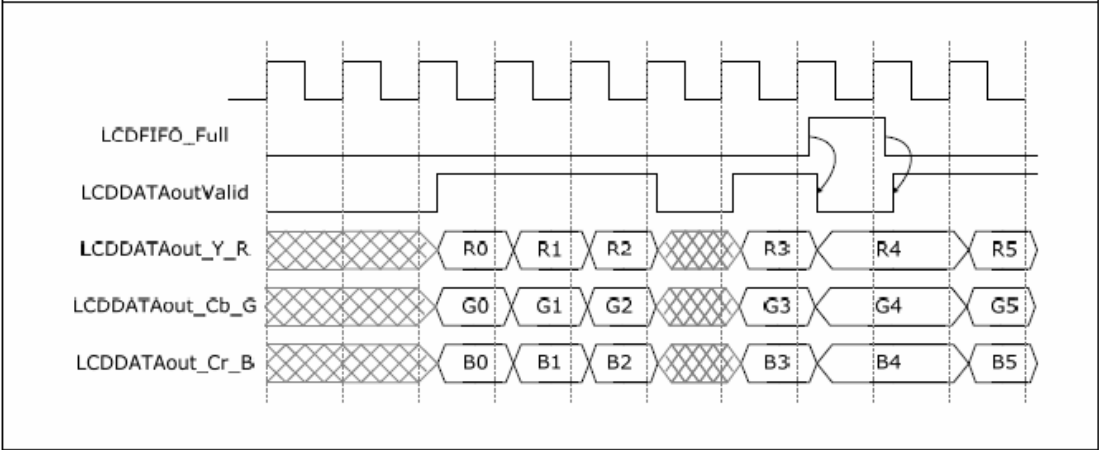


图 20-22 LCD 路径 的 I/O 时序图

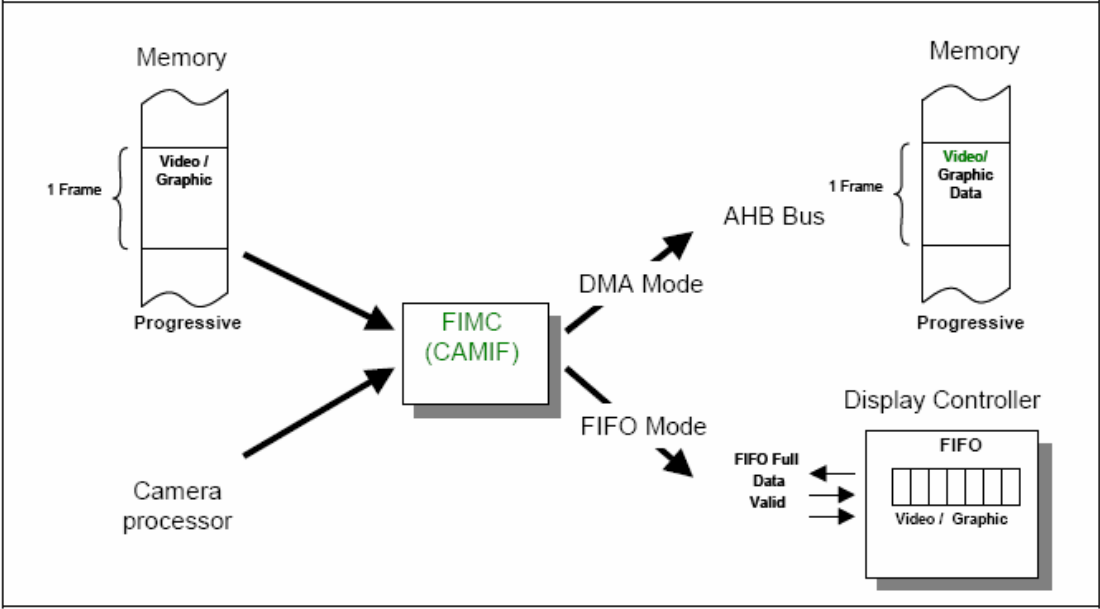


图 20-23 FIMC 内的两 2 输入和 2 输出模式

20.3.24.编解码器 DMA 目标空间寄存器

寄存器	地址	读/写	描述	复位值
CICOTAREA	0x7800005C	读/写	编解码器目标空间	0

CICOTAREA	位	描述	初始状态	M	L
Reserved	[31:26]		0	X	X
CICOTAREA	[25:0]	编解码器 DMA 目标空间=目标横向尺寸 × 目标纵向尺寸	0	0	0

20.3.25.编解码器状态寄存器

寄存器	地址	读/写	描述	复位值
CICOSTATUS	0x78000064	读/写	编解码路径状态	0

CICOSTATUS	位	描述	初始状态	M	L
OvFiY_Co	[31]	编解码器 FIFO Y 的溢出状态	0	X	X
OvFiCb_Co	[30]	编解码器 FIFO Cb 的溢出状态	0	X	X
OvFiCr_Co	[29]	编解码器 FIFO Cr 的溢出状态	0	X	X
VSYNC	[28]	相机 VSYNC	0	X	X
FrameCnt_Co	[27:26]	编解码器 DMA 的帧计数	0	X	X
WinOfstEn_Co	[25]	窗口补偿区使能状态	0	X	X
FlipMd_Co	[24:23]	编解码器 DMA 的 Flip 模式	0	X	X
ImgCptEn	[22]	全局相机接口的图像捕捉使能	0	X	X
ImgCptEn_CoSC	[21]	编解码器路径图像捕捉使能	0	X	X
VSYNC_A	[20]	外部相机 A VSYNC	X	X	X
Reserved	[19]		X	X	X
Reserved	[18]		X	X	X
FrameEnd_Co	[27:26]	结束编解码器帧操作后产生 FrameEnd_Co，用户设置 0 后讲清除 FrameEnd_Co。	0	X	X
Reserved	[16:0]		0	X	X

## 20.3.26.预览输出 Y1 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CIPRYSA1	0x7800006C	读/写	预览 DMA 的第一帧开始地址	0

CIPRYSA1	位	描述	初始状态	M	L
CIPRYSA1	[31:0]	非交错 Y，交错 YCbCr，RGB:第一帧开始地址	0	0	X

### 20.3.27, 预览输出 Y2 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CIPRYSA 2	0x78000070	读/写	预览 DMA 第二帧开始地址	0

CIPRYSA 2	位	描述	初始状态	M	L
CIPRYSA 2	[31:0]	无交错 Y, 交错 YCbCr, RGB: 第二帧开始地址	0	0	X

### 20.3.28.预览输出 Y3 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CIPRYSA 3	0x78000074	读/写	预览 DMA 第三帧开始地址	0

CIPRYSA 3	位	描述	初始状态	M	L
CIPRYSA 3	[31:0]	无交错 Y, 交错 YCbCr, RGB: 第三帧开始地址	0	0	X

### 20.3.29.预览输出 Y4 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CIPRYSA 4	0x78000078	读/写	预览 DMA 第四帧开始地址	0

CIPRYSA 4	位	描述	初始状态	M	L
CIPRYSA 4	[31:0]	无交错 Y, 交错 YCbCr, RGB: 第四帧开始地址	0	0	X

### 20.3.30.预览输出 CB1 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CIPRCSA1	0x7800007C	读/写	预览 DMA 的第一帧开始地址	0

CIPRCBSA1	位	描述	初始状态	M	L
CIPRCBSA1	[31:0]	预览 DMA 的 Cb 第一帧开始地址	0	0	X

### 20.3.31.预览输出 CB2 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CIPRCBSA 2	0x78000080	读/写	预览 DMA 第二帧开始地址	0

CIPRCBSA 2	位	描述	初始状态	M	L
CIPRCBSA 2	[31:0]	预览 DMA 的 Cb 第二帧开始地址	0	0	X

### 20.3.32.预览输出 CB3 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CIPRCBSA 3	0x78000084	读/写	预览 DMA 第三帧开始地址	0

CIPRCBSA 3	位	描述	初始状态	M	L
CIPRCBSA 3	[31:0]	预览 DMA 的 Cb 第三帧开始地址	0	0	X

### 20.3.33.预览输出 CB4 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CIPRCBSA 4	0x78000088	读/写	预览 DMA 第四帧开始地址	0

CIPRCBSA 4	位	描述	初始状态	M	L
CIPRCBSA 4	[31:0]	预览 DMA 的 Cb 第四帧开始地址	0	0	X



20.3.34.预览输出 CR1 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CIPRCRSA1	0x7800008C	读/写	预览 DMA 的第一帧开始地址	0

CIPRCRSA1	位	描述	初始状态	M	L
CIPRCRSA1	[31:0]	预览 DMA 的 CR 第一帧开始地址	0	0	X

20.3.35.预览输出 CR2 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CIPRCRSA 2	0x78000090	读/写	预览 DMA 第二帧开始地址	0

CIPRCRSA 2	位	描述	初始状态	M	L
CIPRCRSA 2	[31:0]	预览 DMA 的 CR 第二帧开始地址	0	0	X

20.3.36.预览输出 CR3 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CIPRCRSA 3	0x78000094	读/写	预览 DMA 第三帧开始地址	0

CIPRCRSA 3	位	描述	初始状态	M	L
CIPRCRSA 3	[31:0]	预览 DMA 的 CR 第三帧开始地址	0	0	X

20.3.37.预览输出 CR4 开始地址寄存器

寄存器	地址	读/写	描述	复位值
CIPRCRSA 4	0x78000098	读/写	预览 DMA 第四帧开始地址	0

CIPRCRSA 4	位	描述	初始状态	M	L
CIPRCRSA 4	[31:0]	预览 DMA 的 CR 第四帧开始地址	0	0	X

## 20.3.38.预览目标格式寄存器

寄存器	地址	读/写	描述	复位值
CIPRTRGFMT	0x7800009C	读/写	预览 DMA 的的目标图像格式	0000_0000

CIPRTRGFMT	位	描述	初始状态	M	L
Reserved	[31]		0	X	X
OutFormat_Pr	[30:29]	00:YCbCr4:2:0 预览输出图像格式（无交错） 01:YCbCr4:2:2 预览输出图像格式（无交错） 10:YCbCr4:2:2 预览输出图像格式（交错） 11:RGB 预览输出图像格式	0	0	0
TargetHsize_Pr	[28:16]	预览 DMA 目标图像的横向像素数（16 的倍数，最小值为 16）	0	0	0
FlipMd_Pr	[15:14]	预览 DMA 图像镜像和旋转 00: 常规 01: X 轴镜像 10: Y 轴镜像 11: 180° 旋转	0	0	0
Rot90_Pr	[13]	1: 顺时针旋转 90° 0: 旁路旋转			
TargetVsize_Pr	[12:0]	预览 DMA 目标图像的纵向像素数，最小值为 4.	0	0	0

TargetHsize\_Pr 和 TargetVsize\_Pr 不能大于相机 SourceHsize 和 SourceVsize。DMA 输入源尺寸可以忽略。行缓冲区的旋转尺寸是每行 720 字。（意思是当 RGB888/666 和 Rot90\_Pr=1 时，TargetHsize 为 720 像素）

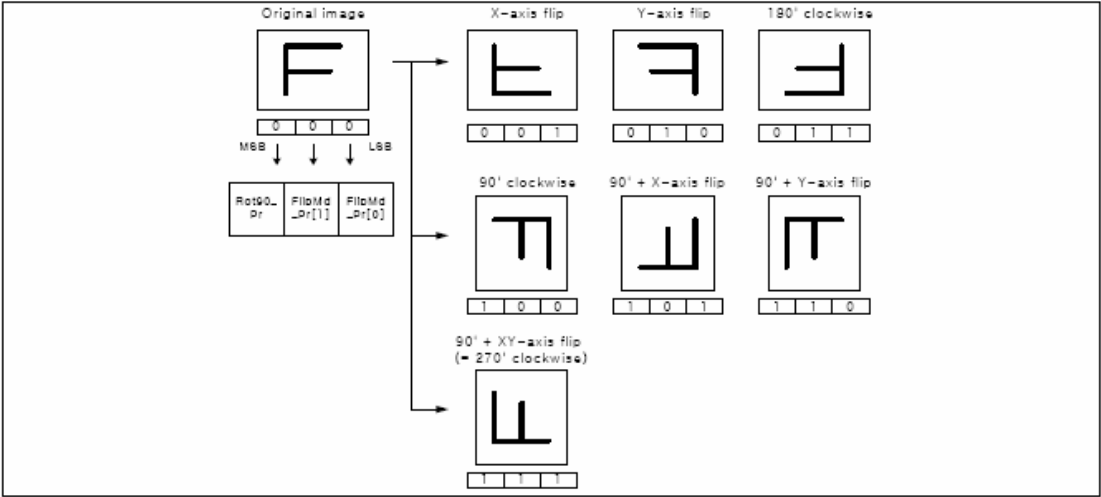


图 20-24 预览图像镜像和旋转

20.3.39.预览 DMA 控制寄存器

寄存器	地址	读/写	描述	复位值
CIPCTRL	0x780000A0	读/写	预览 DMA 控制相关寄存器	0

CIPCTRL	位	描述	初始状态	M	L
Reserved	[31:24]		0	X	X
Yburst1_Pr	[23:19]	预览 Y/RGB 帧的主突发长度	0	0	0
Yburst2_Pr	[18:14]	预览 Y/RGB 帧的 Remained burst 长度	0	0	0
Cburst1_Pr	[13:9]	预览 Cb/Cr 帧的主突发长度	0	0	0
Cburst2_Pr	[8:4]	预览 Cb/Cr 帧的 Remained burst 长度	0	0	0
Reserved	[3]		0	X	X
LastIRQEn_Pr	[2]	1:在帧捕捉末尾 Last IRQ enabled 0: 常规	0	X	X

Order422_Pr	[1:0]	使 YCbCr4:2:2 输出规则寄存器储存形式交错		0	0	0
		错				
			LSB                  MSB			
		00	Y <sub>0</sub> Cb <sub>0</sub> Y <sub>1</sub> Cr <sub>0</sub>			
		01	Y <sub>0</sub> Cr <sub>0</sub> Y <sub>1</sub> Cb <sub>0</sub>			
		10	Cb <sub>0</sub> Y <sub>0</sub> Cr <sub>0</sub> Y <sub>1</sub>			
		11	Cr <sub>0</sub> Y <sub>0</sub> Cb <sub>0</sub> Y <sub>1</sub>			

交错突发长度（交错 YCbCr4:2:2）

	Rot90_Pr=0	Rot90_Pr=1
Y 突发长度	2, 4, 8	2, 1
C 突发长度（C 突发长度=Y 突发长度/2）	1, 2, 4	无意义
希望的突发长度（=Y+2C）	4, 8, 16	4, 2

注释：当预览输出格式是 YCbCr4:2:2 交错格式时，ScalerBypass\_Pr=0，ScaleUp\_V\_Pr=1，不允许希望的主突发长度=16，希望的保持突发长度

≠16

20.3.40.预览预览缩放控制寄存器 1

寄存器	地址	读/写	描述	复位值
CIPRSCPRERATIO	0x780000A4	读/写	预览预览缩放比例控制	0

CIPRSCPRERATIO	位	描述	初始状态	M	L
SHfactor_Pr	[31:28]	预览预览缩放的转移因子	0	0	0
Reserved	[27:23]		0	X	X
PreHorRatio_Pr	[22:16]	预览预览缩放的横向比例	0	0	0
Reserved	[15:7]		0	X	X
PreVorRatio_Pr	[6:0]	预览预览缩放的纵向比例	0	0	0

### 20.3.41.预览预览缩放控制寄存器 2

寄存器	地址	读/写	描述	复位值
CIPRSCPREDST	0x780000A8	读/写	预览预览缩放目标格式	0

CIPRSCPRERATIO	位	描述	初始状态	M	L
Reserved	[31:28]		0	X	X
PreDstWidth_Pr	[27:16]	预览预览缩放的目标宽度	0	X	0
Reserved	[15:12]		0	X	X
PreDstHeight_Pr	[11:0]	预览预览缩放的目标高度	0	X	0

### 20.3.42. 预览主缩放控制寄存器

寄存器	地址	读/写	描述	复位值
CIPRSCCTRL	0x780000AC	读/写	预览主缩放控制	0x18000000

CIPRSCCTRL	位	描述	初始状态	M	L
ScalerBypass_Pr	[31]	预览缩放旁路。这种情况下，ImgCptEn_PrSC 必须为 0， ImgCptEn 必须为 1。  通常这种模式使用大型图片进行最大尺寸缩放。因此，不建议捕捉预览图像。这种模式适合用在 DSC 应用上，进行捕捉 JPEG 输入图像。  这种情况下，输入像素缓冲区仅有输入 FIFO 决定，因此系统总线不会很忙。  ScalerBypass 有一些约束，不可以进行尺寸缩放，不可以进行颜色空间转换，旋转，和 MSDMA 内存输入图像。只允许 YCbCr 非交错 4:2:0 和交错的 4:2:2 的输入输出格式。	0	0	0
ScaleUp_H_Pr	[30]	预览缩放的横向缩放大小标志。	0	0	0

		1: 大, 0: 小			
ScaleUp_V_Pr	[29]	预览缩放的纵向缩放大小标志。 1: 大, 0: 小	0	0	0
CSCR2Y_Pr	[28]	颜色空间 RGB 向 YCbCr 转换的 YCbCr 数据动态范围选择 1: 大范围=>Y/Cb/Cr(0~255):默认范围为大范围 0: 小范围=>Y(16~235), Cb/Cr(16~240)	1	0	0
CSCY2Y_Pr	[27]	颜色空间 YCbCr 向 RGB 转换的 YCbCr 数据动态范围选择 1: 大范围=>Y/Cb/Cr(0~255):默认范围为大范围 0: 小范围=>Y(16~235), Cb/Cr(16~240)	1	0	0
LCDPathEn_Pr	[26]	FIFO 模式使能。 1: FIFO 模式 0:DMA 模式 FIFO 模式输出格式是 YCbCr4:4:4 或 RGB24 位, 根据输出格式寄存器选择格式。OutFormat_Pr=RGB, 为 RGB24 位格式, 否则为 YCbCr4:4:4 输出格式。	0	0	0
Interlace_Pr	[25]	FIFO 模式下的输出浏览方式选择寄存器。1: 交错浏览, 0: 逐步浏览。 在 DMA 模式下, 无论此位设置什么值都进行逐步浏览。 当从相机处理器输入图像数据时不允许采用此模式。	0	0	0
MainHorRatio_Pr	[24:16]	预览主缩放的横向缩放比例	0	0	0
PrScalerStart	[15]	预览缩放开始, 在预览缩放旁路模式下, 此位必须为 0。 1: 开始缩放 0: 停止缩放	0	0	0
InRGB_FMT_Pr	[14:13]	向专用的预览路径输入 RGB 格式的 MSDMA 00:RGB565, 01:RGB666, 10:RGB888, 11: 保留	0	0	0
OutRGB_FMT_Pr	[12:11]	预览写入 DMA 的输出 RGB 格式 00:RGB565, 01:RGB666, 10:RGB888, 11: 保留	0	0	0
Ext_RGB_Pr	[10]	RGB565/666 模式向 RGB888 模式转换的预览路径的输入 RGB 数据扩展使能位 1: 扩展 0: 常规 1) 输入 R=5 位, 在 RGB565 模式下	0	0	0

		10100->10100101(扩展) 10100->10100000(常规) 2) 输入 R=6 位, 在 RGB666 模式下 10100->10110010(扩展) 10100->10110000(常规)			
One2One_Pr	[9:0]	非插值数据复制（警告：应该设置寄存器位 1：相同输入输出格式的缩放尺寸，图像效果不支持 RGB 格式和 One2One 格式）	0	0	0
MainVerRatio_Pr	[8:0]	预览主缩放的纵向缩放尺寸	0	0	0

20.3.43.预览 DMA 目标空间寄存器

寄存器	地址	读/写	描述	复位值
CIPRTAREA	0x780000B0	读/写	预览目标空间	0

CIPRTAREA	位	描述	初始状态	M	L
Reserved	[31:26]		0	X	X
CIPRTAREA	[25:0]	预览 DMA 目标空间=目标横向尺寸×目标纵向尺寸	0	0	X

20.3.44.预览状态寄存器

寄存器	地址	读/写	描述	复位值
CIPRSTATUS	0x780000B8	读/写	预览路径状态	0

CIPRSTATUS	位	描述	初始状态	M	L
OvFiY_Pr	[31]	预览 FIFO Y 的溢出状态	0	X	X
OvFiCb_Pr	[30]	预览 FIFO Cb 的溢出状态	0	X	X

OvFiCr_Pr	[29]	预览 FIFO Cr 的溢出状态	0	X	X
Reserved	[28]		0	X	X
FrameCnt_Pr	[27:26]	预览 DMA 的帧计数	0	X	X
Reserved	[25]		0	X	X
FlipMd_Pr	[24:23]	预览 DMA 的 Flip 模式	0	X	X
Reserved	[22]		0	X	X
ImgCptEn_PrSC	[21]	预览路径图像捕捉使能	0	X	X
OvRLB_Pr	[20]	在预览路径内用于旋转的行缓冲区的溢出状态	0	X	X
FrameEnd_Pr	[19]	结束预览帧操作后产生 FrameEnd_Pr，用户设置 0 后讲清除 FrameEnd_Pr。	0	X	X
Reserved	[18:0]		X	X	X

## 20.3.45.图像捕捉使能寄存器

寄存器	地址	读/写	描述	复位值
CIMGCPT	0x780000C0	读/写	图像捕捉使能命令	0

CIMGCPT	位	描述	初始状态	M	L
ImgCptEn	[31]	相机接口全局步骤使能	0	X	0
ImgCptEn_CoSC	[30]	编解码器缩放捕捉使能。在编解码缩放旁路模式下，此位必须为 0.	0	X	0
ImgCptEn_PrSC	[21]	预览缩放捕捉使能。在预览缩放旁路模式下，此位必须为 0.	0	0	0
Reserved	[28:26]		0	X	X
Cpt_FrEn_Co	[25]	捕捉编解码器帧控制（只适用相机输入） 1:enabled（一步一步帧连拍模式） 0: disable（自由运行模式）	0	X	0



Cpt_FrEn_Pr	[24]	捕捉预览帧控制（只适用相机输入） 1:enabled（一步一步帧连拍模式） 0: disable（自由运行模式）	0	X	0
Cpt_FrPtr	[23:19]	捕捉系列扭转指针（预览与编解码器通用）	0	X	X
Cpt_FrMod	[18]	捕捉帧控制模式（预览与编解码器通用） 1: 应用 Cpt_FrCnt 模式 0: 应用 Cpt_FrEn 模式	0	X	X
Cpt_FrCnt	[17:10]	捕捉希望的帧序号。当读操作时，可以看到阴影寄存器的值，当捕捉帧后降计数。	0	X	X
Reserved	[9:0]		0	X	X

20.3.46.捕捉控制序列寄存器

寄存器	地址	读/写	描述	复位值
CICPTSEQ	0x780000C4	读/写	相关的相机图像捕捉序列	FFFF_FFFF

CICPTSEQ	位	描述	初始状态	M	L
Cpt_FrSeq	[31:0]	相机序列形式	FFFF_FFFF	X	X

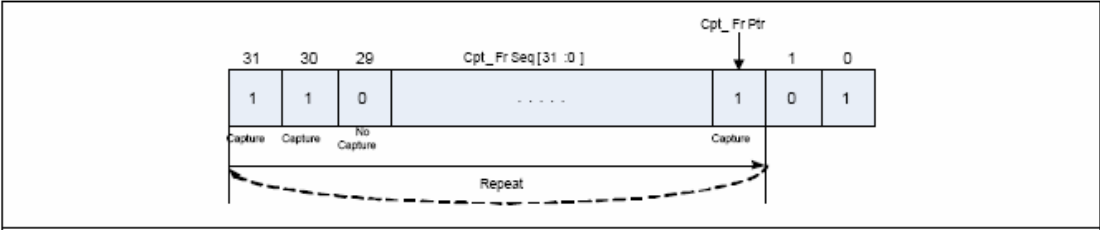


图 20-25 捕捉帧控制

20.3.47. 图像效果寄存器

寄存器	地址	读/写	描述	复位值
-----	----	-----	----	-----

CIIMGEFF	0x780000D0	读/写	相关的图像效果	0010_0080
----------	------------	-----	---------	-----------

CIIMGEFF	位	描述	初始状态	M	L
IE_ON_Pr	[31]	0: 在预览路径内图像效果功能 disable 1: enable	0	0	0
IE_ON_Pr	[30]	0: 在编解码器路径内图像效果功能 disable 1: enable	0	0	0
IE_AFTER_SC	[29]	图像效果位置 1: 在缩放之后 2: 在缩放之前	0	0	0
FIN	[28:26]	图像效果选择 3'd0: Bypass 3'd1: Arbitrary Cb/Cr 3'd2: Negative 3'd3: Art Freeze 3'd4: Embossing 3'd5: Silhouette	0	0	0
Reserved	[25:21]		0	X	X
PAT_Cb	[20:13]	只用于 FIN，而且是任意 Cb/Cr。 大的 CSC 范围: $0 \leq \text{PAT\_Cb} \leq 255$ 小的 CSC 方位: $16 \leq \text{PAT\_Cb} \leq 240$	8'd128	0	0
Reserved	[12:8]		0	X	X
PAT_Cr	[7:0]	只用于 FIN，而且是任意 Cb/Cr。 大的 CSC 范围: $0 \leq \text{PAT\_Cb} \leq 255$ 小的 CSC 方位: $16 \leq \text{PAT\_Cb} \leq 240$	8'd128	0	0

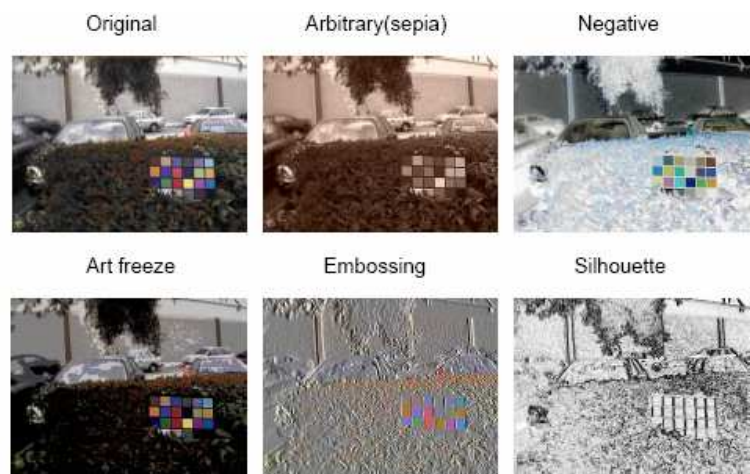


图 20-26 图像效果说明

### 20.3.48.编解码器的 MSDMA Y 开始地址寄存器

寄存器	地址	读/写	描述	复位值
MSCOY0SA	0x780000D4	读/写	相关的 MSDMA Y0 开始地址	0000_0000

MSCOY0SA	位	描述	初始状态	M	L
Reserved	[31]		0	X	X
MSCOY0SA	[30:0]	Y 组成部分的 DMA 开始地址 交错 YCbCr4:2:2 或 RGB 组成部分的 DMA 开始地址	0	0	X

### 20.3.49.编解码器的 MSDMA CB 开始地址寄存器

寄存器	地址	读/写	描述	复位值
MSCOCB0SA	0x780000D8	读/写	相关的 MSDMA CB0 开始地址	0000_0000

MSCOCB0SA	位	描述	初始状态	M	L
Reserved	[31]		0	X	X

MSCOCB0SA	[30:0]	Cb 组成部分的 DMA 开始地址	0	0	X
-----------	--------	-------------------	---	---	---

## 20.3.50.编解码器的 MSDMA CR 开始地址寄存器

寄存器	地址	读/写	描述	复位值
MSCOCR0SA	0x780000DC	读/写	相关的 MSDMA CR0 开始地址	0000_0000

MSCOCR0SA	位	描述	初始状态	M	L
Reserved	[31]		0	X	X
MSCOCR0SA	[30:0]	CR 组成部分的 DMA 开始地址	0	0	X

## 20.3.51.编解码器的 MSDMA Y 结束地址寄存器

寄存器	地址	读/写	描述	复位值
MSCOY0END	0x780000E0	读/写	相关的 MSDMA Y0 结束地址	0000_0000

MSCOY0END	位	描述	初始状态	M	L
Reserved	[31]		0	X	X
MSCOY0END	[30:0]	Y 组成部分的 DMA 结束地址 交错 YCbCr4:2:2 或 RGB 组成部分的 DMA 结束地址	0	0	X

## 20.3.52.编解码器的 MSDMA CB 结束地址寄存器

寄存器	地址	读/写	描述	复位值
MSCOCB0END	0x780000E4	读/写	相关的 MSDMA CB0 结束地址	0000_0000

MSCOCB0END	位	描述	初始状态	M	L
Reserved	[31]		0	X	X

MSCOCB0END	[30:0]	Cb 组成部分的 DMA 结束地址	0	0	X
------------	--------	-------------------	---	---	---

### 20.3.53.编解码器的 MSDMA CR 结束地址寄存器

寄存器	地址	读/写	描述	复位值
MSCOCR0END	0x780000E8	读/写	相关的 MSDMA CR0 结束地址	0000_0000

MSCOCR0END	位	描述	初始状态	M	L
Reserved	[31]		0	X	X
MSCOCR0END	[30:0]	CR 组成部分的 DMA 结束地址	0	0	X

### 20.3.54.编解码器的 MSDMA Y 补偿区寄存器

寄存器	地址	读/写	描述	复位值
MSCOYOFF	0x780000EC	读/写	相关的 MSDMA Y 补偿区	0000_0000

MSCOYOFF	位	描述	初始状态	M	L
Reserved	[31:24]		0	X	X
MSCOYOFF	[23:0]	源图像 Y 组成部分的补偿区 源图像 交错 YCbCr4:2:2 或 RGB 组成部分的补偿区	0	0	X

### 20.3.55.编解码器的 MSDMA CB 补偿区寄存器

寄存器	地址	读/写	描述	复位值
MSCOCBOFF	0x780000F0	读/写	相关的 MSDMA CB 补偿区	0000_0000

MSCOCBOFF	位	描述	初始状态	M	L
Reserved	[31:24]		0	X	X

MSCOCB00FF	[23:0]	源图像 Cb 组成部分的补偿区	0	0	X
------------	--------	-----------------	---	---	---

## 20.3.56.编解码器的 MSDMA CR 补偿区寄存器

寄存器	地址	读/写	描述	复位值
MSCOCROFF	0x780000F4	读/写	相关的 MSDMA CR 补偿区	0000_0000

MSCOCROFF	位	描述	初始状态	M	L
Reserved	[31:24]		0	X	X
MSCOCR0FF	[23:0]	源图像 CR 组成部分的补偿区	0	0	X

## 20.3.57. 编解码器的 MADMA 源图像宽度寄存器

寄存器	地址	读/写	描述	复位值
MSCOWIDTH	0x780000F8	读/写	相关的 MSDMA 源图像宽度	0000_0000

MSCOWIDTH	位	描述	初始状态	M	L
AutoLoadEnable	[31]	在第一帧开始请求 ENVID 开始设置时，MADMA 自动重新开始（只真对软件触发模式）。在第一帧之后，下一帧不需要 ENVID 设置。 0：不能自动下载 1：可以自动下载	0	0	X
ADDR_CH_DIS	[30]	MSDMA 地址改变使能（只针对软件触发模式） 0：可以改变地址 1：不能改变地址	0	0	X
Reserved	[29:28]		0	X	X
MSCOHEIGHT	[27:16]	MSDMA 源图像纵向像素尺寸。最小值是 8。必须是 PreVerRatio 的倍数。	0	0	X
Reserved	[15:12]		0	X	X
MSCOWIDTH	[11:0]	MSDMA 源图像横向像素尺寸。（必须为 8 的倍数，必须是	0	0	X

		PreHorRatio 4 的 倍数，最小值是 16)			
--	--	-----------------------------	--	--	--

20.3.58.编解码器的 MSDMA 控制寄存器

寄存器	地址	读/写	描述	复位值
MSCOCTRL	0x780000FC	读/写	编解码器的 MSDMA 控制寄存器	0000_0000

MSCOCTRL	位	描述	初始状态	M	L										
Reserved	[31:7]		0	X	X										
EOF_M_C	[6]	当 MSDMA 运行完成后，将产生 End Of Frame（只读信号）	0	0	X										
Order422_M_C	[5:4]	当源 MSDMA 图像是交错 YCbCr4:2:2 时，交错 YCbCr4:2:2 输入顺序类型。 <table><tr><td>[4:3]</td><td>LSB      MSB</td></tr><tr><td>00</td><td>Y<sub>0</sub>Cb<sub>0</sub>Y<sub>1</sub>Cr<sub>0</sub></td></tr><tr><td>01</td><td>Y<sub>0</sub>Cr<sub>0</sub>Y<sub>1</sub>Cb<sub>0</sub></td></tr><tr><td>10</td><td>Cb<sub>0</sub>Y<sub>0</sub>Cr<sub>0</sub>Y<sub>1</sub></td></tr><tr><td>11</td><td>Cr<sub>0</sub>Y<sub>0</sub>Cb<sub>0</sub>Y<sub>1</sub></td></tr></table>	[4:3]	LSB      MSB	00	Y <sub>0</sub> Cb <sub>0</sub> Y <sub>1</sub> Cr <sub>0</sub>	01	Y <sub>0</sub> Cr <sub>0</sub> Y <sub>1</sub> Cb <sub>0</sub>	10	Cb <sub>0</sub> Y <sub>0</sub> Cr <sub>0</sub> Y <sub>1</sub>	11	Cr <sub>0</sub> Y <sub>0</sub> Cb <sub>0</sub> Y <sub>1</sub>	0	0	X
[4:3]	LSB      MSB														
00	Y <sub>0</sub> Cb <sub>0</sub> Y <sub>1</sub> Cr <sub>0</sub>														
01	Y <sub>0</sub> Cr <sub>0</sub> Y <sub>1</sub> Cb <sub>0</sub>														
10	Cb <sub>0</sub> Y <sub>0</sub> Cr <sub>0</sub> Y <sub>1</sub>														
11	Cr <sub>0</sub> Y <sub>0</sub> Cb <sub>0</sub> Y <sub>1</sub>														
SEL_DMA_CAM_C	[3]	编解码器路径输入数据选择  0：外部相机输入路径  1：存储器数据输入路径（MSDMA）	0	0	X										
InFormat_M_C	[2:1]	MSDMA 源图像格式  00：YCbCr4:2:0  01：YCbCr4:2:0（非交错）  10：YCbCr4:2:2（交错）  11：RGB	0	0	X										
ENVID_M_C	[0]	MSDMA 操作开始。硬件不会自动清零。这个寄存器只对软件触发模式有效。如果是硬件触发模式，此位是只读形式。  1） SEL_DMA_CAM=0, ENVID 不需要考虑  2） SEL_DMA_CAM=1, ENVID 被设置（由 0 到 1），然后 MSDMA	0	0	X										

		开始向编解码器运行。			
--	--	------------	--	--	--

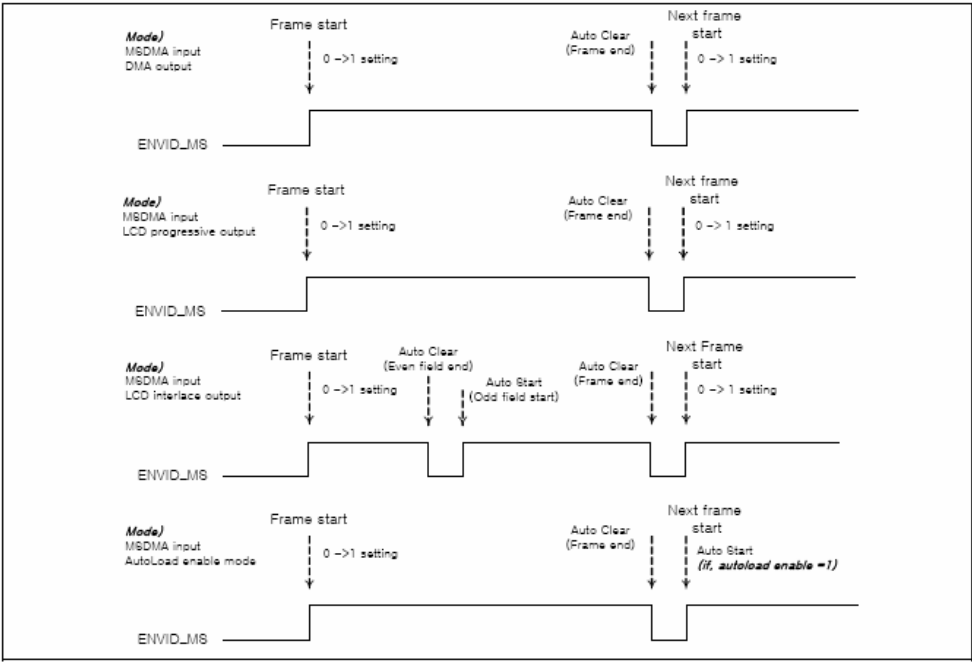


图 20-27 DMS 开始读取存储器数据时的 ENVID\_MS SFR 设置

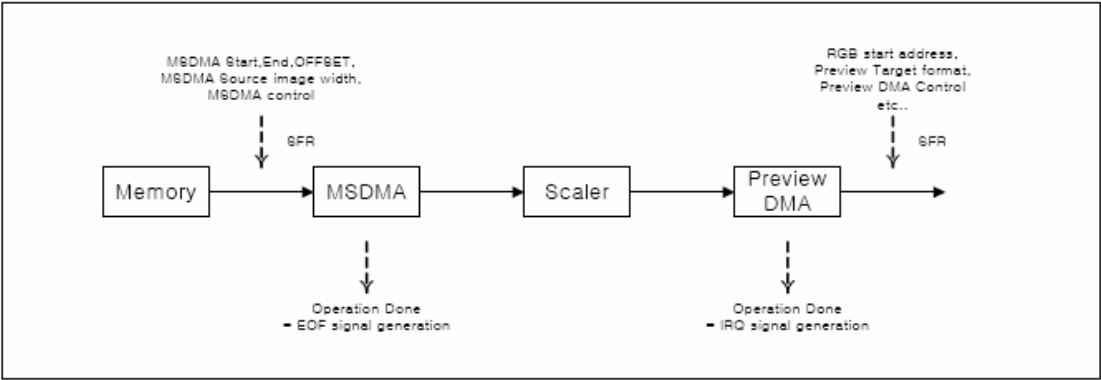


图 20-28 SFR&Operation（选择 MSDMA 输入路径时与每个 DMA 相关）



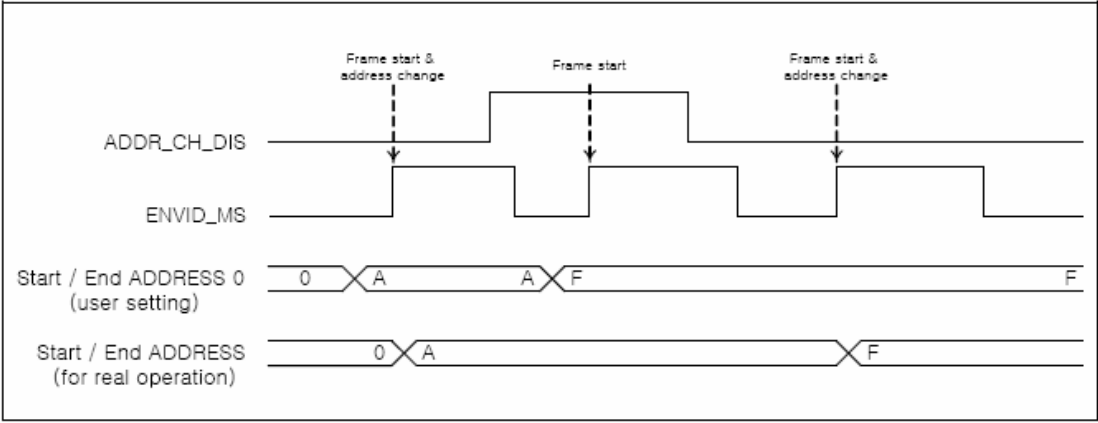


图 20-29 地址改变时序图（与 MSDMA 相关）

20.3.59.预览的 MSDMA Y0 开始地址寄存器

寄存器	地址	读/写	描述	复位值
MSPRY0SA	0x78000100	读/写	相关的 MSDMA Y0 开始地址	0000_0000

MSYSA	位	描述	初始状态	M	L
Reserved	[31]		0	X	X
MSPRY0SA	[30:0]	Y 组成部分的 DMA 开始地址 交错 YCbCr4:2:2 或 RGB 组成部分的 DMA 开始地址	0	0	X

20.3.60.预览的 MSDMA CB0 开始地址寄存器

寄存器	地址	读/写	描述	复位值
MSPRCB0SA	0x78000104	读/写	相关的 MSDMA CB0 开始地址	0000_0000

MSCBSA	位	描述	初始状态	M	L
Reserved	[31]		0	X	X
MSPRCB0SA	[30:0]	Cb 组成部分的 DMA 开始地址	0	0	X

### 20.3.61.预览的 MSDMA CR0 开始地址寄存器

寄存器	地址	读/写	描述	复位值
MSPRCR0SA	0x78000108	读/写	相关的 MSDMA CR0 开始地址	0000_0000

MSPRCRSA	位	描述	初始状态	M	L
Reserved	[31]		0	X	X
MSPRCR0SA	[30:0]	CR 组成部分的 DMA 开始地址	0	0	X

### 20.3.62.预览的 MSDMA Y0 结束地址寄存器

寄存器	地址	读/写	描述	复位值
MSPRY0END	0x7800010C	读/写	相关的 MSDMA Y0 结束地址	0000_0000

MSPRY0END	位	描述	初始状态	M	L
Reserved	[31]		0	X	X
MSPRY0END	[30:0]	Y 组成部分的 DMA 结束地址 交错 YCbCr4:2:2 或 RGB 组成部分的 DMA 结束地址	0	0	X

### 20.3.63.预览的 MSDMA CB 结束地址寄存器

寄存器	地址	读/写	描述	复位值
MSPRCB0END	0x78000110	读/写	相关的 MSDMA CB0 结束地址	0000_0000

MSPRCB0END	位	描述	初始状态	M	L
Reserved	[31]		0	X	X
MSPRCB0END	[30:0]	Cb 组成部分的 DMA 结束地址	0	0	X

## 20.3.64.预览的 MSDMA CR 结束地址寄存器

寄存器	地址	读/写	描述	复位值
MSPRCR0END	0x78000114	读/写	相关的 MSDMA CR0 结束地址	0000_0000

MSPRCR0END	位	描述	初始状态	M	L
Reserved	[31]		0	X	X
MSPRCR0END	[30:0]	CR 组成部分的 DMA 结束地址	0	0	X

## 20.3.65.预览的 MSDMA Y 补偿区寄存器

寄存器	地址	读/写	描述	复位值
MSPRYOFF	0x78000118	读/写	相关的 MSDMA Y 补偿区	0000_0000

MSPRYOFF	位	描述	初始状态	M	L
Reserved	[31:24]		0	X	X
MSPRYOFF	[23:0]	源图像 Y 组成部分的补偿区 源图像 交错 YCbCr4:2:2 或 RGB 组成部分的补偿区	0	0	X

## 20.3.66.预览的 MSDMA CB 补偿区寄存器

寄存器	地址	读/写	描述	复位值
MSPRCBOFF	0x7800011C	读/写	相关的 MSDMA CB0 补偿区	0000_0000

MSPRCBOFF	位	描述	初始状态	M	L
Reserved	[31:24]		0	X	X
MSPRCBOFF	[23:0]	源图像 Cb 组成部分的补偿区	0	0	X

## 20.3.67.预览的 MSDMA CR 补偿区寄存器

寄存器	地址	读/写	描述	复位值
MSPRCROFF	0x78000120	读/写	相关的 MSDMA CR0 补偿区	0000_0000

MSPRCROFF	位	描述	初始状态	M	L
Reserved	[31:24]		0	X	X
MSPRCROFF	[23:0]	源图像 CR 组成部分的补偿区	0	0	X

## 20.3.68. 预览的 MADMA 源图像宽度寄存器

寄存器	地址	读/写	描述	复位值
MSPRWIDTH	0x78000124	读/写	相关的 MSDMA 源图像宽度	0000_0000

MSPRWIDTH	位	描述	初始状态	M	L
AutoLoadEnable	[31]	在第一帧开始请求 ENVID 开始设置时, MADMA 自动重新开始(只真对软件触发模式)。在第一帧之后,下一帧不需要 ENVID 设置。 0: 不能自动下载 1: 可以自动下载	0	0	X
ADDR_CH_DIS	[30]	MSDMA 地址改变使能(只针对软件触发模式) 0: 可以改变地址 1: 不能改变地址	0	0	X
Reserved	[29:28]		0	X	X
MSPRHEIGHT	[27:16]	MSDMA 源图像纵向像素尺寸。最小值是 8。必须是 PreVerRatio 的倍数。	0	0	X
Reserved	[15:12]		0	X	X

MSPRWIDTH	[11:0]	MSDMA 源图像横向像素尺寸。（必须为 8 的倍数，必须是 PreHorRatio 4 的 倍数，最小值是 16）	0	0	X
-----------	--------	--	---	---	---

### 20.3.69.预览的 MSDMA 控制寄存器

寄存器	地址	读/写	描述	复位值
MSPRCTRL	0x78000128	读/写	预览的 MSDMA 控制寄存器	0000_0000

MSPRCTRL	位	描述	初始状态	M	L										
Reserved	[31:7]		0	X	X										
EOF_M_P	[6]	当 MSDMA 运行完成后，将产生 End Of Frame（只读信号）	0	0	X										
Order422_M_P	[5:4]	当源 MSDMA 图像是交错 YCbCr4:2:2 时，交错 YCbCr4:2:2 输入顺序类型。 <table><tr><td>[4:3]</td><td>LSB      MSB</td></tr><tr><td>00</td><td>Y<sub>0</sub>Cb<sub>0</sub>Y<sub>1</sub>Cr<sub>0</sub></td></tr><tr><td>01</td><td>Y<sub>0</sub> Cr<sub>0</sub>Y<sub>1</sub> Cb<sub>0</sub></td></tr><tr><td>10</td><td>Cb<sub>0</sub>Y<sub>0</sub> Cr<sub>0</sub>Y<sub>1</sub></td></tr><tr><td>11</td><td>Cr<sub>0</sub>Y<sub>0</sub>Cb<sub>0</sub>Y<sub>1</sub></td></tr></table>	[4:3]	LSB      MSB	00	Y <sub>0</sub> Cb <sub>0</sub> Y <sub>1</sub> Cr <sub>0</sub>	01	Y <sub>0</sub> Cr <sub>0</sub> Y <sub>1</sub> Cb <sub>0</sub>	10	Cb <sub>0</sub> Y <sub>0</sub> Cr <sub>0</sub> Y <sub>1</sub>	11	Cr <sub>0</sub> Y <sub>0</sub> Cb <sub>0</sub> Y <sub>1</sub>	0	0	X
[4:3]	LSB      MSB														
00	Y <sub>0</sub> Cb <sub>0</sub> Y <sub>1</sub> Cr <sub>0</sub>														
01	Y <sub>0</sub> Cr <sub>0</sub> Y <sub>1</sub> Cb <sub>0</sub>														
10	Cb <sub>0</sub> Y <sub>0</sub> Cr <sub>0</sub> Y <sub>1</sub>														
11	Cr <sub>0</sub> Y <sub>0</sub> Cb <sub>0</sub> Y <sub>1</sub>														
SEL_DMA_CAM_P	[3]	预览路径输入数据选择  0：外部相机输入路径  1：存储器数据输入路径（MSDMA）	0	0	X										
InFormat_M_P	[2:1]	MSDMA 源图像格式  00：YCbCr4:2:0  01：YCbCr4:2:0（非交错）  10：YCbCr4:2:2（交错）  11：RGB	0	0	X										
ENVID_M_P	[0]	MSDMA 操作开始。硬件不会自动清零。这个寄存器只对软件触发模式有效。如果是硬件触发模式，此位是只读形式。  3） SEL_DMA_CAM=0, ENVID 不需要考虑	0	0	X										

		4) SEL_DMA_CAM=1, ENVID 被设置（由 0 到 1），然后 MSDMA 开始向预览运行。			
--	--	--	--	--	--

### 20.3.70.编解码器浏览行 Y 补偿区寄存器

寄存器	地址	读/写	描述	复位值
CICOSCOSY	0x7800012C	读/写	相关的编解码器浏览行 Y 补偿区	0

CICOSCOSY	位	描述	初始状态	M	L
Reserved	[31:29]		0	X	X
Initial_Yoffset_Co	[28:16]	初始 Y 补偿区跳跃像素数值。浏览行 Y 补偿区可以用非交错 Y 或交错 YCbCr422 或 RGB 格式。	0	X	0
Reserved	[15:13]		0	X	X
Line_Yoffset_Co	[12:0]	当浏览行改变时，目标图像屏幕上跳跃像素的数值。当非交错 Y 或交错 YCbCr422 或 RGB 格式时可以用浏览行 Y 补偿区。	0	X	0

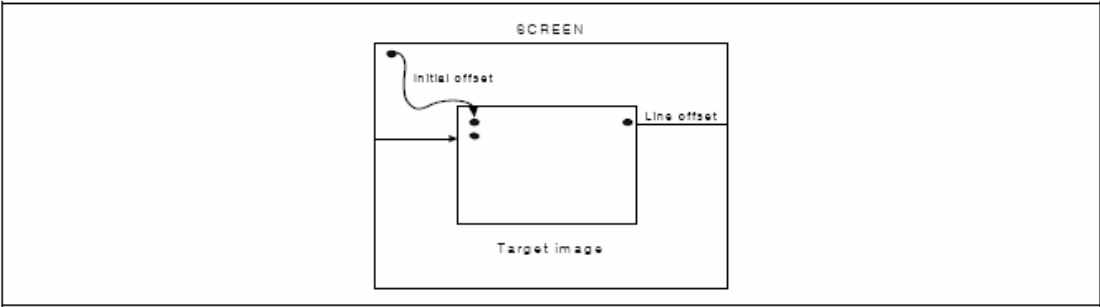


图 20-30 浏览行补偿区

### 20.3.71.编解码器浏览行 CB 补偿区寄存器

寄存器	地址	读/写	描述	复位值
CICOSCOSCB	0x78000130	读/写	相关的编解码器浏览行 CB 补偿区	0

CICOSCOS	位	描述	初始状态	M	L
Reserved	[31:29]		0	X	X
Initial_Yoffset_Co	[28:16]	初始 CB 补偿区的跳跃像素的数值。当非交错 YCbCr4:2:0/4:2:2 时可以使用浏览行 CB 补偿区。	0	X	0
Reserved	[15:13]		0	X	X
Line_Yoffset_Co	[12:0]	当浏览行改变时，目标图像屏幕上跳跃像素的数值。当非交错 YCbCr4:2:0/4:2:2 格式时，可以使用浏览行 CB 补偿区。	0	X	0

20.3.72.编解码器浏览行 CR 补偿区寄存器

寄存器	地址	读/写	描述	复位值
CICOSCOSCR	0x78000134	读/写	相关的编解码器浏览行 CR 补偿区	0

CICOSCOS	位	描述	初始状态	M	L
Reserved	[31:29]		0	X	X
Initial_Yoffset_Co	[28:16]	初始 CR 补偿区的跳跃像素的数值。当非交错 YCbCr4:2:0/4:2:2 时可以使用浏览行 CB 补偿区。	0	X	0
Reserved	[15:13]		0	X	X
Line_Yoffset_Co	[12:0]	当浏览行改变时，目标图像屏幕上跳跃像素的数值。当非交错 YCbCr4:2:0/4:2:2 格式时，可以使用浏览行 CB 补偿区。	0	X	0

20.3.73.预览浏览行 Y 补偿区寄存器

寄存器	地址	读/写	描述	复位值
CIPRSCOSY	0x78000138	读/写	相关的预览浏览行 Y 补偿区	0

CIPRSCOSY	位	描述	初始状态	M	L
-----------	---	----	------	---	---

Reserved	[31:29]		0	X	X
Initial_Yoffset_Pr	[28:16]	初始 Y 补偿区跳跃像素数值。浏览行 Y 补偿区可以用非交错 Y 或交错 YCbCr422 或 RGB 格式。	0	X	0
Reserved	[15:13]		0	X	X
Line_Yoffset_Pr	[12:0]	当浏览行改变时，目标图像屏幕上跳跃像素的数值。当非交错 Y 或交错 YCbCr422 或 RGB 格式时可以用浏览行 Y 补偿区。	0	X	0

20.3.74.预览浏览行 CB 补偿区寄存器

寄存器	地址	读/写	描述	复位值
CIPRSCOSCB	0x7800013C	读/写	相关的预览浏览行 CB 补偿区	0

CIPRSCOS	位	描述	初始状态	M	L
Reserved	[31:29]		0	X	X
Initial_Yoffset_Pr	[28:16]	初始 CB 补偿区的跳跃像素的数值。当非交错 YCbCr4:2:0/4:2:2 时可以使用浏览行 CB 补偿区。	0	X	0
Reserved	[15:13]		0	X	X
Line_Yoffset_Pr	[12:0]	当浏览行改变时，目标图像屏幕上跳跃像素的数值。当非交错 YCbCr4:2:0/4:2:2 格式时，可以使用浏览行 CB 补偿区。	0	X	0

20.3.75.预览浏览行 CR 补偿区寄存器

寄存器	地址	读/写	描述	复位值
CIPRSCOSCR	0x78000140	读/写	相关的预览浏览行 CR 补偿区	0

CIPRSCOS	位	描述	初始状态	M	L
Reserved	[31:29]		0	X	X
Initial_Yoffset_Pr	[28:16]	初始 CR 补偿区的跳跃像素的数值。当非交错 YCbCr4:2:0/4:2:2 时可以使用浏览行 CB 补偿区。	0	X	0



Reserved	[15:13]		0	X	X
Line_Yoffset_Pr	[12:0]	当浏览行改变时，目标图像屏幕上跳跃像素的数值。当非交错 YCbCr4:2:0/4:2:2 格式时，可以使用浏览行 CB 补偿区。	0	X	0

## 21 多格式视频编解码器

FIMV-MFC 1.0 版本是一个高能的视频编解码器 IP，它支持 H.263P3，MPEG-4 SP，H.264 和 VC-1。FIMV-MFC 1.0 版本由嵌入式位处理器和视频编解码器核心模块组成。该位处理器解析或构成位流，并控制视频编解码器。加快位流处理，在位处理器中包括一些硬件加速器。通过 AMBA APB 总线和 AMBA AXI 总线下载位处理器的程序和数据。如图 21-1 所示，显示 FIMV-MFC 1.0 版结构框图。

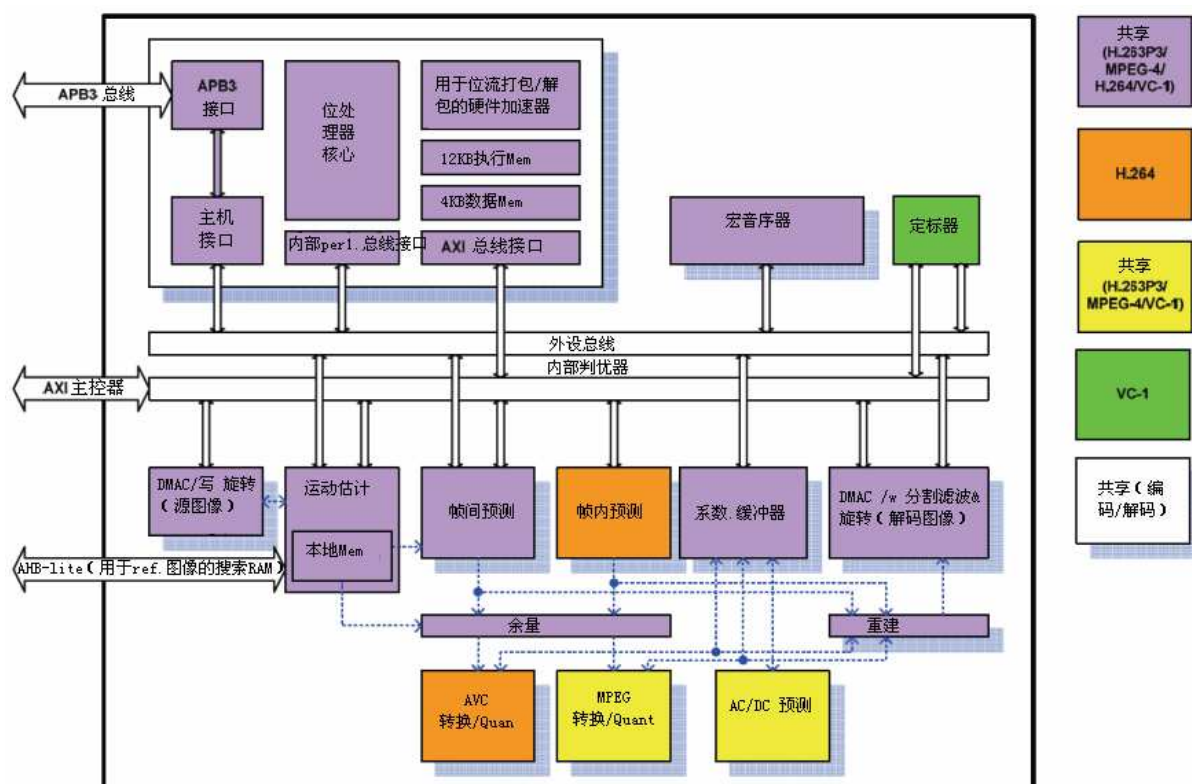


图 21-1 FIMV-MFC 1.0 版结构框图

FIMV-MFC V1.0 视频编解码器被优化，以减少逻辑门的数目。它具有共用大部分的子模块，用于多重标准。运动估值模块采用搜索 RAM 来减少外部 SDRAM 上的带宽。一般来说，运动估值通过多次读取相关像素数据。运动估值模块加载的相关像素数据来自外部 SDRAM，并将它们存储到搜索 RAM 中。通过 AMBA AHB 存取搜索 RAM。FIMV-MFC 1.0 版本包括一个旋转/镜像模块。在编码器旋转和/或镜像初始图像时，无需请求附加的带宽。然而在解码器中，带有任一旋转和/或镜像的解码图像都将被写入外部存储

器。

内部 AXI 判优器模块作出公断请求，从内部 DMA 控制器到用户 SoC。

如图 21-2 所示，描述位处理的任务，视频编解码器的核心模块以及如何连接应用软件。基本上，主机处理器以帧的形式通过提供的 API 和 FIMV-MFC 1.0 进行通信。给视频编解码器更大的灵活性和调试的能力，与位流相关的所有的处理被指派到位处理器中。

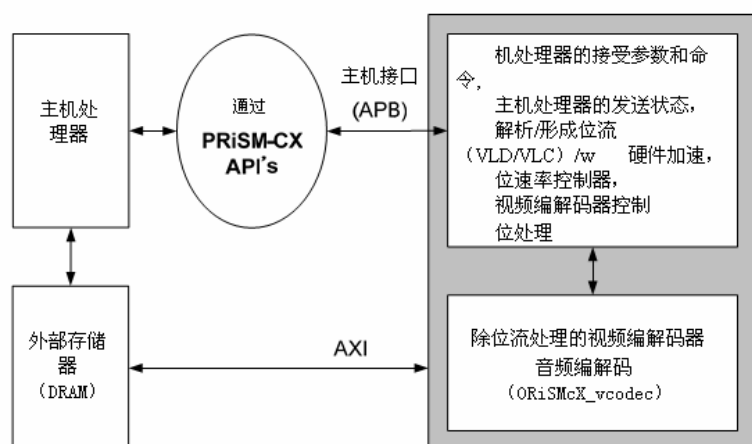


图 21-2 位处理器的任务和视频编解码器模块

## 21.1 特征

FIMV-MFC 1.0 版本是一个多标准视频编解码器的 IP，它能够处理个别编解码器元件的 H.263P3，MPEG-4 SP，H.264 BP 和 VC-1 MP。

FIMV-MFC 1.0 版本包括以下功能：

### 1. 多标准视频编解码器

- (1) MPEG-4 部分 2 应用简单文件的编码/解码。
- (2) H. 264/AVC 基线应用文件编码/解码。
- (3) H. 263 P3 编码/解码。
- (4) VC-1 (WMV9) 主要应用文件解码。
- (5) 支持多方呼叫。

如：同时发生 1 流解码和 3 流解码是可能的。

- (6) 支持多格式。

如：视频 IP 同时编码成 MPEG-4 位流和解码成 H. 264 位流。

## 2. 编码工具

- (1)  $[-16, +16]$   $1/2$  和  $1/4$  像素精度的运动估值。
- (2) 支持所有可变块的大小。

在编码情况下，不支持  $8 \times 4$ ， $4 \times 8$  和  $4 \times 4$  块大小。

- (3) 无限制的运动向量。
- (4) MPEG-4 交流/直流预测。
- (5) H. 264/AVC 的帧内预测。
- (6) 支持 H. 263 的附件 I，J，K(RS=0 和 ASO=0)和 T。

在编码情况下，不支持附件 I 和 K (RS=1 或 ASO=1)。

- (7) 错误修复工具

MPEG-4 重新同步。关于 RVLC 的标记和数据分割。

宏之间的位/宏的固定数量。

H. 264/AVC FMO 和 ASO

- (8) 位速率控制 (CBR&VBR)。
- (9) CIR(循环内部刷新)/AIR(适应内部刷新)。

## 3. 前/后旋转/镜像

- (1) 编码器中，8 旋转/镜像模式用于导入图像。
- (2) 解码器中，8 旋转/镜像模式用于输出图像。

## 4. 可编程

- (1) FIMV-MFC 1.0 版本嵌入 16 位 DSP 处理器，它是专注于处理位流和控制编解码器的元件。
- (2) 用于主机处理器和视频 IP 通信的通用寄存器和中断。

## 5. 性能

- (1) 向上全双工 VGA 30fps 编码/解码。
- (2) 向上半双工 720x480 30fps (720×576 25fps) 编码/解码。

## 6. 易集成性

- (1) 用于主机处理器通信的 AMBA，32 位 APB(w/ PREADY) 接口。
- (2) 用于外部存储器 AMBA，32 位 AXI 接口。

## 21.2 位处理器

本节描述位处理器，这是在各种不同格式下优化过程的位流，如 MPEG-4，H.263，H.264 和 VC-1。

该位处理器是一个嵌入式的可编程 16 位 DSP，它可以高度优化处理位流数据。除处理位流之外，位处理器通过主机接口控制视频编码解码器和主机处理器的通信。该处理器有 12KB 的程序存储器和 4KB 的数据存储器。

如图 21-3 所示，显示位处理器的结构框图。专用的寄存器包括指令，中断和代码下载寄存器。通用寄存器，64 个 32 位寄存器，可以用于主机处理器发送参数给视频编码解码器。如果应用程序需要超过 64 个寄存器，一个外部存储器可以用于扩展，因为位处理器能通过 AXI 总线接口存取外部存储器。

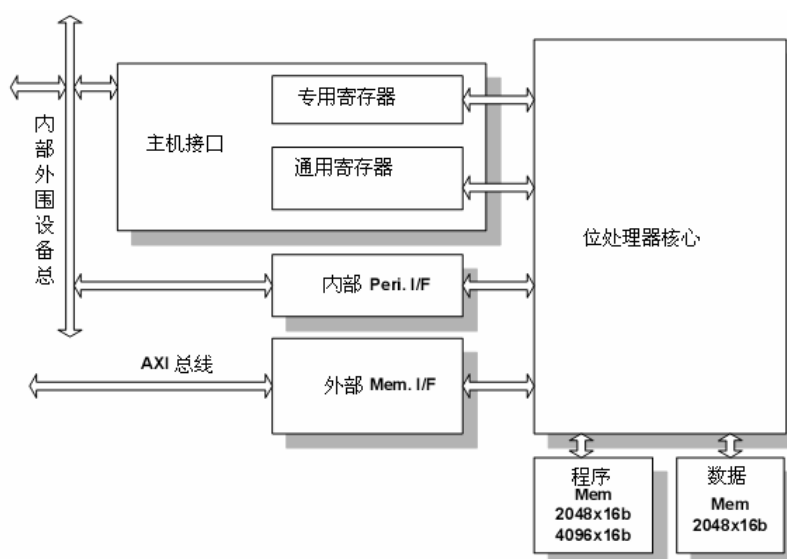


图 21-3 位处理器的结构框图

### 硬件加速

该位处理器核心嵌入硬件加速子模块如下。

- 加速器支持位流打包的指令，如 put\_bits。
- 加速器支持位流未打包的指令，如 get\_bits 和 show\_bits。

- 用于 VLC 和 VLD 操作的查表和搜索模块。
- 运动向量预测/重建模块。
- DMA 控制器从/到外部的 SDRAM 传输位流。

## 下载固件

驱动视频编解码模块的位固件和主机处理器接口被分成两部分。一部分用于通过 APB 总线，主机处理器下载的启动代码。启动代码大小是 1024 字节。另一部分用于编解码，如 MPEG-4，H.263，H.264 和 VC-1。用于下载固件的执行程序，在初始化步骤中只执行一次。

### 1. 启动代码

运行编解码器之前，主机处理器必须下载 BIT 启动代码到程序存储器中， $2048 \times 16$  同步单端口 SRAM，通过主机接口如下。

第 1 步:  $\text{addr} = 0$  ；

第 2 步:  $\text{code\_data} = (\text{地址} \ll 16) + \text{bit\_code}[\text{地址}]$ ；

第 3 步: 将  $\text{code\_data}$ （数据代码）写入到主机接口的 CodeDownLoad 寄存器，该接口嵌入到位处理器中；

第 4 步:  $\text{地址} = \text{地址} + 1$ ；

第 5 步: 如果（ $\text{地址} < 512$ ）到第 1 步，否则转到程序下载编解码器固件。

$\text{bit\_code}$  是一个数组，其大小是  $512 \times 16$  位。

注：上述  $\text{bit\_code}[0-511]$  必须被写入主机接口的 CodeBufAddr 寄存器中的指定区域。

### 2. 编解码器固件

除启动代码之外，操纵 IP 需要一个固件的封装。基本上，封装是用于提供 MPEG-4，H.263P3，H.264 和 VC-1 的编解码器。必须写固件到外部存储器的区域并发送关于区域的低级的地址，通过写入到 CodeBufAddr 寄存器。在运行时序时，固件部分自动加载到内部存储器中。如图 21-4 所示，显示位固化存储空间框图。

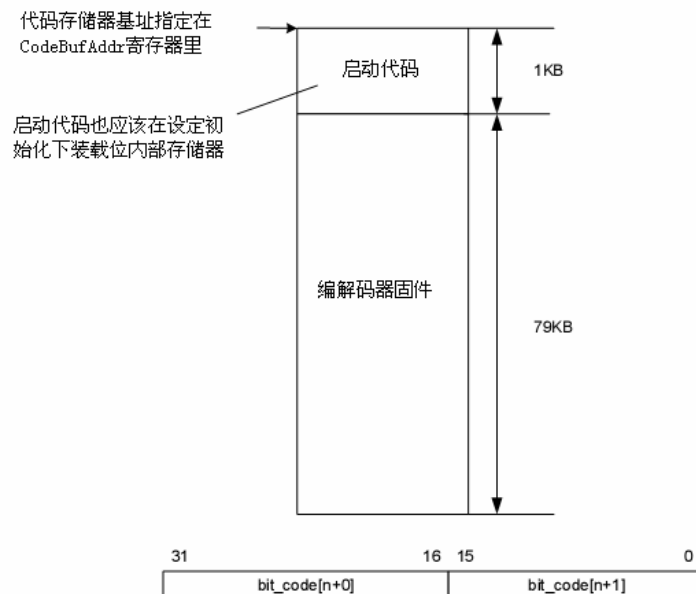


图 21-4 位固件的存储空间

以下程序是在视频编解码器中下载编解码器固件初始化的步骤：

第 1 步:  $addr = 512$  ；

$sdr\_addr = *CodeBufAddr + 1024$ ；（sdr 即动态随机存取存储器）

第 2 步:  $code\_data = (bit\_code[地址] \ll 16) + bit\_code[地址+1]$ ；（code\_data 代码数据）（bit\_code 位代码）

第 3 步:  $*sdr\_addr = code\_data$ ；

第 4 步: 地址=地址+ 2 ；

第 5 步:  $sdr\_addr = sdr\_addr + 4$ 。

第 6 步: 如果（地址 < （ 80 × 1024 ） ）回到第 1 步，否则完成下载。

### 3. 运行编解码器

该部分描述位处理器如何控制视频编解码器和通信主机处理器。

所提供的固件可以同时处理 8 个进程。每个进程可以有不同的格式，MPEG-4，H. 362P3，H. 264 或 VC-1 和编解码器进程中的编码或解码。举例来说，可能同时处理一个 MPEG-4 的编码进程，一个 H. 264 编码进程中，一个 H. 263 P3 解码进程和一个 VC-1 的解码进程。

编码图像和/或解码位流进程如下：

- 创建进程：可以创建和配置进程。

- 运行进程：在一个适当的时间场合，可以执行特定的进程。适当的时间场合意思是，当编解码器是空闲状态和在外部存储器中的图像编码或位流解码准备就绪。
- 结束进程：可以退出某个具体的进程。

如图 21-5 所示， 举例说明运行编解码器的固化状态。

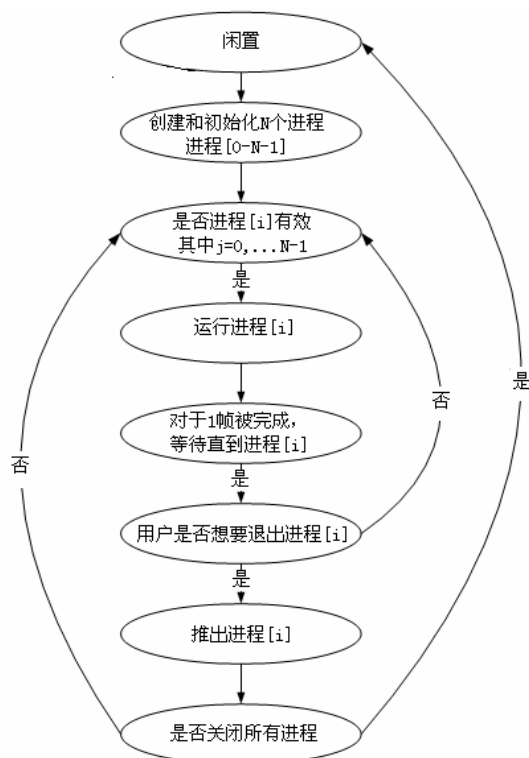


图 21-5 编解码器固件状态图

#### 4. 进程 ID-运行索引

每个进程的建立，以特定的 ID 命名，运行索引范围为 0~到 7。

基本上，ID 被分配基于创建的顺序。在创建过程初始化步骤后，主机处理器命令位处理器执行关于运行索引的指定进程。

如果进程创建顺序为 (a) MPEG-4 编码，(b) H.264 编码，(c) 适用于位流 A 的 H263P3 解码，(d) 适用于位流 B 的 VC-1 解码，分配 ID 如下。

MPEG-4 编码：RunIndex = 0。

H.264 编码：RunIndex = 1。



适用于位流 A 的 H. 263P3 解码: RunIndex = 2。

适用于位流 B 的 VC-1 解码: RunIndex = 3。

## 5. 分配编码格式 - RunCodStd

除进程 ID 外, RunCodStd 是用来在创建过程中定义编码标准和是否创建进程编码图像或解码位流。

RunCodStd = 0: MPEG-4/H. 263P3 编码。

RunCodStd = 1: MPEG-4/H. 263P3 编码。

RunCodStd = 2: H. 264 解码。

RunCodStd = 3: H. 264 编码。

RunCodStd = 4: VC-1 解码。

例如: 如果进程创建一个例子, RunIndex 和 RunCodStd 的每个进程列举如下:

(a) MPEG-4 编码进程: RunIndex = 0, RunCodStd = 0。

(b) H.264 编码进程: RunIndex = 1, RunCodStd = 3。

(c) H.263P3 解码进程: RunIndex = 2, RunCodStd = 1。

(d) VC-1 解码进程: RunIndex = 3, RunCodStd = 4。

以下为多格式视频编解码器中 Mpeg4, H264, H263 和 Vc1 的初始化代码。请大家参考:

```
void MFC_InitProcessForDecodingMpeg4(
    u32 uProcessIdx, u32 uStreamBufStAddr, u32 uStreamBufSize,
    u32 uFrameBufStAddr, bool bDecRotEn, bool bMp4DeblkEn)
{
    MFC_InitProcessForDecoding(uProcessIdx, MP4_DEC, uStreamBufStAddr, uStreamBufSize,
        uFrameBufStAddr, bDecRotEn, bMp4DeblkEn, false);
}

void MFC_InitProcessForDecodingH264(
    u32 uProcessIdx, u32 uStreamBufStAddr, u32 uStreamBufSize,
    u32 uFrameBufStAddr, bool bDecRotEn, bool bH264ReorderEn)
{
    MFC_InitProcessForDecoding(uProcessIdx, AVC_DEC, uStreamBufStAddr, uStreamBufSize,
        uFrameBufStAddr, bDecRotEn, false, bH264ReorderEn);
}
```

```

}

void MFC_InitProcessForDecodingH263(
    u32 uProcessIdx, u32 uStreamBufStAddr, u32 uStreamBufSize,
    u32 uFrameBufStAddr, bool bDecRotEn, bool bH263ReorderEn)
{
    MFC_InitProcessForDecoding(uProcessIdx, AVC_DEC, uStreamBufStAddr, uStreamBufSize,
        uFrameBufStAddr, bDecRotEn, false, bH263ReorderEn);
}

void MFC_InitProcessForDecodingVc1(
    u32 uProcessIdx, u32 uStreamBufStAddr, u32 uStreamBufSize, u32 uFrameBufStAddr, bool
bDecRotEn)
{
    MFC_InitProcessForDecoding(uProcessIdx, VC1_DEC, uStreamBufStAddr, uStreamBufSize,
        uFrameBufStAddr, bDecRotEn, false, false);
}

```

## 6. 编解码器状态

FIMV-MFC V1.0 提供繁忙标志寄存器和中断，该中断可以给出是否以指定的格式编码或解码一帧完成的信息。当 IP（低于）运作，BusyFlag 读取 ‘1’。如果编码或解码一帧被完成，同时 IP 是在等待状态，可以从主机处理器接受一个命令，BusyFlag 变为 ‘0’。IP 发生中断，在这个时间，BusyFlag 变为 ‘1’。请求下一个进程前，主机处理器必须清除中断。

如果在处理一帧过程中有一些错误，BusyFlag 和中断也有效。主机处理器可以知道，是否一个过程被正常地终止或不被正常地终止，是通过提交主机接口的其它状态寄存器决定。

## 7. 中断

### 1) 到一个主机处理器

位处理器，可以产生中断请求到主机处理器。基本上，这个中断是用来显示编码或解码一个帧的完成。中断信号 IREQ 高位有效，直到主机处理器通过将主机接口的中断清除寄存器写 “1” 来清除它。

IREQ 同步到 CCLK。

## 2) 来自一个主机处理器

通过写入 ‘1’ 到主机接口的 HostIntReq 寄存器，主机处理器可以请求一个中断到 IP。位处理器的确认后中断自动清除。

## 21.3 硬件视频编解码器

这个部分描述 FIMV-MFC 1.0 版本的硬件视频编解码器。除 VLC 和 VLD 的处理系数外，所有的视频编解码处理通过硬件执行。

### 1. H. 264 编码器的数据流

如图 21-6 所示，显示 FIMV-MFC 1.0 版本 H.264 编码过程的数据流。在编码时，帧间预测模块只载入相关帧的色度数据。从运动估值模块的本地存储器读取亮度数据，以便帧间预测的总线运载量被删除。当分割滤波器操作在 on-the-fly 模式里时，部分重建，但不过滤，在帧间预测模块中，像素数据被写入后面的外部存储器中使用。当分割滤波器在单机模式中操作，全部重建像素数据在宏模块中被写入外部存储器，如果宏模块被重建，则它被过滤。

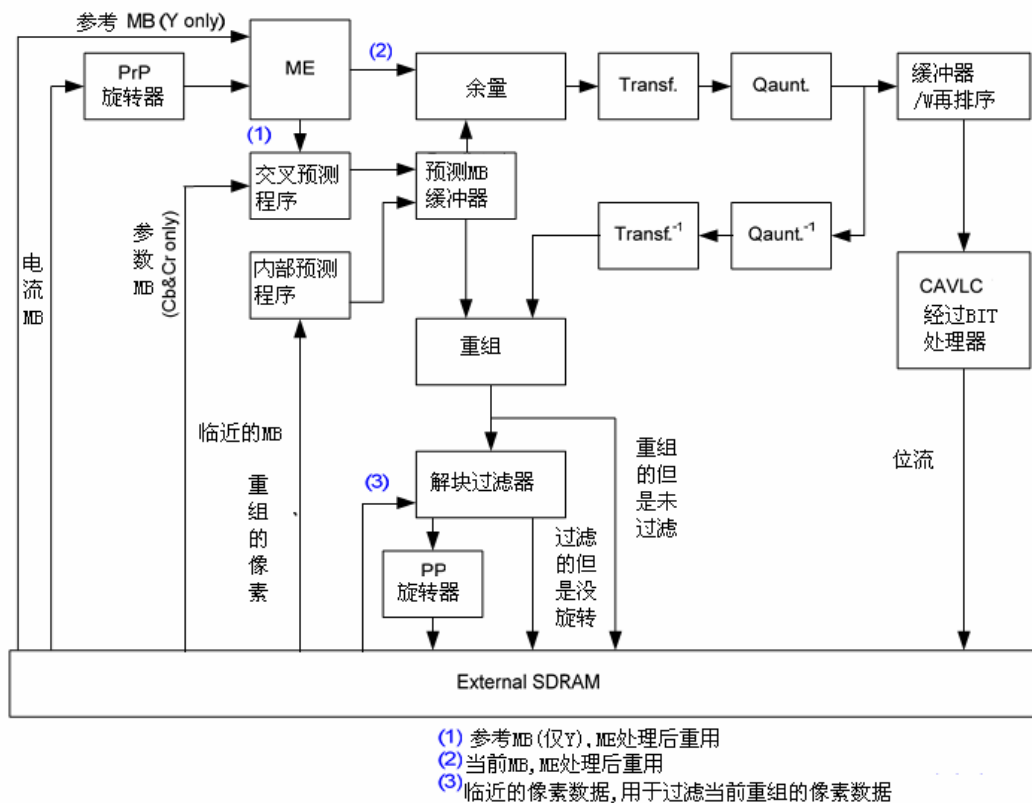


图 21-6 H.264 编码器的数据流

## 2. MPEG - 4 的编码数据流

如图 21-7 所示，显示 FIMV-MFC 1.0 版本 MPEG-4 编码过程的数据流。它的数据流非常相似于 H.264 编码。不相同的如下列出：

MPEG-4 的编码过程包括交流/直流预测代替内部预测 (intra-prediction)。

分割滤波脱离编码循环。

该总线载入量的位比 H.264 的少，因为它的 1/2 的像素采样所需的像素数据比 H.264 1/4 的少。

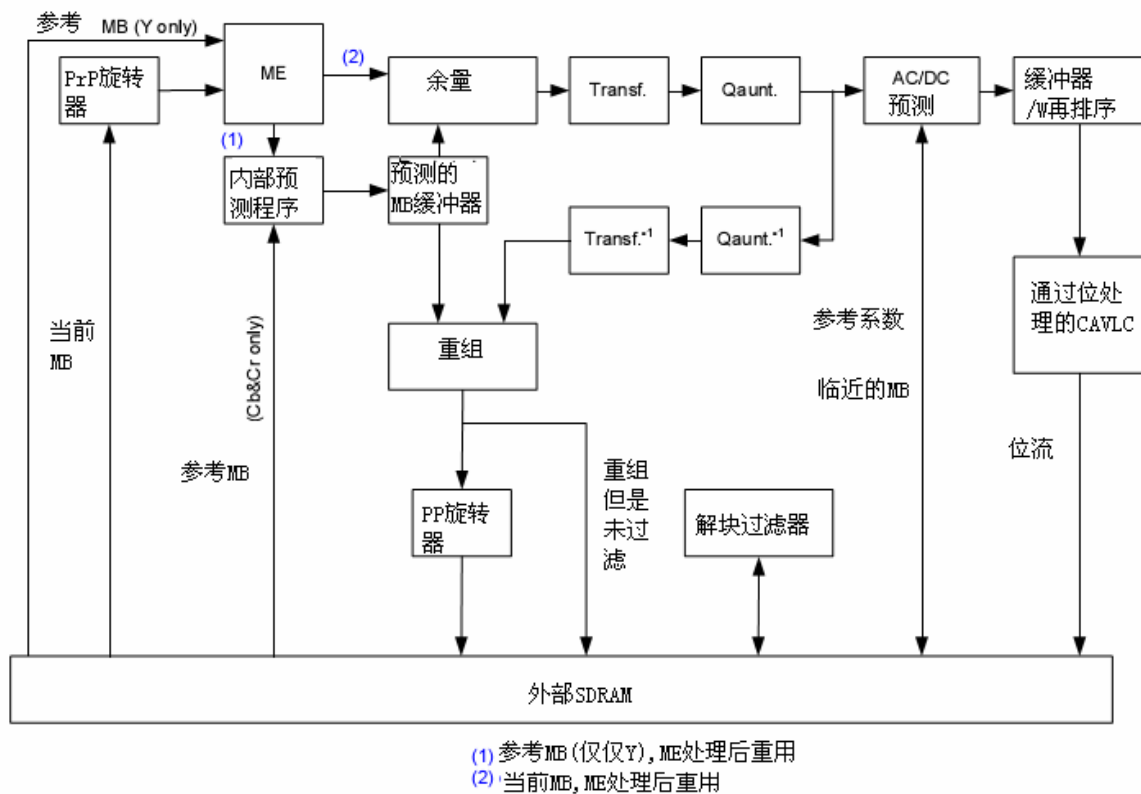


图 21-7 MPEG-4 编码器的数据流

### 3. H.263 编码器的数据流

H.263 编码器数据流和 MPEG-4 编码器几乎一样的，除以下说明：

当附件 I 启动，在量化前，嵌入交流/直流预测过程。

注：FIMV-MFC1.0 版本在编码过程中不支持附件 I。

当附件 J 启动，分割滤波运作在编码循环内部。

编码器总线加载，如表 21-1 所示。

表 21-1 编码器总线加载

项目	单位	GIF	VGA	D1
NO. 字节/像素行	Byte/pixel row	352	640	720
NO. 像素行/图片	Pixel row/picture	288	480	480
无字节/亮度	Byte/luminance	101376	307200	345600
无字节/色度	Byte/chrominance	50688	153600	172800

无字节/帧	Byte/frame	152064	460800	518400
亮度数据为 30 帧	MB/sec	3.041	9.216	10.368
色度数据为 30 帧	MB/sec	1.521	4.608	5.184
帧数据为 30 帧	MB/sec	4.562	13.824	15.552

用于 30fps 的编码器带宽的要求，如表 21-2 所示。

表 21-2 用于 30fps 的编码器带宽的要求

用于 30fps 的编码器带宽的要求				
项目	单位	GIF	VGA	D1
读取要编码的中断帧数据，用于运动估值和处理剩余的数据	MB/s	4.562	13.824	15.552
读取参考亮度数据，用于运动估值/补偿	MB/s	3.041	9.216	10.368
读取参考色度数据，用于运动补偿	MB/s	3.041	9.216	10.368
写入重建的帧	MB/s	4.562	13.824	15.552
读/写像素行，用于分割滤波	MB/s	3.041	9.216	10.368
读/写系数，用于交流/直流预测（MPEG-4 的情况下）	MB/s	3.041	9.216	10.368
位流封装	MB/s	0.375	1.500	1.500
读/写重建像素，用于帧内预测（H.264 的情况下）	MB/s	0.4	1.3	1.5
MPEG-4 编码	MB/s	21.663	66.012	74.976
*当分割滤波器启动				
H.264 编码	MB/s	19.022	58.096	65.208

在编码情况下，H.264 的总线加载要求类似 MPEG-4 的要求。这是因为 FIMV-MFC V1.0 在 H.264 编码时只用一个相关 FRAME，并且编解码模块重新使用运动估值模块的本地存储器中相关数据，而不是从外部存储器重新载入。因为通信量在搜索 SRAM 和代码模块之间，H.264 的搜索 SRAM 的接入率比支持不同模块 MPEG-4 的大小要高。

#### 4. H.264 解码器的数据流

关于 H.264 解码过程，解码在编码回路里的数据分支被重新使用，除以下内容：

位处理解码位流，并且加码的系数存储在系数缓冲器中。

关于参考像素数据，帧间预测模块读 Y 和 Cb/Cr 数据，组成外部存储器。

5. MPEG-4 解码器的数据流

MPEG-4 解码进程的数据流是和 MPEG-4 编码循环中解码的一样的，除了来自位处理器的引入系数和来自外部 SDRAM 的 Y 参考帧，不是来自运动估值模块的内部存储器。

6. H. 263 解码器的数据流

H.263 解码进程的数据流是和 H.263 编码循环中解码的一样的，除了来自位处理器的引入系数和来自外部 SDRAM 的 Y 参考帧，不是来自运动估值模块的内部存储器。】

7. VC-1 解码器的数据流

VC-1 解码进程的数据流是和 H.264 解码循环中解码的一样的，除了使用定标器。如果标头是 MULTIREES 标志，解码的图片被定标器放大。

解码器总线加载，如表 21-3 所示。

表 21-3 解码器总线加载

项目	单位	GIF	VGA	D1
N0. 字节/像素行	字节/像素行	352	640	720
N0. 像素行/图片	像素行/图片	288	480	480
无字节/亮度	字节/亮度	101376	307200	345600
无字节/色度	字节/色度	50688	153600	172800
无字节/帧	字节/帧	152064	460800	518400
亮度数据为 30 帧	MB/s	3. 041	9. 216	10. 368
色度数据位 30	MB/s	1. 521	4. 608	5. 184
帧数据为 30 帧	MB/s	4. 562	13. 824	15. 552

用于 30fps 的解码器带宽的要求，如表 21-4 所示。

表 21-4 用于 30fps 的解码器带宽的要求

用于 30fps 的解码器带宽的要求				
项目	单 位	GIF	VGA	D1

读取参考的数据，用于 16×16 块大小（ MPEG - 4 的情况下） *假设所有的宏有 16×16 块大小。				MB/ s	6.05 9	18.36 0	20.65 5
读取参考的数据，用于 8×8 块大小（ MPEG - 4 的情况下） *假定所有宏有 8×8 块大小。				MB/ s	7.69 8	23.32 8	26.24 4
读取参考数据，用于 16×16 块大小（H. 264 的情况下） *假设所有的宏有 16×16 块大小。				MB/ s	8.98 1	27.21 6	30.61 8
在典型情况下读取参考数据 （H. 264 的情况下） *实验的基础上				MB/ s	15.8 95	48.16 8	54.18 9
读参数数据，用于 4×4 块的大小 （H. 264 的情况下） *假定所有宏块有 4×4 块的大小				MB/ s	30.7 93	93.31 2	104.9 76
写重建帧				MB/ s	4.56 2	13.82 4	15.55 2
读/写像素行，用于分割滤波				MB/ s	3.04 1	9.216	10.36 8
读/写系数，用于交流/直流预测（ MPEG-4 的情况下）				MB/ s	3.04 1	9.216	10.36 8
位流加载				MB/ s	0.12 5	0.500	0.500
VC-1 解码器	单位	GIF	VGA	MB/ s	13.7 20	41.90 0	47.07 5
X		352	640				
Y		288	480				
亮度（BW）	MB/s	3.041	9.216				
色度（BW）	MB/s	1.521	4.608				
总共（BW）	MB/s	4.562	13.824				
宏/帧		396	1200				
参读 8×8（VC-1）最差情况	MB/s	10.288	31.176				



参读 16×16 (VC-1)	MB/s	7.342	22.248	25.029		
分割滤波器	MB/s	3.041	9.216	10.368		
重叠滤波器	MB/s	3.041	9.216	10.368		
ACDC 预估器	MB/s	3.041	9.216	10.368		
回复以供显示	MB/s	4.562	13.824	15.552		
回复以供参考	MB/s	4.562	13.824	15.552		
位加载	MB/s	0.125	0.500	0.500		
VC-1 的解码（最坏的情况下，所有参考 8×8 ）	MB/s	28.661	86.972	97.781		
MPEG-4 16×16 解码						
MPEG-4 8×8 解码	MB/s	15.426	46.868	52.664		
H. 264 16×16 解码	MB/s	16.709	50.756	57.038		
H. 264 典型解码	MB/s	23.624	71.708	80.609		
H. 264 4×4（最大值）	MB/s	38.521	116.852	131.396		

其 VC-1 解码器，如表 21-5 所示。

表 21-5 VC-1 解码器

VC-1 解码器	单位	GIF	VGA	D1
X		352	640	720
Y		288	480	480
亮度 (BW)	MB/s	3.041	9.216	10.368
色度 (BW)	MB/s	1.521	4.608	5.184
总共 (BW)	MB/s	4.562	13.824	15.552
宏/帧		396	1200	1350

参读 8×8（VC-1）最差情况	MB/s	10.288	31.176	35.073
参读 16×16（VC-1）	MB/s	7.342	22.248	25.029
分割滤波器	MB/s	3.041	9.216	10.368
重叠滤波器	MB/s	3.041	9.216	10.368
ACDC 预估器	MB/s	3.041	9.216	10.368
回复以供显示	MB/s	4.562	13.824	15.552
回复以供参考	MB/s	4.562	13.824	15.552
位加载	MB/s	0.125	0.500	0.500
VC-1 的解码（最坏的情况下，所有参考 8×8）	MB/s	28.661	86.972	97.781

8. 全双工通信编解码器处理

在编码循环来降低逻辑区域过程中，FIMV-MFC V1.0 编码数据流重新使用解码数据路径。为全双工通信编解码器的应用，设置时间段。如图 21-8 所示。



图 21-8 全双工通信编解码器进程

9. 全双工通信编解码器总线负载

全双工通信编解码器总线负载和编码和解码总线负载的总数一样。

21.4 帧缓冲器

该部分叙述用于 FIMV-MFC 1.0 版本的视频编解码器模块中的帧缓冲器的内存映射。如图 21-9 所示，显示帧缓冲器的配置。

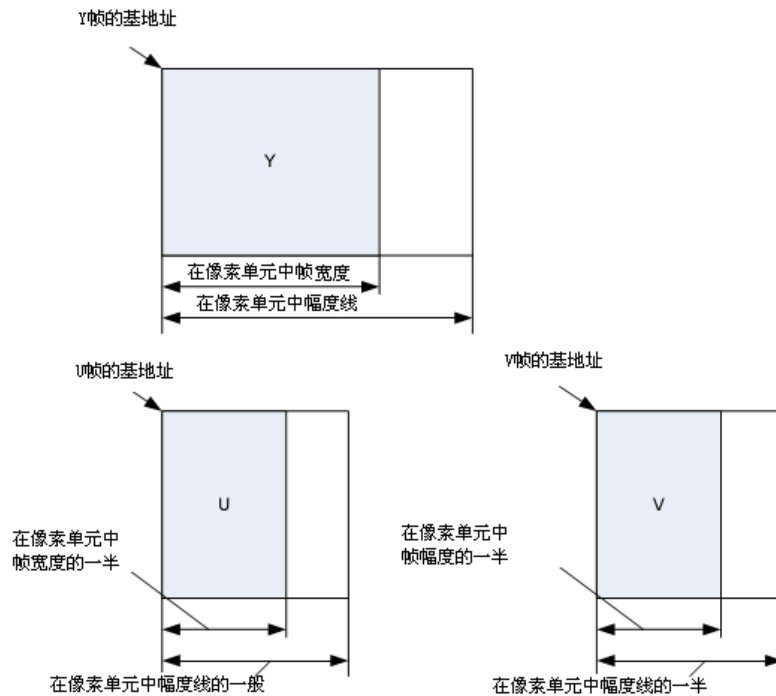


图 21-9 帧缓冲器配置

详有基地址和幅度线的帧缓冲器。一个完整的图像由 Y，U，和 V 部分组成。因此一个图像需要 3 个帧缓冲器，由 Y，U，和 V 组成。幅度线的意思是在像素单位中亮度的宽度组成缓冲器并必须是 8 的倍数。U 和 V 帧缓冲器的幅度线是 Y 帧缓冲器的一半，并可自动在 Y 帧缓冲器的幅度线上提取。FIMV-MFC V1.0 支持 11 位的幅度线。

如图 21-10 所示，显示帧缓冲器的内存映射。对于 V 帧缓冲器，除基地址以外，内存映射与 U 帧缓冲器是一样的。

FIMV-MFC V1.0 支持小端和大端系统。它意味着在图 21-10 中的 Y (0, 0) 设在位[31: 24]中。用户可以指定端到主机接口的寄存器。

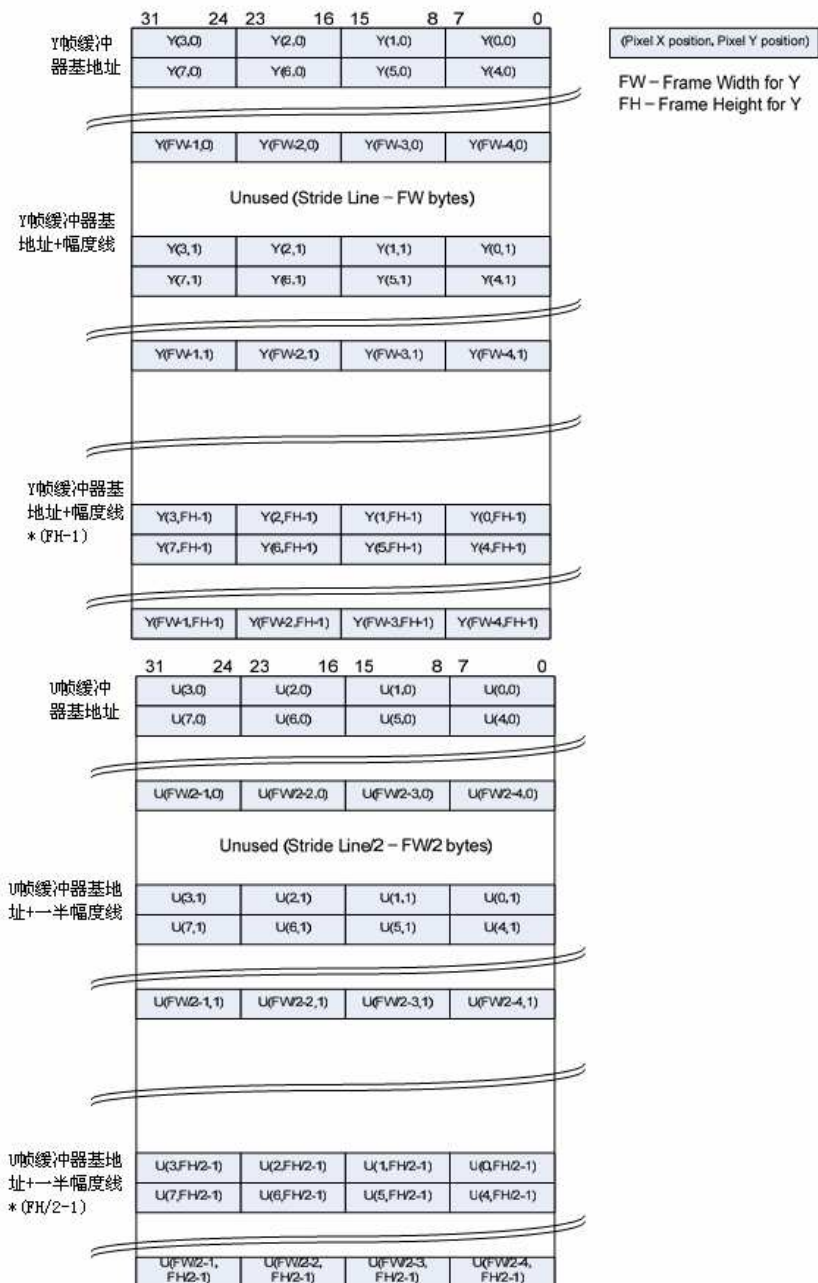


图 21-10 在小端中的帧缓冲器地址图

关于编码通常情况下，4 个帧缓冲器是必需的。这些缓冲器是用于存储从照相机或预处理器导入的图像，编码、存储当前重构的图像和先前重构的帧。

## 21.5 旋转/镜像

FIMV-MFC V1.0 支持旋转和镜像功能，用于显示编码的引入图像和解码的输出图像。PrP 旋转器模块完成前者的功能，PP 旋转器模块完成后者的功能。

### 1. PrP 旋转器模块

FIMV-MFC V1.0 使用以旋转器模块的输出作为输入，在外部 SDRAM 上，编码器没有额外的带宽消耗，其主要用于旋转器自身。旋转图像被发送到运动估值模块的本地缓冲器上，并重新使用为内部模式下的帧内预测的模块和剩余估计模块。PrP 旋转器不能工作在解码进程中。如图 21-11 所示，显示 PrP 旋转器数据流。

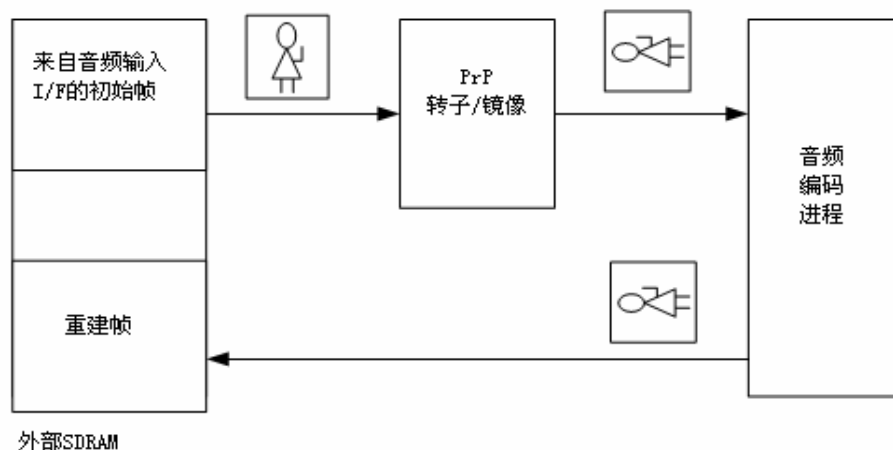


图 21-11 PrP 旋转器数据流

### 2. PP 旋转器模块

在解码过程中旋转/镜像处理需要附加带宽，因为视频编解码器的 IP 已为下一个图像重新使用未旋转图像。因此旋转图像被写入其它的内存空间。在这个设计下，显示 I/F 没有改变内存空间是为显示解码图像，因为后来的旋转图像被写入同一个空间。当然，目标帧缓冲器的改变尽可能通过设置寄存器来指定它的基地址。PP 旋转器模块不能工作在编码过程中。如图 21-12 所示，显示 PP 旋转器数据流。

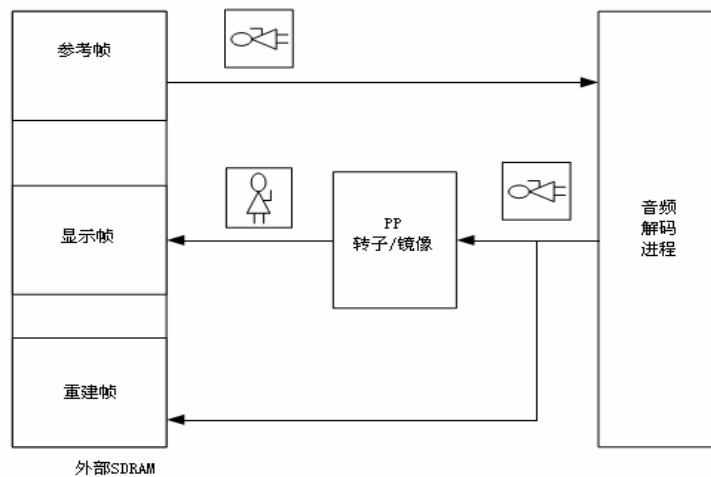




图 21-12 PP 旋转器数据流

### 3. 旋转/镜像模式

旋转器模块同时支持  $90 \times n$  次数 ( $n=0, 1, 2, 3$ ) 旋转的 8 类型模式和镜像。下面是支持旋转/镜像的列表和这些代表所有可能旋转和镜像的组合。有两个寄存器设置。一个为 **PrP** 模块，另一个为 **PP** 模块。

表 21-6 旋转/镜像模式

模式	旋转/镜像图像	描述
NONE_ROTATE		原始图像（没有旋转/镜像）， 例如图片大小：720×480。
ROT_LEFT_90		向左旋转 90（向右旋转 270）， 例如图片大小：480×720
ROT_LEF_180		向左旋转 180（向右旋转 180）， 例如图片大小：720×480
ROT_LEFT_270		向左旋转 270（向右旋转 90）， 例如图片大小：480×720
MIR_HORIZ		横向镜像 例如图片大小：720 × 480
MIR_VERT		垂直镜像 例如图片大小：720 × 480

MIR_HOR_ROT_RIGHT_90		横向镜像和向右旋转 90 例如图片大小：480×720
MIR_HOR_ROT_LEFT_90		横向镜像和向左旋转 90 例如图片大小：480×720

如表 21-5 所示，显示旋转/镜像模式。在上面例子中，输入所有的旋转/镜像模式是与 NONE\_ROTATE 模式的输出图像一样的。主机处理器可以通过设置专用寄存器选择这些模式中的一个。实际上，是命名一个 API 来配置旋转模式和发送信息到位处理器中。位处理器设置一个旋转/镜像模块的寄存器来指定模式。

## 21.6 运动估值

运动估值模块使用全搜索运算法则，搜索范围是+/- 16 像素或+/- 8 像素。如图 21-13 所示，显示运动估值模块框图。

它支持以下功能：

- UMV（无限制的运动向量）模式。
- 为 H. 264-BP，高达四分之一搜索。
- 为 MPEG-SP，高达一半搜索。
- 为 MPEG4-SP，支持 16×16/8×8 块。
- 为 H. 264-BP，支持 16×16/16×8/8×16/8×8 块。
- 全搜索运算法则。

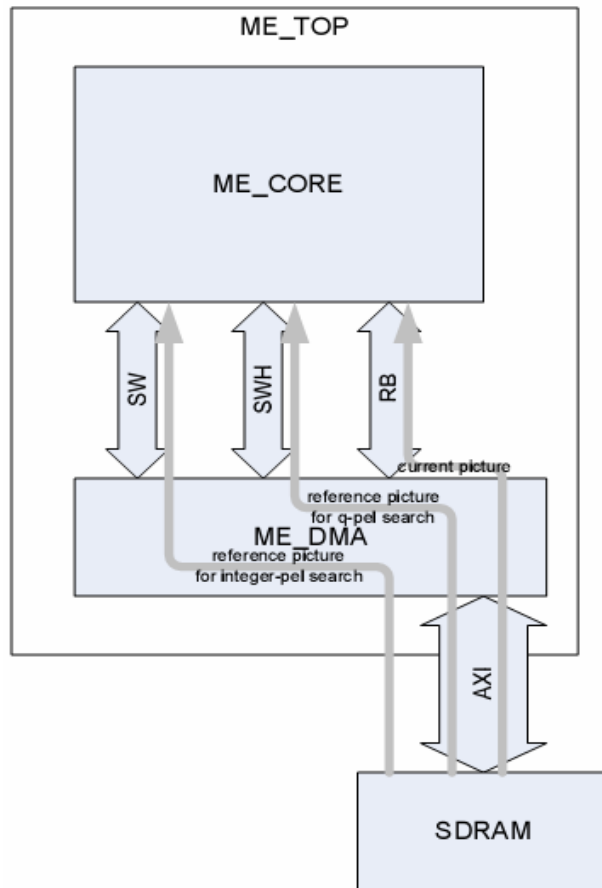


图 21-13 运动估值模块框图

ME 块能利用高优先级在零运动向量上通过减去适当的 SAD 和自定义寄存器值。预测块大小 ( $16 \times 16$ ,  $16 \times 8$ ,  $8 \times 16$ ,  $8 \times 8$ ) 可以以相同的方式划分。有 3 个偏移量寄存器分别为  $8 \times 8$  块,  $8 \times 16$  块及  $16 \times 8$  块。可以利用高优先级在大块的大小上通过加上适当 SAD 和自定义寄存器。

运动估值块读取当前帧和参考帧。ME\_DMA 处理所有存储器操作。ME\_CORE 请求 ME\_DMA 为当前帧图像和参考帧图像, ME\_DMA 读数据和发送数据到 ME\_CORE。如图 21-14 所示, 显示运动估值核心框图。



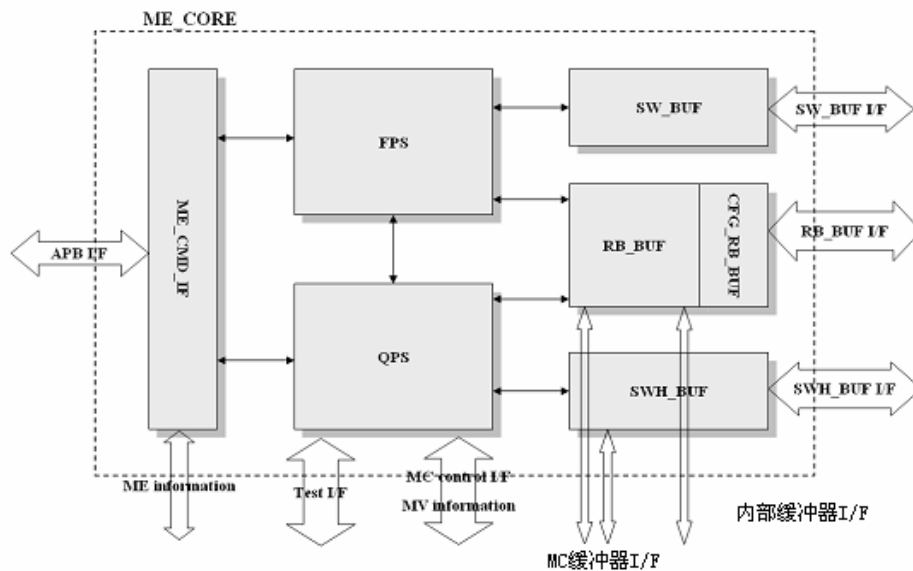


图 21-14 运动估值的核心块框图

ME\_CORE 由命令接口 (ME\_CMD\_IF)，整数像素搜索块 (FPS)，四分之一/一半像素搜索块 (QPS) 和内部缓冲器 (SW\_BUF, RB\_BUF, SWH\_BUF) 组成。

ME\_CMD\_IF 块接口有位处理器或 CPU。当前帧图像储存在 RB\_BUF 中，参考帧图像储存在 SW\_BUF 和 SWH\_BUF 中。整数像素搜索块 (FPS) 用 RB\_BUF 为当前帧和用 SW\_BUF 为参考帧。四分之一/一半像素搜索块 (QPS) 用 RB\_BUF 为当前帧，用 SWH\_BUF 为参考帧。如图 21-15 所示，显示运动估值的 DMA 模块框图。

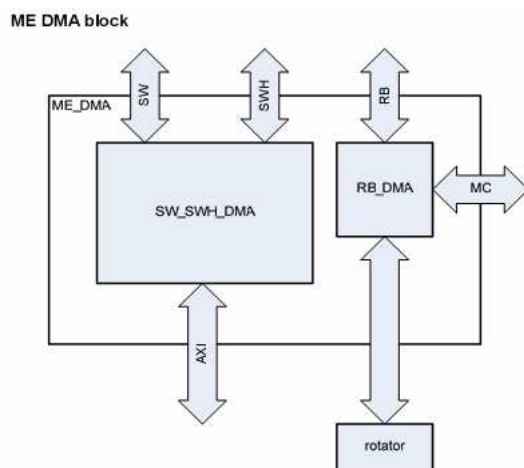


图 21-15 运动估值的 DMA 模块框图

ME\_DMA 服务当前和参考帧图像到 ME\_CORE。

SW\_DMA 和 SWH\_DMA 参考搜索窗口数据请求通道。SW\_DMA 和 SWH\_DMA 读取数据来自 SDRAM。RB\_DMA 通过像素旋转块读取当前帧数据。

## 21.7 帧间预测（Inter-Prediction）

帧间预测用于重建运动向量，代表解码当前块和参考帧相应位置之前的位移，来计算插值像素数据位运动补偿。

帧间预测由主控制器，插入器，DMA 和本地内存组成。每个子块的任务，如表 21-7 所示。

表 21-7 每个子块的任务

主控制器	位处理器约束它写入控制信息（运动向量，块模式，参考图片指数，图片的大小和运行命令等）进入主控制器中的控制寄存器。它包括运动向量的数据寄存器和在 sprog64x32 中参考图片基址。
插入器	它插入关于 2-pe1/4-pe1 分辨率的参考图片数据。它使用当时的存储器（sprog64x32e16）为计算 H. 264 四分之一像素。
DMA	从 SDRAM（解码）或 ME 本地存储器（编码）读取参考图片数据并写它入本地缓冲存储器（dprog60x96e8, dprog36x96e8）

帧间预测支持所有 H. 264 块模式（16×16，16×8，8×16，8×8，8×4，4×8，4×4），四分之一/一半像素分辨率，参考帧 16 在 H. 264BP 和填充像素中，当运动向量指向一个区域是一个外面的图像。

如图 21-16 所示，显示帧间预测的模块框图。

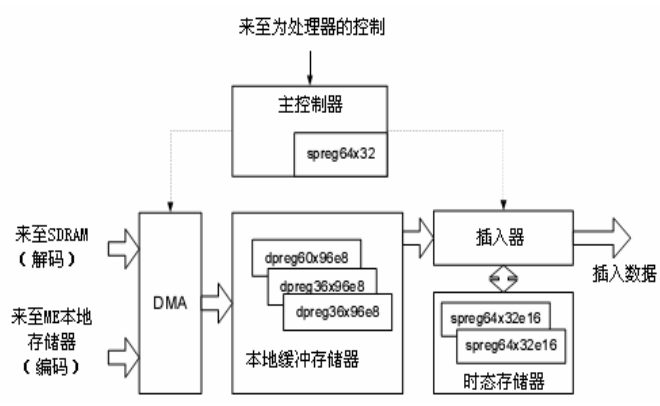


图 21-16 帧间预测框图

当位处理器设置主要控制器的寄存器（运动向量，参考帧指数，宏模式等，主控制器控制 DMA 和插入器。

DMA 从 SDRAM 解码中和 ME 本地存储器编码中读取参考像素数据来减少 SDRAM 的带宽。如果运动向量用于在参考帧中定位预测模块，包括像素位置，在解码模式下这些像素位置超出参考帧的边界。在这些情况下，超出限度的像素值是被 DMA 复制的边缘像素值。

如图 21-17 所示，显示本地缓冲内存配置。

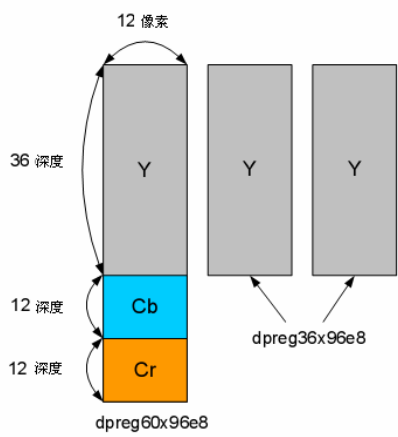


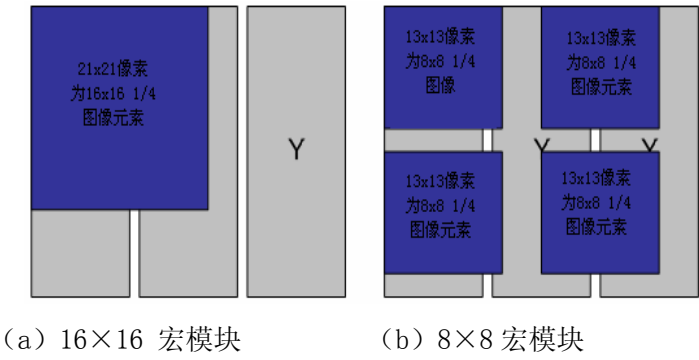
图 21-17 本地缓冲存储器配置

帧间预测的本地缓冲存储器由一个 dpreg60x96e8 和两个 dpreg36x96e8 组成。一些亮度的数据和所有的色度数据存储在一个 dpreg60x96e8 中，其他亮度数据存储在一个 dpreg36x96e8 中。

本地缓冲存储器包含一个宏像素数据在所有块模式中。

当宏块模式是 4×4 块模式时，帧间预测的 DMA 从 SDRAM（解码）读 16 子块数据并每个子块是 9×9。本地缓冲存储器的配置是 36 像素（9 像素×4 个子块）在横向和纵向中。

如图 21-18 所示，描述通过 DMA 存储参考像素数据在本地缓冲存储器中。



(a) 16×16 宏模块

(b) 8×8 宏模块

图 21-18 在  $16 \times 16$  和  $8 \times 8$  模块中存储参考像素

帧间预测的插入器模块计算一半或四分之一像素通过 DMA 使用在本地缓冲存储器中的参考像素数据。

插入器通过重建运动向量使用时态存储器插入像素在  $(1/4, 1/4)$ ,  $(1/4, 1/2)$ ,  $(1/2, 1/4)$  指示位置。插入像素的结果被写入缓冲器中来添加剩余的误差和当前像素。如图 21-19 和图 21-20 所示，显示编码情况下的帧内预测管线和编码情况下的帧间预测管线。

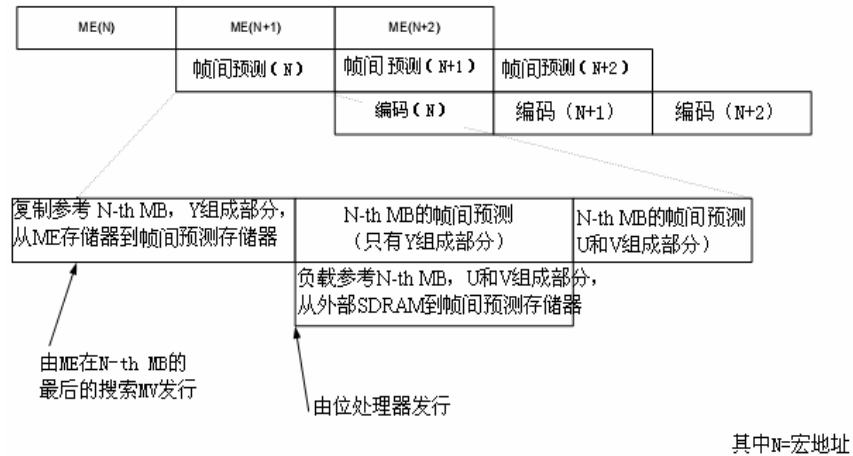


图 21-19 编码情况下的帧内预测管线

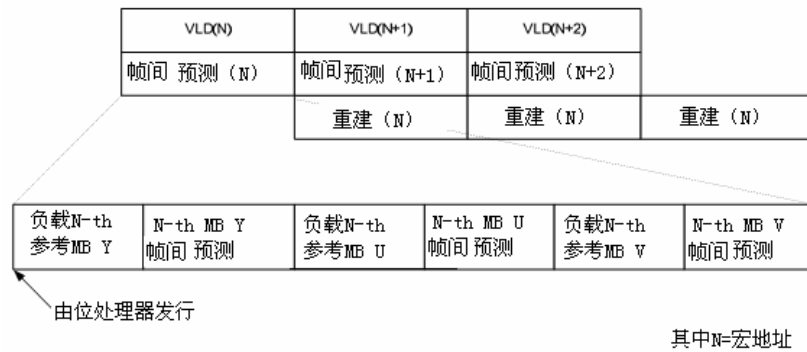


图 21-20 编码情况下的帧间预测管线

帧间预测支持以下特点：

### 1. MPEG-4/H. 263P3

- (1) UMC（无限制运动补偿）。
- (2) 4MV（支持  $16 \times 16$  和  $8 \times 8$  块大小）。
- (3)  $1/2$  像素块补偿。

## 2. H. 264

- (1)  $16 \times 16$ ,  $16 \times 8$ ,  $8 \times 16$ ,  $8 \times 8$ ,  $8 \times 4$ ,  $4 \times 8$  和  $4 \times 4$ 。
- (2) 16 参考帧。
- (3)  $1/4$  像素块补偿。

## 21.8 帧内预测 (Intra-Prediction)

FIMV-MFC V1.0 包括一个个帧内预测模块。一个是为 MPEG-4/H. 263P3 AC/DC 预测，另一个为 H. 264 帧内预测。

数字 6. x 显示在 MPEG 和 H. 264 中的帧内预测的数据流。在 H. 263P3 AIC(先进帧内编码)模式下, AC/DC 预测是以交换系数执行, 而不是以量化系数执行。如图 21-21 所示, (a) 显示 MPEG - 4 的交流/直流预测数据流和 (b) H. 264 帧内预测数据流

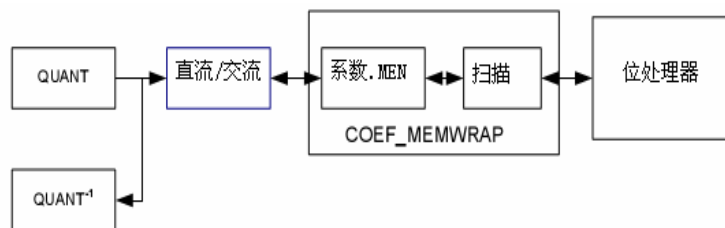


图 21-21 (a) MPEG - 4 的交流/直流预测数据流

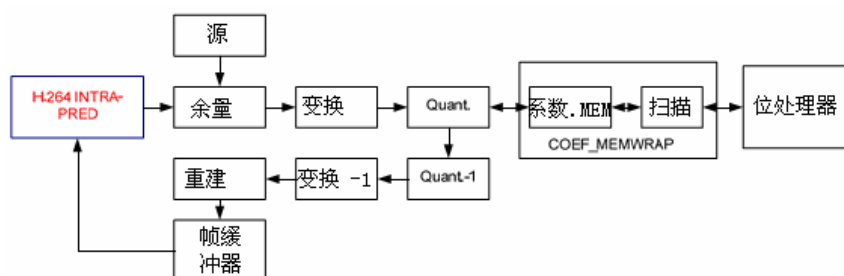


图 21-21 (b) H. 264 帧内预测数据流

大部份模块共享在双方的编码和解码中。这个模块通过带有 AMBA AXI 总线的 DMA 控制器接口来加载和存储邻近像素或系数。

内部缓冲器, dpsram80x32, 在交流/直流预测和帧内预测模块之间共享。它有个双端口, 因为预测模块在核心时钟里的操作和总线时钟的总线接口, 它们不同步的。如图 21-22 所示, 显示帧内预测块框图。

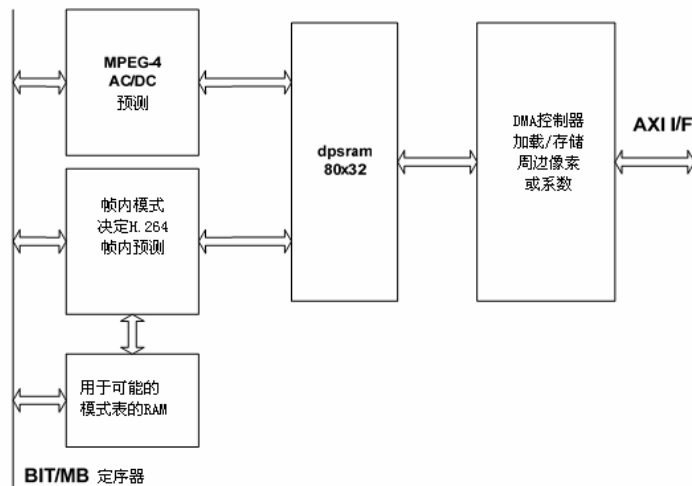


图 21-22 帧内预测块框图

在 MPEG-4 编码情况下，使用固定的预测模式决定。它带来高性能和低功耗。基于检测预测模式之上，编码和解码的系数数据自动重新排序。

对于编码中的 H. 264 帧内预测模式，使用固定模式决定或基于软件模式决定。在固定模式决定情况下，位处理器下载一个可能模式表。这个表为帧内预测模式决定逻辑使用来搜索一些候选模式中的最佳一项。候选数量是可配置的。由于编码的最高质量，所有可能的预测模式可估计。

位处理器管理控制邻近的宏或块的有效性。当前宏（或块）的切片，宏类型和位置决定它的有效性。如图 21-23 所示，显示在邻近像素缓冲器中部分像素存储。

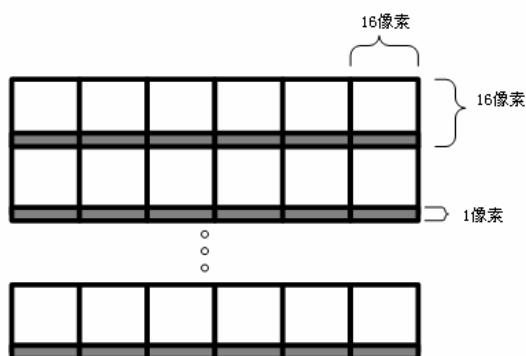


图 21-23 在邻近像素缓冲器中部分像素存储

当 MPEG-4/H. 263P3 启动时，对于每个宏，DMA 控制器存储或加载 32 个系数。所需存储器空间是  $(\text{number\_of\_macroblock\_in\_width\_of\_picture} \times 32)$  字。

在 H. 264 情况下，在外部存储器上所需存储器大小和带宽不同于依据编码选项。如果分割滤波器运作在“独立”的方式，DMA 控制器只能从重建帧缓冲器中加载邻近的像素。因此一个附加的存储器不需要为帧内预测。然而，在“on-the-fly”分割滤波情况中， $(32 \times \text{number\_of\_macroblock\_in\_picture})$  字节是必需的，因为重建帧不会存储在一个外部存储器中。在这种情况下，只有过滤帧被写入外部存储器中。因此，重建像素数据的第 16 行，必须存储为宏下面的帧内预测。外部主机处理器通过主机接口的寄存器赋值给缓冲器。

## 21.9 变换/量化

FIMV-MFC V1.0 有两个变换/量化模块。一个是为 MPEG-4/H. 263P3，另一个是为 H. 264。

FIMV-MFC V1.0 H. 264 T/Q 块处理余量数据的变换和量化（或逆变换和逆量化）。T/Q 块压缩余量数据并发送到其它块（系数缓冲器，运动补偿块）。

### 1. 概述

下列图片显示 FIMV-MFC V1.0 H. 264 T/Q 的编码和解码流。chroma\_dc 块共享在编码和解码过程中。Trans 和 itrans 块不仅处理余量变换而且处理  $4 \times 4$  亮度直流变换。在编码情况下，量化系数被写入系数缓冲器并位处理器读取它们，在系数缓冲器接口模块中被重新排序。在相同的时间，量化系数被处理在编码过程的解码循环中。

对于解码处理器，T/Q 模块读取系数被解码通过为处理器和在系数缓冲器接口中重新排序。如图 21-24 所示，显示 H. 264 的变换/量化的数据流。

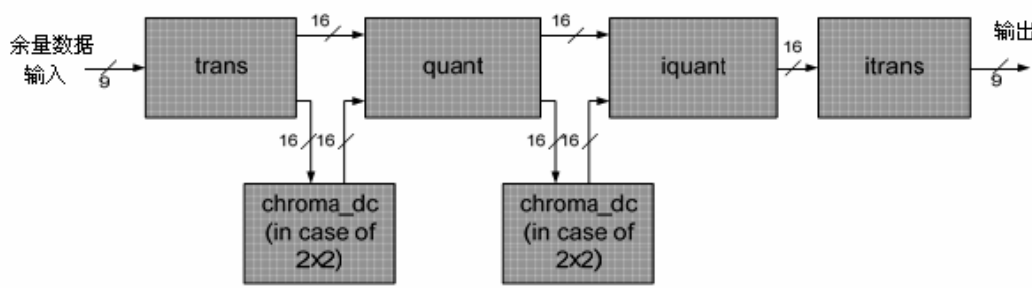


图 21-24 H. 264 的变换/量化的数据流

### 2. 方框图

如图 21-25 所示，显示 FIMV-MFC V1.0 H. 264 T/Q 块的方框图。

TRANS 块通过 TQ\_CTRL 块发送变换系数数据到 QUANT 块。TRANS 块可以剩余的  $4 \times 4$  块的转换和阿达玛

变换（在亮度情况下）。在色度的阿达玛变换情况下，CHROMA\_DC 块处理色度 DC 变换。CHROMA\_DC 处理正向的和反向的色度 DC 变换。

QUANT 块传输系数数据的量化结果到系数存储器或 IQUNT 块。

TQ\_CTRL 控制 H.264 T/Q 块的编码和解码的处理。

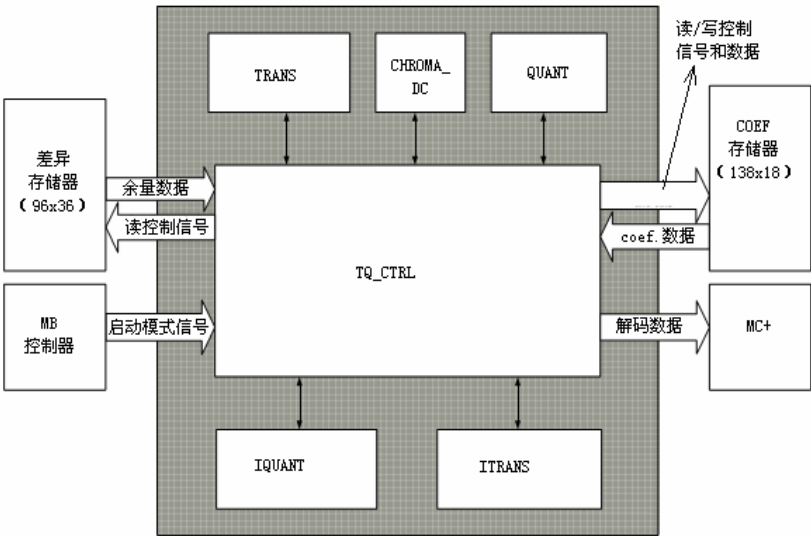


图 21-25 H.264 变换/量化方框图

3. MPEG-4/H.263

MPEG-4/H.263P3 变量/量化模式只支持方法 1 量化模式，用于 MPEG-1 位流。它可以为 H.263P3 位流处理 AIC（先进内部解码）和改进量化模式。在 MPEG-4/H.263P3 处理一个宏，FIMV-MFC V1.0 变换/量化需要 500 个周期。

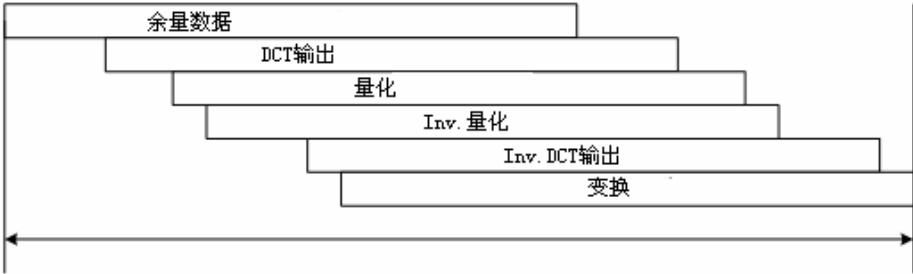


图 21-26 显示在宏通道中的 MPEG-4/H.263 变换/量化处理

如图 21-26 所示，显示在宏通道中的 MPEG-4/H.263 变换/量化处理。在编码过程中，在同一时间内量



化系数被发送到逆量化模块和系数缓冲器接口。在系数缓冲器中的系数再次被处理到 AC/DC 预测模块中。

注：对于 H.263P3 解码，支持 H.263 附件 I（先进帧内编码模式）。在编码过程中，不支持它。

#### 4. H.264

有很多宏类型在标准 H.264 中，如 INTRA\_4X4，INTRA\_16X16，各种不同大小的帧间宏和 I\_PCM。I\_PCM 宏在它的像素数据上没有变换和量化。因此，在 I\_PCM 宏的情况下，T/Q 模块通过处理引入数据到系数缓冲存储器而没有处理的逆量化模块中。在帧间宏和 INTRA\_16X16 宏的情况下，操作非常相似。为 H.264 编解码器，最复杂的宏类型执行和控制的观点的是 INTRA\_4X4。除 INTRA\_4X4 外，所有处理如变换和量化可能被有效的传递。不过，在编码 INTRA\_4X4 类型宏的情况下，变换当前  $4 \times 4$  块前，已完成重建先前  $4 \times 4$  块。这是因为先前重建  $4 \times 4$  块被用于当前  $4 \times 4$  块的帧内预测，输入余量估计模块。如图 21-27 所示。显示 H.264 编码管线。

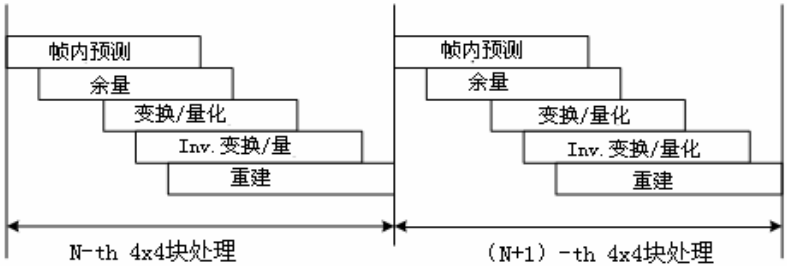
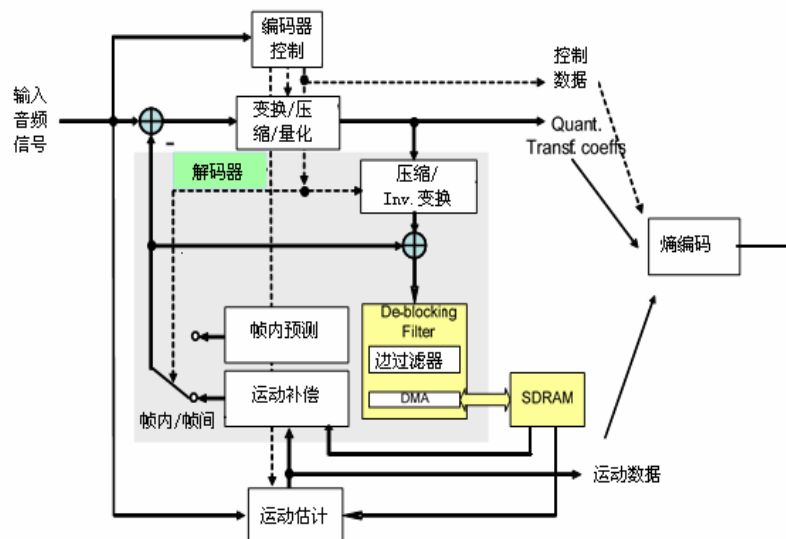


图 21-27 H.264 编码管线

### 21.10 重叠/分割过滤器

该滤波器处理适用在解码器和编码器中。FIMV-MFC V1.0 分割滤波器支持 H.264/H.263/MPEG4。对于 H.264 和 H.263，分割滤波器操作在编码循环中。滤波帧作为参考帧适用于并发的编码解码帧的运动补偿。但对 MPEG-4，分割滤波器操作以外的编码循环只为显示。

如图 21-28 所示，显示 H.264 的基本编码结构。



## 1. 处理方式

FIMV-MFC V1.0 支持两个操作模式,为每个标准关于是否滤波进程,在宏管线中适用于重建图像或不适用。

## 2. On-the-fly 模式

在 On-the-fly 模式中, 从重建中输出宏没有全部保存到外部 SDRAM 就马上过滤。DMA 控制器只传输重建的宏的一部分, 这些是由于缺乏邻近宏的重建像素数据而不能被过滤的。这个模式改善了带宽的功效。

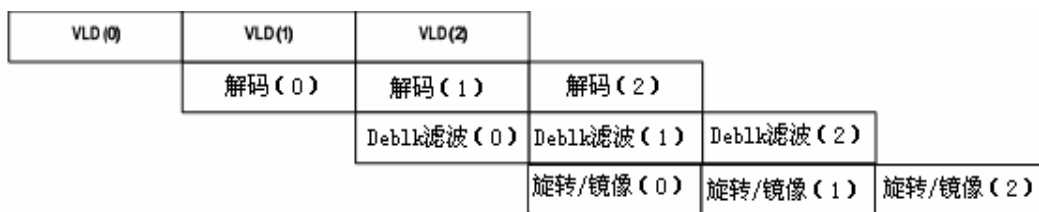


图 21-29 解码的管线结构

如图 21-29 所示, 说明解码时应用于 H. 263 和 H. 264 的标准的管线结构。数目之间的括号是指宏地址正在处理中。

### 3. 独立模式

另一种模式，被称为‘独立模式’，执行过滤过程后重建帧，而不是宏。在独立模式下，分割滤波器模块已经读取一个完整的帧的解码，并在过滤之后写入指定帧缓冲器中。它需要更多外部存储器总线的带

宽来比较 on-the-fly 模式。在 on-the-fly 模式中，输入到过滤器是来自编解码器中的一个缓冲器，而不是来自外部存储器。

当独立模式分割过滤时，编解码器的其它模块是空闲状态。因此，为处理完成一个帧的总时间，其时间要比 On-the-fly 模式的时间长。

经过滤的帧可以被存储到一个指定帧缓冲器中或改写到原始帧缓冲器中，输入帧继续存入滤波器上。

4. 方框图

如图 21-30 所示，显示重叠/分割滤波器的结构。过滤像素数据被存储在工作缓冲器中。输出数据被存储在输出缓冲器中，旋转/镜像块将读取这个输出缓冲器。输入缓冲器和 DMA 缓冲器是由于处理旋转/镜像而用来存储/加载中间数据。

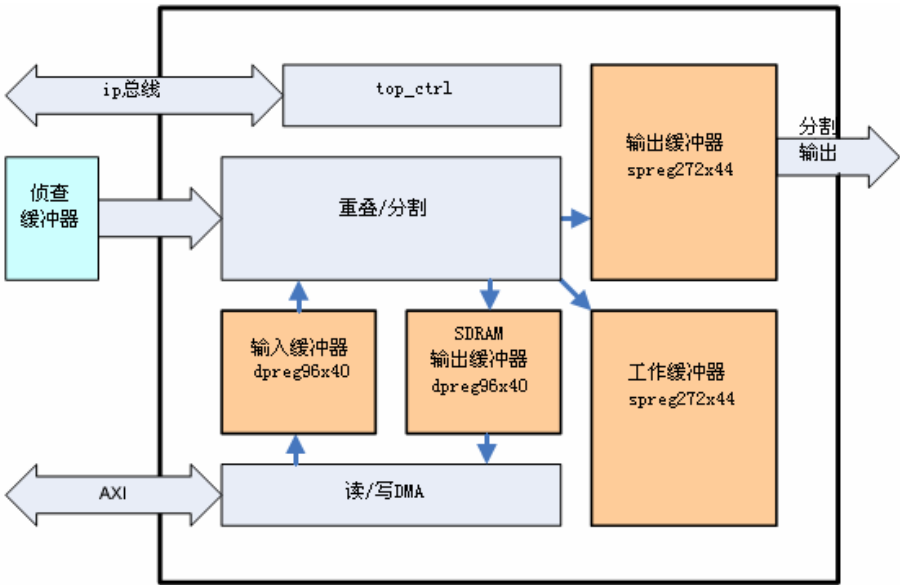


图 21-30 重叠/分割滤波器的结构

位处理器通过 IP 总线控制主控制器（top\_ctrl）。位处理器写入控制信息（处理模式，滤波模式（H. 263/H. 264/MPEG-4/VC-1），运动命令等等）到主控制器中的控制寄存器。

重叠平滑滤波器启动 VC-1 模式。在 VC-1 模式中，重叠平滑滤波器在分割滤波器之前被处理。分割/重叠平滑滤波器读取数据是从重建缓冲器到滤波器的。在独立模式下重建数据加载来自 SDRAM。分割/重叠平滑滤波器使用工作缓冲器作为暂时缓冲器为滤波处理。滤波处理完成后，为下一个管线电路级输出数据被移动到输出缓冲器。由于滤波宏边缘，邻近宏数据存储到工作缓冲器中。

## 5. H. 264 分割滤波器

滤波应适用于所有的  $4 \times 4$  图片边缘块，除边缘在图片的边界线上。滤波被执行在宏的基础上并被处理在增加宏地址的命令中。因为每个宏，垂直边缘首先被过滤从左到右，而且水平边缘从顶部到底部被过滤。

如图 21-31 所示，显示在 H. 264 模式中有效数据输出。

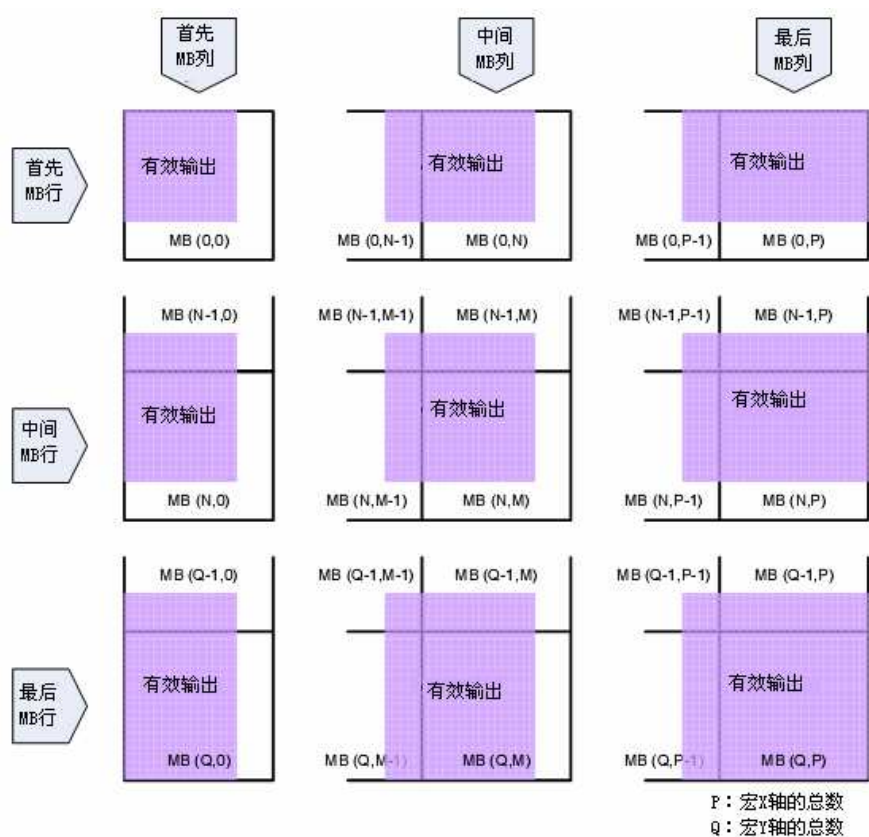


图 21-31 H. 264 模式中有效数据输出

## 6. H. 263 附加 J 分割滤波器

滤波在  $8 \times 8$  块边缘进行。横向边缘首先被过滤从顶端到低端，并垂直边缘被过滤从左到右。像素被使用在滤波横穿水平边缘不需要被改变由先前滤波横穿垂直边缘。位处理器设置滤波操作的参数。

如图 21-32 所示，显示 H.263 附加 J 模式的有效数据输出。

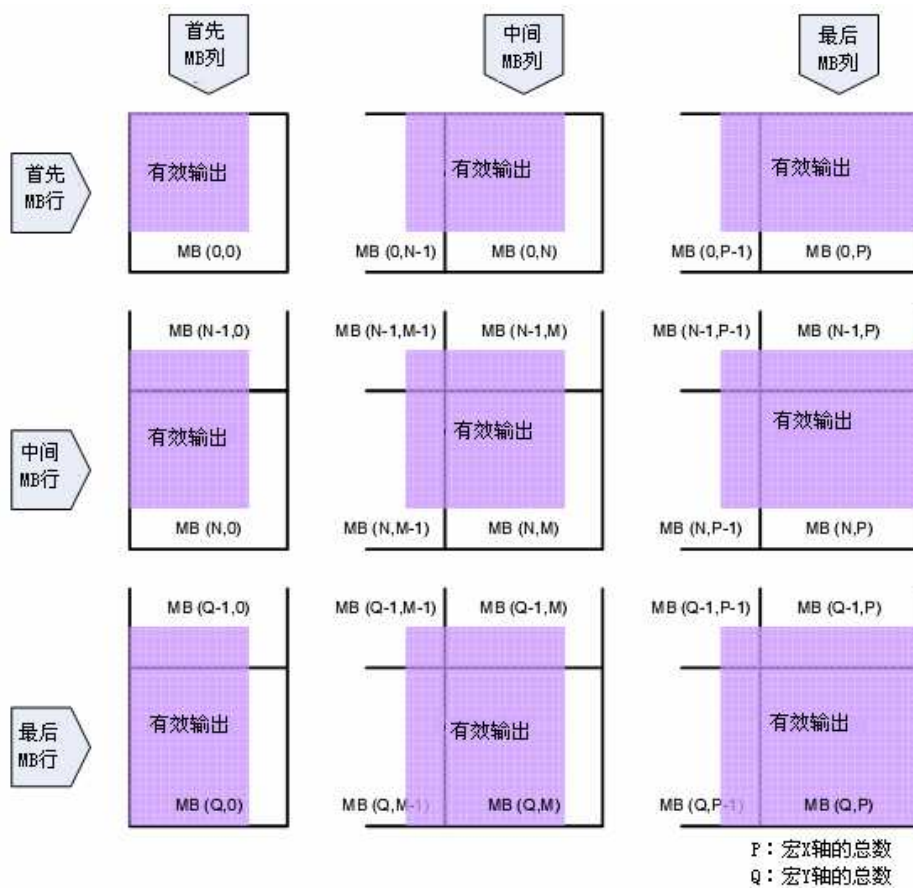


图 21-32 H.263 附加 J 模式的有效数据输出

## 7. VC-1 的重叠平滑滤波器

VC-1 的重叠平滑滤波器应被并发的执行解码帧并优先分割滤波器。8×8 块的边缘支持两个内部块。垂直边缘首先被过滤，其次水平边缘被过滤。并发过滤时，常数值 128 将添加到块的每个像素，来重组输出。位处理器为重叠平滑滤波器写入邻近块信息。

如图 21-33 所示，显示 VC-1 的重叠平滑滤波器的有效数据输出。

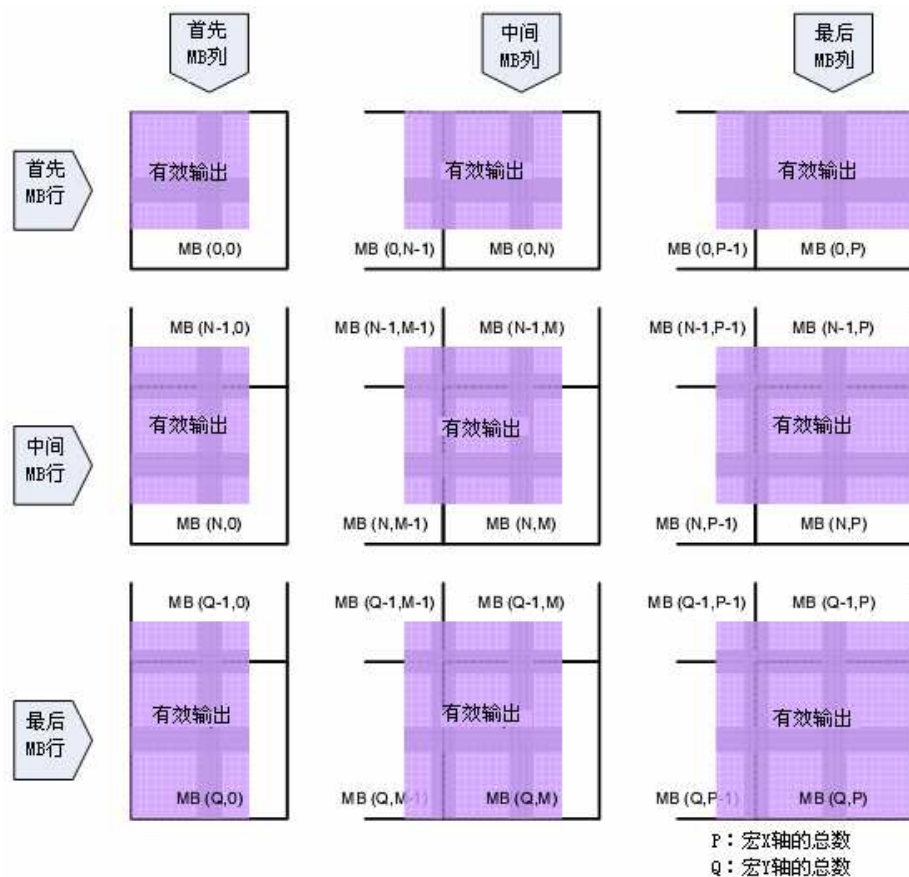


图 21-33 VC-1 的重叠平滑滤波器的有效数据输出

## 8. VC-1 分割滤波器

VC-1 分割滤波器过程在像素上操作，接近周边的块。在 P 图片中块边界可能发生在每个第 4，第 8，第 12 等像素行或列上。过滤 I 图片发生在第 8，第 16，第 24 等像素行或列上。横向边界线将首先被过滤，其次垂直线被过滤。所有块和子块有一个水平边界线沿着第 8，第 16，第 24 等水平线将被过滤。其次，所有子块有横向边界线沿着第 4，第 12，第 20 等水平线被过滤。然后，所有块和子块有垂直边界线沿着第 8，第 16，第 24 等垂直线被过滤。最后，所有子块有垂直边界线沿着第 4，第 12，第 20 等垂直线被过滤。

如图 21-34 所示，显示 VC-1 分割滤波器的有效数据。由于重叠平滑滤波器，有效输出区域被左移。

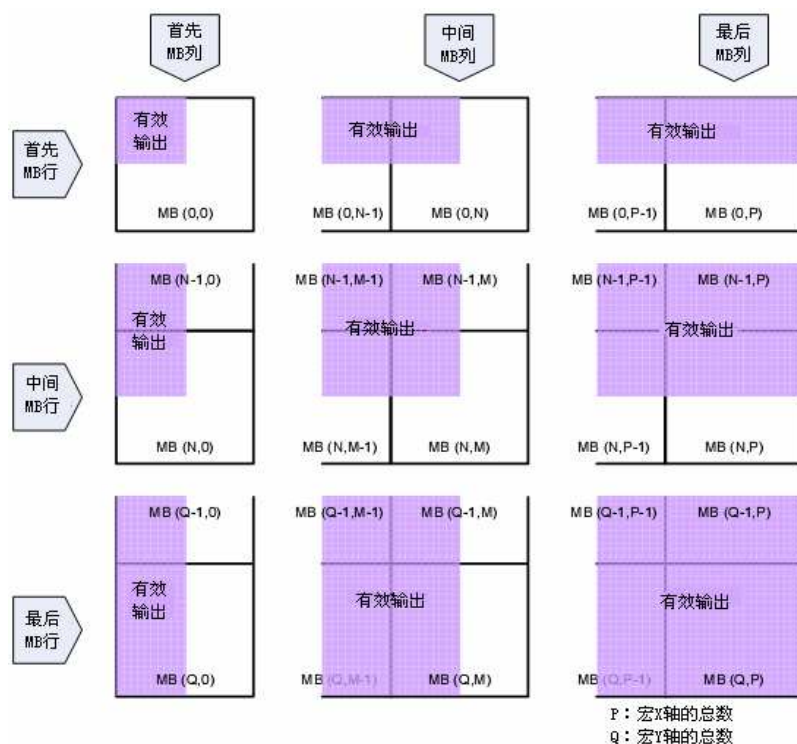


图 21-34 VC-1 分割滤波器的有效数据

## 9. MPEG-4 后处理的分割滤波器

FIMV-MFC 1.0 版本可以为 MPEG-4 将解码图像应用分割滤波器, 使用于 H. 264-like 滤波或 H. 263-like 滤波操作。位处理器产生合适的系数, 用于基于 MPEG-4 解码过程的选择模式。

## 21.11 系数缓冲器接口

系数缓冲器接口为位处理器提供一个通道, 来读取来自编码过程的量化系数或为解码过程发送可变长度解码系数。基于扫描类型上系数缓冲器接口也执行系数的重新排序。

### 1. 方框图

如图 21-35 所示, 说明系数缓冲器接口的方框图。所有子模块相关的处理系数作为输入或输出, 这就是连接系数缓冲器接口。位处理器设置扫描类型基于来自编码过程或解码过程的结果之上。删除读取系数为 0 值, 由标记寄存器-6×64 位来显示它。



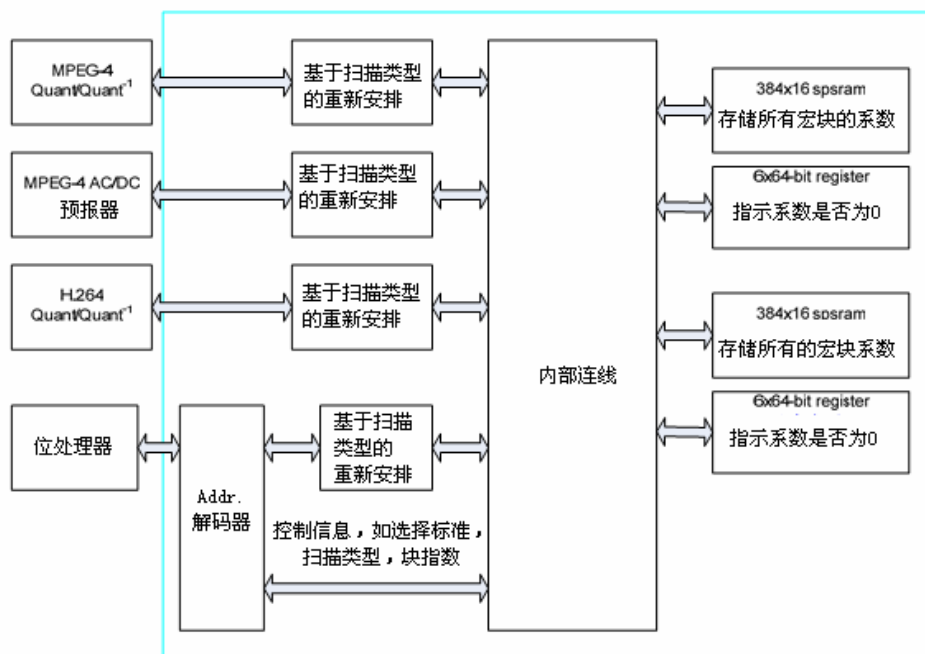


图 21-35 系数存储器接口方框图

## 2. 调整系数

位处理器基于扫描类型趋势执行重新排序。

下面列出扫描类型：

(1) H. 264

有关 H. 264 基线只适用 zig-zag 扫描类型。

(2) MPEG-4/H. 263P3

MPEG-4/H. 263P3 适用三种扫描类型，包括 zig-zag，交替水平和交替垂直扫描类型。其中扫描类型基于交流/直流预测标记和宏类型上被选定。在帧间宏类型情况下，只能用 zig-zag 扫描类型。由于帧内宏，如果交流预测标记是‘1’，扫描类型依赖于直流系数的预测指示。如果交流预测标记是‘0’，只应用 zig-zag 扫描类型。

## 3. 访问存储器系数

位处理器在系数存储器中读或写系数通过指定的块指数，扫描类型以及相应的系数指数。如图 21-36 所示，说明为处理器如何访问存储器系数。

(1) 块指数：在 MPEG-4/H. 263P3 情况下，每个  $8 \times 8$  块在 0~6 范围内都有自己的指数。对于 H. 264，块大小是  $4 \times 4$ ，指数范围是 -1~25。指数分配基于标准之上。



(2) 扫描类型：对于编码和解码，位处理器并没有考虑它应该写或读系数存储器的哪个位置的系数。重新排序它们基于指定扫描类型后，系数存储器接口写或读系数。

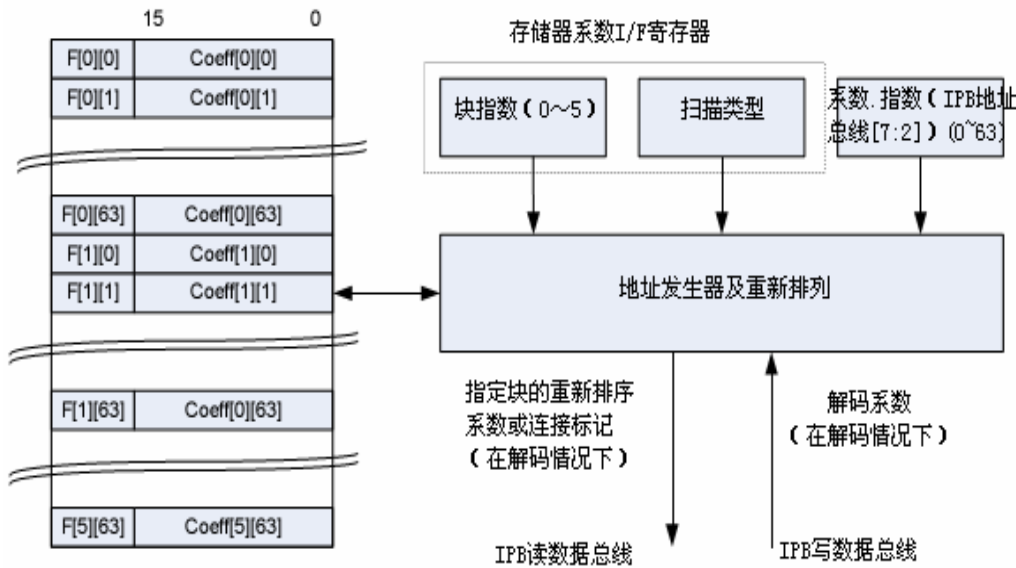


图 21-36 存储器系数访问

4. 编码器操作

在编码情况下，量化系数不需要任何调整即可写入系数缓冲器接口的内部存储器中。当处理器读取系数时，它们被重新排序并标记显示相应的系数有一个非零值被发送到位处理器来阻止是否有一个系数被编码。通过简单的计算非零位的数量，位处理器可以知道非零的系数的数量。因此，位处理器只读非零的系数。如图 21-37 和图 21-38 所示，显示 MPEG-4 编码案例和 H.264 编码案例。

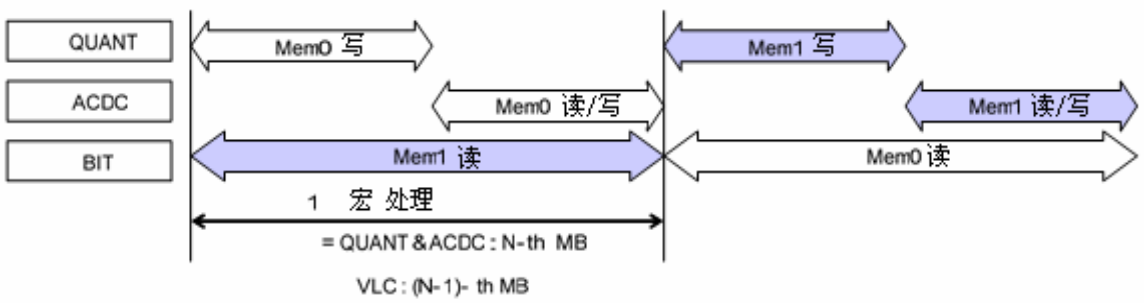


图 21-37 MPEG-4 编码案例

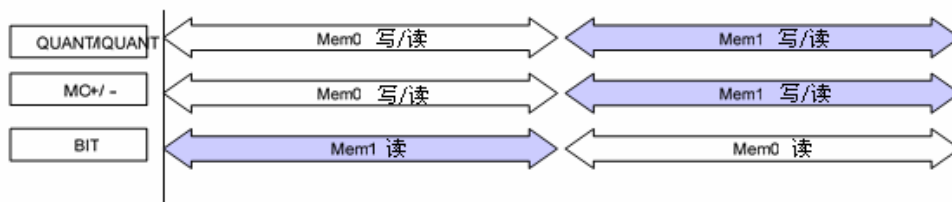


图 21-38 H.264 编码案例

## 5. 解码器操作

位处理器没有重新排序反向 zig-zag 扫描来写入解码系数。当逆量化模块从系数缓冲器接口读取系数时，执行重新排序的过程。如果系数相应标记为零，子模块如量化和 AC/DC 预测处理为零。如图 21-39 和图 21-40 所示，显示 MPEG-4 编码案例和 H.264 编码案例。

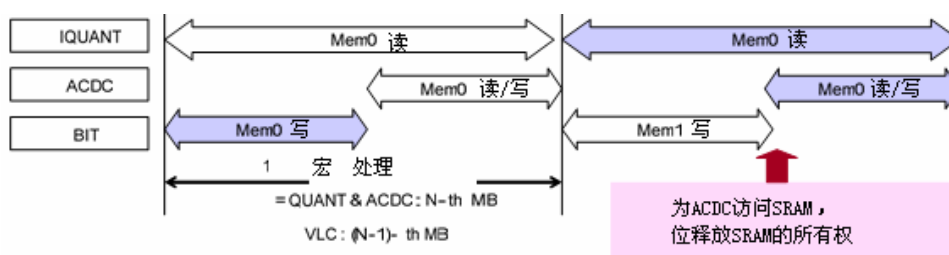


图 21-39 MPEG-4 解码案例

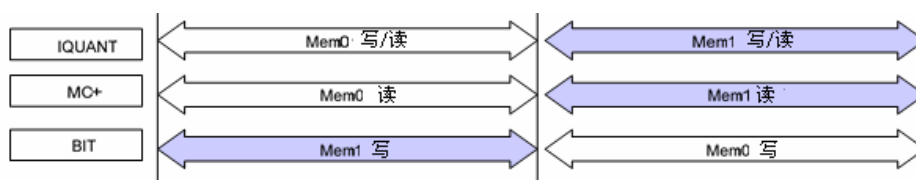


图 21-40 H.264 解码案例

注：在图 12-86，12-87，12-88 和 12-89 中，MC+意味重建，MC-意味处理余量。

## 6. 宏控制器

FIMV-MFC V1.0 有一个复杂而大量的管线，性能高。位处理器完全控制它是不合适的。因此，通过位处理器，FIMV-MFC V1.0 嵌入了宏控制器基于管线的配置上所有的视频编解码器的子模块。这个计划降低了位处理器的加载和保证 IP 的可编程。

视频编解码器编码或解码一个宏，位处理器配置编解码器的管线是如何构成的。如果为编码/解码一

个宏，所有进程被完成，宏控制器显示它的完成。

总之，位处理器配置，为当前宏处理，其中子模块启动，并宏控制器控制相应的子模块基于配置之上。如图 21-41 所示，显示宏控制器的连接。

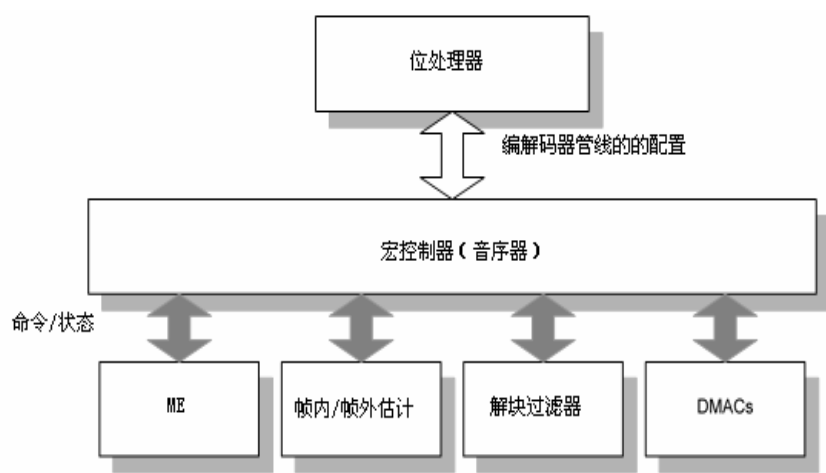


图 21-41 宏控制器的连接

## 21.12 FIMV-MFC V1.0 编程模式（特殊功能寄存器）

FIMV-MFC V1.0 通过 APB 总线接口与一个主机处理器相通。如表 21-8 所示，说明区域的地址图可以通过 APB 被存取。

表 21-8 内部寄存器地址图

PADDR[11:9]	模块	描述
3' b000	位处理器的主机接口	在正常运作中，主机控制器无法访问这些寄存器
3' b001	宏控制器（音序器）	
3' b010	系数存储器接口	
3' b011	分割滤波器	
3' b100	运动估值	
3' b101	帧间估计	
3' b110	VC-1 定标器	
3' b111	S/W RESET	Soft-ware 重置模块

## 1. 主机接口寄存器

位处理器寄存器分为两类。

地址 0x000~0x0FC（64 位寄存器地址空间）是 H/W 寄存器。这些寄存器有重置值，并且它们的功能是固定的（无可配置）。地址 0x100~0x1FC（64 位寄存器）是通用 S/W 寄存器。它们没有重置值并通过位固件可配置的。它们被用来作为接口在主机和位处理器之间。高位 32 位寄存器（地址 0x100~0x17C）用来作为静态参数。这些寄存器的含义或功能没有被改变，用于所有种类运动命令（SEQ\_INIT，SEQ\_END，PICTURE\_RUN），并且适用于所有命令。低位 32 位寄存器（地址 0x180~0x1FC）用来作为暂时命令自变量。这些寄存器的含义或功能被改变，用于每个运动命令。

## 2. 主机接口寄存器的概要

如表 21-9 所示，显示了位处理器共同寄存器概要说明。

表 21-9 位处理器共同寄存器概要（基址=0x7E002000）

地址	类型	宽度	复位值	名称	描述
BASE+0x000	写		0	CodeRun	位运动启动
BASE+0x004	写		0	CodeDownLoad	代码下载数据寄存器
BASE+0x008	写		0	HostIntReq	主机中断请求至位
BASE+0x00C	写		0	BitIntClear	位中断清除
BASE+0x010	读		0	BitIntSts	位中断状态
BASE+0x014	写		0	BitCodeReset	位代码重置
BASE+0x018	读		0	BitCurPc	位通用 PC
BASE+0x100	读/写		N/A	CodeBufAddr	代码表 SDRAM 地址
BASE+0x104	读/写		N/A	WorkBufAddr	工作缓冲 SDRAM 地址
BASE+0x108	读/写		N/A	ParaBufAddr	自变量/返回参数缓冲 SDRAM 地址
BASE+0x10C	读/写		N/A	BitStreamCtrl	位流缓冲控制
BASE+0x110	读/写		N/A	FrameMemCtrl	帧存储器控制
BASE+0x114	读/写		N/A	DecFuncCtrl	解码器功能控制 1
BASE+0x11C	读/写		N/A	BitWorkBufCtrl	工作 Buf 控制 2
BASE+0x120	读/写		N/A	BitStreamRdPtr0	位流缓冲器读运动指数 0 的地址

BASE+0x124	读/写		N/A	BitStreamWrPtr0	位流缓冲器写运动指数 0 的地址
BASE+0x128	读/写		N/A	BitStreamRdPtr1	位流缓冲器读运动指数 1 的地址
BASE+0x12C	读/写		N/A	BitStreamWrPtr1	位流缓冲器写运动指数 1 的地址
BASE+0x130	读/写		N/A	BitStreamRdPtr2	位流缓冲器读运动指数 2 的地址
BASE+0x134	读/写		N/A	BitStreamWrPtr2	位流缓冲器写运动指数 2 的地址
BASE+0x138	读/写		N/A	BitStreamRdPtr3	位流缓冲器读运动指数 3 的地址
BASE+0x13C	读/写		N/A	BitStreamWrPtr3	位流缓冲器写运动指数 3 的地址
BASE+0x140	读/写		N/A	BitStreamRdPtr4	位流缓冲器读运动指数 4 的地址
BASE+0x144	读/写		N/A	BitStreamWrPtr4	位流缓冲器写运动指数 4 的地址
BASE+0x148	读/写		N/A	BitStreamRdPtr5	位流缓冲器读运动指数 5 的地址
BASE+0x14C	读/写		N/A	BitStreamWrPtr5	位流缓冲器写运动指数 5 的地址
BASE+0x150	读/写		N/A	BitStreamRdPtr6	位流缓冲器读运动指数 6 的地址
BASE+0x154	读/写		N/A	BitStreamWrPtr6	位流缓冲器写运动指数 6 的地址
BASE+0x158	读/写		N/A	BitStreamRdPtr7	位流缓冲器读运动指数 7 的地址
BASE+0x15C	读/写		N/A	BitStreamWrPtr7	位流缓冲器写运动指数 7 <sup>3</sup> 的地址
BASE+0x160	读		N/A	BusyFlag	处理器忙碌标记
BASE+0x164	读/写		N/A	RunCommand 4	运动命令
BASE+0x168	读/写		N/A	RunIndex	运动进程指数
BASE+0x16C	读/写		N/A	RunCodStd	运动编解码器标准
BASE+0x170	读/写		N/A	IntEnable	中断启动
BASE+0x174	读/写		N/A	IntReason	中断原因
为内部使用的保护					
BASE+0x180 ~ BASE + 1D8	读/写		N/A	Command I/O Reg	命令 I/O 寄存器

<sup>1</sup> 这个控制寄存器的功能是由主机处理器新增提供一个扩展控制配置。

<sup>2</sup> 这个缓冲控制寄存器的工作是有主机处理器新增的支持缓冲配置的工作。

<sup>3</sup> 新增的从 0x140 到 0x15C 寄存器，可支持最多 8 种实例。

<sup>4</sup> 为检查 F/W 版本的一个新命令，用来新增正好的安排。

如表 21-10 所示，显示了 DEC\_SEQ\_INIT 参数寄存器的概要说明。

表 21-10 DEC\_SEQ\_INIT 参数寄存器概要

DEC_SEQ_INIT				
	地址	类型	名称	描述
输入 自变量	BASE+0x180	读/写	CMD_DEC_SEQ_BIT_BUF_START	位流缓冲器地址
	BASE+0x184	读/写	CMD_DEC_SEQ_BIT_BUF_SIZE	位流缓冲器大小
	BASE+0x188	读/写	CMD_DEC_SEQ_OPTION5	解码序列的选择
	BASE+0x18C	读/写	CMD_DEC_SEQ_PRO_BUF	进程缓冲器地址
	BASE+0x190	读/写	CMD_DEC_SEQ_TMP_BUF_1	暂时缓冲器 1 地址
	BASE+0x194	读/写	CMD_DEC_SEQ_TMP_BUF_2	暂时缓冲器 2 地址
	BASE+0x198	读/写	CMD_DEC_SEQ_TMP_BUF_3	暂时缓冲器 3 地址
	BASE+0x19C	读/写	CMD_DEC_SEQ_TMP_BUF_4	暂时缓冲器 4 地址
	BASE+0x1A0	读/写	CMD_DEC_SEQ_TMP_BUF_5	暂时缓冲器 5 地址
	BASE+0x1A4	读/写	CMD_DEC_SEQ_START_BYTE	有效流数据的起始字节
输出返 回	BASE+0x1C0	读	RET_DEC_SEQ_SUCCESS	命令执行结果的状态
	BASE+0x1C4	读	RET_DEC_SEQ_SRC_SIZE	解码初始图大小
	BASE+0x1C8	读	RET_DEC_SEQ_SRC_F_RATE	解码初始帧速率
	BASE+0x1CC	读	RET_DEC_SEQ_FRAME_NEED	必须的最低解码帧缓冲器
	BASE+0x1D0	读	RET_DEC_SEQ_FRAME_DELAY	最大显示帧缓冲器延迟
	BASE+0x1D4	读	RET_DEC_SEQ_INFO <sup>7</sup>	解码序列信息

<sup>5</sup> 表现文件播放模式的位，动态缓冲器分配启动被添加这个寄存器。

<sup>6</sup> 从 18C 到 1A0，这个寄存器被添加提供一种用来工作缓冲器配置的方式。

<sup>7</sup> 表现附件 J 指示的位，被添加这个寄存器。

如表 21-11 所示，显示了 ENC\_SEQ\_INIT 参数寄存器的概要说明。

表 21-11 ENC\_SEQ\_INIT 参数寄存器概要

ENC_SEQ_INIT				
	地址	类型	名称	描述
	BASE+0x180	读/写	CMD_ENC_SEQ_BIT_BUF_START	流缓冲器地址

输入 自变量	BASE+0x184	读/写	CMD_ENC_SEQ_BIT_BUF_SIZE	流缓冲器大小
	BASE+0x188	读/写	CMD_ENC_SEQ_OPTION	编码序列选择
	BASE+0x18C	读/写	CMD_ENC_SEQ_COD_STD	编码译码标准
	BASE+0x190	读/写	CMD_ENC_SEQ_SRC_SIZE	编码初始帧大小
	BASE+0x194	读/写	CMD_ENC_SEQ_SRC_F_RATE	编码初始帧速率
	BASE+0x198	读/写	CMD_ENC_SEQ_MP4_PARA	编码的 MPEG4 参数
	BASE+0x19C	读/写	CMD_ENC_SEQ_263_PARA	编码的 H. 263 参数
	BASE+0x1A0	读/写	CMD_ENC_SEQ_264_PARA	编码的 H. 264 参数
	BASE+0x1A4	读/写	CMD_ENC_SEQ_SLICE_MODE	编码片模式
	BASE+0x1A8	读/写	CMD_ENC_SEQ_GOP_NUM	编码 GOP 数量
	BASE+0x1AC	读/写	CMD_ENC_SEQ_RC_PARA	编码率控制参数
	BASE+0x1B0	读/写	CMD_ENC_SEQ_RC_BUF_SIZE	编码率控制缓冲器大小
	BASE+0x1B4	读/写	CMD_ENC_SEQ_INTRA_MB	编码内部 MB 刷新数量
	BASE+0x1B8	读/写	CMD_ENC_SEQ_FMO	在 H. 264 Enc 中 FMO 配置
	BASE+0x1D0	读/写	CMD_ENC_SEQ_TMP_BUF_1	临时缓冲器 1 地址
	BASE+0x1D4	读/写	CMD_ENC_SEQ_TMP_BUF_2	临时缓冲器 2 地址
	BASE+0x1D8	读/写	CMD_ENC_SEQ_TMP_BUF_3	临时缓冲器 2 地址
	BASE+0x1DC	读/写	CMD_ENC_SEQ_TMP_BUF_4	临时缓冲器 4 地址
输出返回	BASE+0x1C0	读	RET_ENC_SEQ_SUCCESS	命令执行结果状态

如表 21-12 所示，显示了 DEC\_PIC\_RUN 参数寄存器的概要说明。

表 21-12 DEC\_PIC\_RUN 参数寄存器概要

DEC_PIC_RUN				
	地址	类型	名称	描述
输入 自变量	BASE+0x180	读/写	CMD_DEC_PIC_ROT_MODE	显示帧后旋转模式
	BASE+0x184	读/写	CMD_DEC_PIC_ROT_ADDR_Y	后旋转帧存储在 Y 地址

	BASE+0x188	读/写	CMD_DEC_PIC_ROT_ADDR_CB	后旋转存储在 CB 地址
	BASE+0x18C	读/写	CMD_DEC_PIC_ROT_ADDR_CR	后旋转帧存储在 CR 地址
	BASE+0x190	读/写	CMD_DEC_PIC_DBK_ADDR_Y	解块帧存储在 Y 地址
	BASE+0x194	读/写	CMD_DEC_PIC_DBK_ADDR_CB	解块帧存储在 CB 地址
	BASE+0x198	读/写	CMD_DEC_PIC_DBK_ADDR_CR	解块帧存储在 CR 地址
	BASE+0x19C	读/写	CMD_DEC_PIC_ROT_STRIDE	后旋转帧的步幅
	BASE+0x1A8	读/写	CMD_DEC_PIC_CHUNK_SIZE	帧块大小
	BASE+0x1AC	读/写	CMD_DEC_PIC_BB_START	图片流缓冲器的 4 字节排队初始地址
	BASE+0x1B0	读/写	CMD_DEC_PIC_START_BYTE	有效流数据的初始字节
输出返回	BASE+0x1C0	读	RET_DEC_PIC_FRAME_NUM	解码帧的数量
	BASE+0x1C4	读	RET_DEC_PIC_IDX	显示帧的指数
	BASE+0x1C8	读	RET_DEC_PIC_ERR_MB_NUM	在解码图片中错误 MB 数量
	BASE+0x1CC	读	RET_DEC_PIC_TYPE	解码图片类型
	BASE+0x1D8	读	RET_DEC_PIC_SUCCESS	命令执行结果的状态

如表 21-13 所示，显示了 ENC\_PIC\_RUN 参数寄存器的概要说明。

表 21-13ENC\_PIC\_RUN 参数寄存器概要

ENC_PIC_RUN				
	地址	类型	名称	描述
输入自变量	BASE+0x180	读/写	CMD_ENC_PIC_SRC_ADDR_Y	输入初始帧缓冲器 Y 的 SDRAM 地址
	BASE+0x184	读/写	CMD_ENC_PIC_SRC_ADDR_CB	输入初始帧缓冲器 CB 的 SDRAM 地址
	BASE+0x188	读/写	CMD_ENC_PIC_SRC_ADDR_CR	输入初始帧缓冲器 CR 的 SDRAM 地址
	BASE+0x18C	读/写	CMD_ENC_PIC_QS	编码图片量化的步幅
	BASE+0x190	读/写	CMD_ENC_PIC_ROT_MODE	输入帧前旋转模式
	BASE+0x194	读/写	CMD_ENC_PIC_OPTION	编码图片选项
输出返回	BASE+0x1C0	读	RET_ENC_PIC_FRAME_NUM	编码帧数量
	BASE+0x1C4	读	RET_ENC_PIC_TYPE	编码图片类型



	BASE+0x1C8	读	RET_ENC_PIC_IDX	重建帧指数
	BASE+0x1CC	读	RET_ENC_PIC_SLICE_NUM	编码片的图片数量

如表 21-14 所示，显示了 SET FRAME BUFFER 参数寄存器的概要说明。

表 21-14 SET FRAME BUFFER 参数寄存器概要

SET_FRAME_BUFFER				
	地址	类型	名称	描述
输入自变量	BASE+0x180	读/写	CMD_SET_FRAME_BUF_NUM	编解码器使用帧缓冲器的数量
	BASE+0x184	读/写	CMD_SET_FRAME_BUF_STRIDE	帧缓冲线的步幅
输出返回				

如表 21-15 所示，显示了 ENC HEADER 参数寄存器的概要说明。

表 21-15 ENC HEADER 参数寄存器概要

ENC_HEADER				
	地址	类型	名称	描述
输入自变量	BASE+0x180	读/写	CMD_ENC_HEADER_CODE	编码标题代码
输出返回				

如表 21-16 所示，显示了 DEC PARA SET 参数寄存器的概要说明。

表 21-16 DEC PARA SET 参数寄存器概要

DEC_PARA_SET				
	地址	类型	名称	描述
输入自变量	BASE+0x180	读/写	CMD_DEC_PARA_SET_TYPE	序列/图片参数设置类型
	BASE+0x184	读/写	CMD_DEC_PARA_SET_SIZE	序列/图片参数设置 Rbsp 字节大小
输出返回				

如表 21-17 所示，显示了 ENC PARA SET 参数寄存器的概要说明。

表 21-17 ENC PARA SET 参数寄存器概要

ENC_PARA_SET				
	地址	类型	名称	描述
输入自变量	BASE+0x180	读/写	CMD_ENC_PARA_SET_TYPE	序列/图片参数设置类型
输出返回	BASE+0x1C0	读	RET_ENC_PARA_SET_SIZE	编码序列/图片参数设置 RBSP 字节大小

如表 21-18 所示，显示了 GET F/W VER 参数寄存器的概要说明。

表 21-18 GET F/W VER 参数寄存器概要

GET_F/W_VER				
	地址	类型	名称	描述
输入自变量				
输出返回	BASE+0x1C0	读	RET_GET_FW_VER	返回版本代码有以下格式： [31: 16]:产品号 (0xF202) [15:0]M.m.rr 的显示号 (0xMmrr)

下面我们主要对位处理器寄存器的描述。

**CodeRun (0x000)**

位	名称	类型	功能	复位值
0	CodeRun	写	0- 位处理器停止执行 1- 位处理器开始执行	

**CodeDownLoad (0x004)**

位	名称	类型	功能	复位值
15: 0	CodeData	写	16 位位代码下载数据	0
28: 16	CodeAddr	写	13 位位代码下载地址 位代码字地址 (16 位地址) *当前的设计有 4K 代码字空间 (8KB)。 因此 CodeAddr[12: 0]必须少于 4095	0

**HostIntReq (0x008)**

位	名称	类型	功能	复位值
0	IntReq	写	位处理器的中断请求。  主机可以写 ‘1’ 到这个寄存器，请求中断到位  *当前的固件版本不使用中断，从主机到位。因此，不使用这个寄存器	0

**BitIntClear (0x00C)**

位	名称	类型	功能	复位值
0	IntClear	写	写 ‘1’ 到这个寄存器，清除位中断到主机	0

**BitIntSts (0x010)**

位	名称	类型	功能	复位值
0	IntSts	读	1 意味那个位中断到主机是有效的  当主机写入 IntClear 寄存器 ‘1’，这个位被清除。	0

**BitCodeReset (0x014)**

位	名称	类型	功能	复位值
0	CodeReset	写	如果主机写 ‘1’ 到这个寄存器，位的程序计数器被设置为  ‘0’，因此重新启动到初始程序	0

**BitCurPc (0x018)**

位	名称	类型	功能	复位值
13: 0	CurPc	读	当前计划我饿处理器的计数器。  这个寄存器可能被用于调试效果	0

**CodeBufAddr (0x100)**

位	名称	类型	功能	复位值
31: 0	CodeBufAddr	读/写	位固件代码图像启动字节地址，其中在 SDRAM 中。	N/A

			主机必须设定启动位代码图像的 SDRAM 字节地址到这个寄存器之前，开始执行位处理器 *当前设计使用 80KB 为代码缓冲器。	
--	--	--	--	--

#### WorkBufAddr (0x104)

位	名称	类型	功能	复位值
31: 0	WorkBufAddr	读/写	位处理器工作缓冲器 SDRAM 的字节地址。 主机必须在 SDRAM 中保存工作缓冲，用于位处理器编码/解码	N/A

#### ParaBufAddr (0x108)

位	名称	类型	功能	复位值
31: 0	ParaBufAddr	读/写	位处理器参数的缓冲器 SDRAM 字节地址。 主机必须在 SDRAM 中保存参数缓冲器，用于位处理器命令执行自变量和返回数据。 *当前设计使用 8KB 为参数缓冲器	N/A

#### BitStreamCtrl (0x10C)

位	名称	类型	功能	复位值
0	SelBigEndian	读/写	0- 位流缓冲器是 4 字节无效格式 1- 位流缓冲器是有效	N/A
1	BufStsCheckDis	读/写	0- 位流缓冲器溢出/下溢出控制启动 1- 位流缓冲器溢出/下溢出控制禁止 如果位流缓冲器下溢发生在解码情况中，位处理器停止位流加载，如果位流缓冲器溢出发生在编码情况中，停止位流节省。如果这个标记是 ‘1’，位不控制位流缓冲器溢出/下溢出状态。	N/A
2	BufPicFlush	读/写	为 ‘1’ 的值意味那个位流缓冲器被刷新在编码图片的每个末端。 在编码情况下，编码一个图片之后，内在位流被刷新到外部 SDRAM 中。因此，整个编码位流数据被提供给主机。 如果这个标记是 ‘0’，内部位流缓冲器被刷新，只有当内部位流缓	N/A

			<p>冲器被填充到它的最大模式大小（512 字节）。因此，编码的最后一个图片，有些（低于 512 字节）最后编码数据没被刷新到外部 SDRAM，并只位于内部位流缓冲器中。刷新其余的编码数据位于内部位流缓冲器，主机必须执行 ENC_SEQ_END 命令。</p> <p>当[BufPicReset]标记是 ‘0’，这个标记是有效的，在解码情况下，这个标记被忽略。</p>	
3	BufPicReset	读/写	<p>值位 ‘1’ 意味着那个位流缓冲器被复位为每个图片编码/解码命令。</p> <p>在编码情况下，编码一个图片之后，位流缓冲器被刷新到外部 SDRAM 上，下一个图片编码数据被写入位流缓冲器的初始。因此，主机必须得到编码数据在编码图片的最后。</p> <p>如果这个标记是 “1”，[BufPicFlush]位被忽略。</p> <p>在解码情况下，这个标记被忽略。</p>	N/A

#### FrameMemCtrl (0x110)

位	名称	类型	功能	复位值
0	SelBigEndian	读/写	<p>0: 帧存储器是 4 字节最少有效格式</p> <p>1: 帧存储器是最大有效</p>	N/A

#### DecFuncCtrl (0x114)

位	名称	类型	功能	复位值
0	StreamEnd	读/写	<p>0: 有更多位流需要送到解码器</p> <p>1: 整个位流已经送到解码器</p> <p>在以下初始状态下，这个标记使用为发信号到位处理器，为了躲避来自 SEQ_INIT 的延迟状态。当这个设置为 1，然后位处理器放弃分析更多流数据，并逃避延迟状态由于返回值是 0（失败）。</p> <p>在编码情况下这个标记被忽略。</p>	N/A

**BitWorkBufCtrl (0x11C)**

位	名称	类型	功能	复位值
0	WorkBufConfig	读/写	0: 工作缓冲器配置设置为禁止 1: 工作缓冲器配置设置为启动	N/A

**BitStreamRdPtr0 (0x120)**

位	名称	类型	功能	复位值
31: 0	StreamRdPtr0	读/写	<p>在解码情况下, 当前的外部 SDRAM 的位流缓冲器读取进程的地址指数 0, 通过位处理器被设置到这个寄存器。</p> <p>在编码情况下, 主机必须设置当前外部位流缓冲器读取进程的地址指数 0 到这个寄存器。</p> <p>这个寄存器通过位处理器更新每个位流数据的加载和自动包围。</p> <p>*为每个传输, 当前设计加载 512 字节到内部缓冲器。因此, 加载数据完成后, 位流读取指示器被增强到 512。</p>	N/A

**BitStreamWrPtr0 (0x124)**

位	名称	类型	功能	复位值
31: 0	StreamWrPtr0	读/写	<p>在解码情况下, 主机必须设定当前外部位流缓冲器写入进程地址指数 0 到这个寄存器上。</p> <p>在编码情况下, 当前外部 SDRAM 的位流缓冲器写入进程的地址指数 0 通过位处理器被设置在这个寄存器上。</p> <p>这个寄存器通过位处理器更新每个位流数据的保存并自动包围。</p> <p>*为每个传输, 当前设计保存 512 字节来自内部缓冲器。因此, 保存数据完成后, 位流写入指示器被增加到 512.</p>	N/A

**BitStreamRdPtr1 (0x128)**

位	名称	类型	功能	复位值
31: 0	StreamRdPtr1	读/写	外部 SDRAM 的位流缓冲器读取进程的地址指数 1	N/A

**BitStreamWrPtr1 (0x12C)**

位	名称	类型	功能	复位值
31: 0	StreamWrPtr1	读/写	外部 SDRAM 的位流缓冲器写入进程的地址指数 1	N/A

**BitStreamRdPtr2 (0x130)**

位	名称	类型	功能	复位值
31: 0	StreamRdPtr2	读/写	外部 SDRAM 的位流缓冲器读取进程的地址指数 2	N/A

**BitStreamWrPtr2 (0x134)**

位	名称	类型	功能	复位值
31: 0	StreamWrPtr2	读/写	外部 SDRAM 的位流缓冲器写入进程的地址指数 2	N/A

**BitStreamRdPtr3 (0x138)**

位	名称	类型	功能	复位值
31: 0	StreamRdPtr3	读/写	外部 SDRAM 的位流缓冲器读取进程的地址指数 3	N/A

**BitStreamWrPtr3 (0x13C)**

位	名称	类型	功能	复位值
31: 0	StreamWrPtr3	读/写	外部 SDRAM 的位流缓冲器写入进程的地址指数 3	N/A

**BitStreamRdPtr4 (0x140)**

位	名称	类型	功能	复位值
31: 0	StreamRdPtr4	读/写	外部 SDRAM 的位流缓冲器读取进程的地址指数 4	N/A

**BitStreamWrPtr4 (0x144)**

位	名称	类型	功能	复位值
31: 0	StreamWrPtr4	读/写	外部 SDRAM 的位流缓冲器写入进程的地址指数 4	N/A

**BitStreamRdPtr5 (0x148)**

位	名称	类型	功能	复位值
31: 0	StreamRdPtr5	读/写	外部 SDRAM 的位流缓冲器读取进程的地址指数 5	N/A

**BitStreamWrPtr5 (0x14C)**

位	名称	类型	功能	复位值
31: 0	StreamWrPtr5	读/写	外部 SDRAM 的位流缓冲器写入进程的地址指数 5	N/A

**BitStreamRdPtr6 (0x150)**

位	名称	类型	功能	复位值
31: 0	StreamRdPtr6	读/写	外部 SDRAM 的位流缓冲器读取进程的地址指数 6	N/A

**BitStreamWrPtr6 (0x154)**

位	名称	类型	功能	复位值
31: 0	StreamWrPtr6	读/写	外部 SDRAM 的位流缓冲器写入进程的地址指数 6	N/A

**BitStreamRdPtr1 (0x158)**

位	名称	类型	功能	复位值
31: 0	StreamRdPtr7	读/写	外部 SDRAM 的位流缓冲器读取进程的地址指数 7	N/A

**BitStreamWrPtr1 (0x15C)**

位	名称	类型	功能	复位值
31: 0	StreamWrPtr7	读/写	外部 SDRAM 的位流缓冲器写入进程的地址指数 7	N/A

**BusyFlag (0x160)**

位	名称	类型	功能	复位值
0	BusyFlag	读	值为 ‘0’ 意味位处理器已准备主机命令 值为 ‘1’ 意味位处理器已执行主机命令，但是未完成 写入运行命令寄存器之前，主机必须控制这个位。如果这个位	N/A



			是 ‘1’，主机必须等待知道值为 ‘0’ 来设置命令。	
--	--	--	-----------------------------	--

### RunCommand (0x164)

位	名称	类型	功能	复位值
2: 0	RunCommand	读/写	<p>主机写入命令代码到这个寄存器。</p> <p>命令代码</p> <p>3’ b001 (SEQ_INIT): 编码/解码序列初始化。在编码情况下，位处理器分析编码参数和编码序列标题。在解码情况下，位处理器解码序列标题和报告序列标题信息。</p> <p>3’ b010 (SEQ_END): 终止编码/解码序列。在编码情况下，位处理器刷新内部位流缓冲器到外部位流缓冲器。在解码情况下，位处理器终止进程。</p> <p>3’ b011 (PICURE_RUN): 编码/解码一个图片。位处理器编码/解码一个图片。</p> <p>3’ b100 (SET_FRAME_BUF): 设置解码/重建帧缓冲器 SDRAM 的地址和最高帧缓冲器数量。在编码/解码图片运行命令，主机必须通知帧缓冲器 SDRAM 地址到位处理器，然后位处理器安排帧缓冲器为解码/重建图像并返回帧缓冲器指数到主机，在编码/解码图片的最后。</p> <p>3’ b101 (ENCODE HEADER): 编码标题。例如在 H. 264 情况下，SPS (序列参数设置)，PPS (图片参数设置) 可通过这个命令擦插入图片边界之间。</p> <p>3’ b110 (ENC PARA SET): 编码 SPS，PPS 到位处理器的参数设置缓冲器。H. 264 中，主机可以通过这个命令获得 SPS/PPS。</p> <p>3’ b111 (DEC PARA SET): 增加 SPS，PPS 到位处理器的参数设置缓冲器。H. 264 中，为使用在解码经进程中，多种 SPS/PPS 被允许并主机可能通知其中一个参数设置到位处理器。</p> <p>4’ b1111 (GET F/W VER): 控制 F/W 版本的命令。这个命令只适用于杜宇 F/W 版本 1. 2. 5。</p>	N/A

**RunIndex (0x168)**

位	名称	类型	功能	复位值
1: 0	RunIndex	R/W	每写一个运动命令之前，主机写入编解码器进程指数到这个寄存器。  位处理器可以同时执行最大值为 4 的编码/解码进程。如果有一个以上的进程被运行，每个进程通过这个寄存器必须被赋值不同的进程指数。例如，当 1MPEG 解码器+1AVC 解码器+1AVC 编码器同时运行，MPEG4 解码器进程指数为 ‘0’， AVC 解码器进程指数为 ‘1’， AVC 编码器进程指数为 ‘2’。	N/A

**RunCodStd (0x16C)**

位	名称	类型	功能	复位值
2: 0	CodSrd	读/写	每写入一个运动命令之前，主机写入编解码器标准指数的代码到这个寄存器中。  3’ b000: MPEG4/H. 263 解码器 3’ b001: MPEG4/H. 263 编码器  3’ b010: H. 264 解码器 3’ b011: H. 264 编码器  3’ b100: VC-1 解码器	N/A

**IntEnable (0x170)**

位	名称	类型	功能	复位值
15: 0	IntEnable	读/写	中断启动标记寄存器。  这个寄存器的每个位是各种中断的中断启动标记。  第 1 位: SEQ_INIT 命令执行完成 第 2 位: SEQ_END 命令执行完成 第 3 位: PIC_RUN 命令执行完成 第 4 位: SET_FRAME_BUF 命令完成 第 5 位: ENC_HEADER 命令完成 第 6 位: ENC_PARA_SET 命令完成 第 7 位: DEC_PARA_SET 命令完成	N/A

			第 8 位～第 13 位：保留	
			第 14 位：外部位流缓冲器在解码情况下为空状态	
			第 15 位：外部位流缓冲器在解码情况下为满状态	

#### IntReason (0x174)

位	名称	类型	功能	复位值
15: 0	IntReason	读/写	中断理由标记寄存器。  这个寄存器的每个位是每个中断的中断的报告标记。“1”是指中断产生，“0”是指不产生中断。  每个位领域的中断匹配相同于 IntEnable 寄存器。	N/A

#### CMD\_DEC\_SEQ\_BIT\_BUF\_START (0x180)

位	名称	类型	功能	复位值
31: 0	BitBufAddr	读/写	位流缓冲器 SDRAM 字节地址  位流缓冲器必须是 512 字节对齐。  执行 DEC_SEQ_INIT 命令之前，主机必须写入这个寄存器	DEC_SEQ_INIT

#### CMD\_DEC\_SEQ\_BIT\_BUF\_SIZE (0x184)

位	名称	类型	功能	复位值
13: 0	BitBufSize	读/写	在千字节中的位流缓冲器大小  注意执行 DEC_SEQ_INIT 命令之前，主机必须写入这个寄存器  最大位流缓冲器大小是 4G 字节	DEC_SEQ_INIT

#### CMD\_DEC\_SEQ\_OPTION (0x188)

位	名称	类型	功能	指令
0	Mp4DbkOn	读/写	MPEG4 分离模块滤波器。  如果该标志是“1”，MPEG-4分离模块滤波器有效，并且过滤的图像存储到DbkAddrY、DbkAddrCb、 DbkAddrCr地址。  解码的图像（指向分离模块滤波器）也被存储以备将来运动补偿参	DEC_SEQ_INIT

			考。 只有在 MPEG4 / H. 263 条件下该标志有效。 H. 263 条件下，如果附件 J 打开，该标志被忽略，并且 H. 263+ 附件 J 分离模块滤波器被执行。如果附件 J 关闭，MPEG-4 分离模块滤波器有效（同于 MPEG-4）。	
1	ReorderEn	读/写	H. 264 解码条件下，显示缓冲区再排序有效。 在 H. 264 条件下，如果 pic_order_cnt_type 设置为“0”或者“1”输出解码图片可能再排序。在那种情况下，解码器必须延迟输出显示用于再排序，但是一些应用软件（例如，视频电话）不想有这样的显示延迟。 主机可能设置该标志为“0”来使数据显示缓冲器再排序无效。那么当 ic_order_cnt_type 是“0”或者“1”时，BIT 处理器不再对数据缓冲器进行再排序。在 pic_order_cnt_type 是 2 或者 MPEG4/H. 263 的情况下，该标志被忽略，因为输出显示缓冲器再排序不被允许。 如果该表示是“1”，BIT 处理器执行输出解码的图片再排序，并且达到[RET_DEC_SEQ_FRAME_DELAY]，数据显示被延迟。	
2	FilePlayEn	读/写	在对基于帧的流进行解码操作时，文件显示模式有效	
3	DecDynBufA llocEn	读/写	文件显示模式下，动态图片流缓冲区设置有效 如果文件显示模式有效，通过 DEC_PIC_RUN 指令给予的流缓冲区地址将用于代替 DEC_SEQ_INIT 指令给予的流缓冲区地址。通过使用这中动态的设置，请求能在流中高效率地完成。	
5:4	设置为 0			
6 <sup>12</sup>	VC1_Reode r_Disable	读/写	VC-1 解码，再排序无效 只用于测试。	

#### CMD\_DEC\_SEQ\_PRO\_BUF (0x18C)

位	名称	类型	功能	指令
31:0	ProcessBufAddr	读/写	处理缓冲器 SDRAM 字节地址	DEC_SEQ_INIT

			<p>处理缓冲器必须是 256 字节对齐。</p> <p>在执行 DEC_SEQ_INIT 指令前，主机必须写入该寄存器。</p> <p>处理缓冲器被用做 PS 数据保存缓冲区（对于 AVC）和 MV 直接预测缓冲区（对于 VC1），不用于 mpeg4。</p> <p>处理缓冲器必须比 <math>MB\ number \times 4</math>（对于 VC1）大。处理缓冲区不能预测，因此，主机处理临时分配一些数。</p> <p>实例关闭前，处理缓冲器必须保持。因此，主机处理器必须给处理缓冲器分配实例数。</p>	
--	--	--	---	--

#### CMD\_DEC\_SEQ\_TMP\_BUF\_1 (0x190)

位	名称	类型	功能	指令
31:0	TempBufAddr	读/写	<p>临时缓冲器 SDRAM 字节地址</p> <p>临时缓冲器必须是 256 个字对齐。</p> <p>执行 DEC_SEQ_INIT 指令前，主机必须写入该缓冲器。</p> <p>临时缓冲器 1 被用做 ACDC 预测缓冲器（对于 mpeg4）、内部预测 Y 缓冲器（对于 AVC）和 ACDC 预测缓冲器（对于 VC1）。</p> <p>临时缓冲器 1 必须比 <math>picWidth \times 8</math>（对于 mpeg4）大，AVC 是 <math>stride \times picHeight / 16</math>，VC1 是 <math>picWidth \times 8</math>。</p>	DEC_SEQ_INIT

#### CMD\_DEC\_SEQ\_TMP\_BUF\_2 (0x194)

位	名称	类型	功能	指令
31:0	TempBufAddr	读/写	<p>临时缓冲器 SDRAM 字节地址</p> <p>临时缓冲器必须是 256 个字节对齐。</p> <p>在执行 DEC_SEQ_INIT 指令之前主机必须写入该寄存器。</p> <p>临时缓冲器 2 用于数据分区部分 1 保存缓冲器（对于 mpeg4）、内部预测 Cb 缓冲器（对于 AVC）和解锁缓冲器（对于 VC1）。</p> <p>临时缓冲器 2 必须比 <math>stride / 2 \times picHeight / 2 / 16</math>（对于 AVC）和 <math>picWidth \times 10</math>（对于 VC1）大。在 mpeg4 情况下，临时缓冲器 2 大小不能预测出来。</p>	DEC_SEQ_INIT

**CMD\_DEC\_SEQ\_TMP\_BUF\_3 (0x198)**

位	名称	类型	功能	指令
31:0	TempBufAddr	读/写	<p>临时缓冲器 SDRAM 字节地址</p> <p>临时缓冲器必须是 256 字节对齐。</p> <p>在执行 DEC_SEQ_INIT 指令之前，主机必须写入给寄存器。</p> <p>临时缓冲器 3 用于数据分区部分 2 保存缓冲器（对于 mpeg4）和内部预测 Cr 缓冲器（对于 AVC），不用于 VC1。</p> <p>临时缓冲器 3 必须比 <math>\text{stride}/2 \times \text{picHeight}/2/16</math>（对于 AVC）大。在 mpeg4 情况下，不能预测临时缓冲器 2 的大小。</p>	DEC_SEQ_INIT

**CMD\_DEC\_SEQ\_TMP\_BUF\_4 (0x19C)**

位	名称	类型	功能	指令
31:0	TempBufAddr	读/写	<p>临时缓冲器 SDRAM 字节地址</p> <p>临时缓冲器必须是 256 个字节对齐。</p> <p>执行 DEC_SEQ_INIT 指令之前，主机写该寄存器。</p> <p>临时缓冲器 4 用于部分信息保存缓冲器（对于 AVC），不用于 mpeg4。</p> <p>最大临时缓冲器 4 是 <math>\text{MB number} \times 8</math>（对于 AVC）。</p>	DEC_SEQ_INIT

**CMD\_DEC\_SEQ\_TMP\_BUF\_5 (0x1A0)**

位	名称	类型	功能	指令
31:0	TempBufAddr	读/写	<p>临时缓冲器 SDRAM 字节地址</p> <p>临时缓冲器必须是 256 字节对齐。</p> <p>在 执行 DEC_SEQ_INIT 指令之前，主机必须写入该寄存器。</p> <p>临时缓冲器 5 用做部分保存缓冲器（对于 AVC），不用于 VC1 和 mpeg4。</p> <p>临时缓冲器 5 最大可能到 <math>\text{picwidth} \times \text{picheight} \times 1.5</math>。</p>	DEC_SEQ_INIT

**CMD\_DEC\_SEQ\_START\_BYTE (0x1A4)**

位	名称	类型	功能	指令
1:0	DecSeqValidByteStart	读/写	输入流缓冲器有效位流的字节地址	DEC_SEQ_INIT

**RET\_DEC\_SEQ\_SUCCESS (0x1C0)**

位	名称	类型	功能	指令
0	RetStatus	读	0 - DEC_SEQ_INIT 指令错误执行 1 - DEC_SEQ_INIT 指令成功执行	DEC_SEQ_INIT

**RET\_DEC\_SEQ\_SRC\_SIZE (0x1C4)**

位	名称	类型	功能	指令
9:0	PictureHeight	读	解码的图片高度大小（像素）	DEC_SEQ_INIT
19:10	PictureWidth	读	解码的图片宽度大小（像素）	

**RET\_DEC\_SEQ\_SRC\_F\_RATE (0x1C8)**

	名称	类型	功能	指令
15:0	FrameRateRes	读	解码图片帧速余量  例如： 如果[FrameRateDiv] = 30000 和 [FrameRateRes] = 1001 那么，视频帧速= 30000 / 1001 = 29.97 Hz 如果[FrameRateDiv] = 1 和 [FrameRateRes] = 15 那么， 视频帧速 = 15 / 1 = 15 Hz	DEC_SEQ_INIT
31:16	FrameRateDivMinus1	读	解码图片帧速的单位数量，以 Hz-1 计 [FrameRateDiv]源自增加该值到 1。	

**RET\_DEC\_SEQ\_FRAME\_NEED (0x1CC)**

	名称	类型	功能	指令
4:0	FrameBufNeed	读	最小解码帧缓冲器需要解码流成功  MPEG4/H.263 情况下，该值是 2（一个用于运动补偿	DEC_SEQ_INIT

			<p>参考，另一个用于当前帧储存）。</p> <p>在 H.264 情况下，该值可能比 2 大，最大值可能是 18（16 个用做参考，一个用做通用的，一个用于显示）。</p> <p>主机必须保留帧缓冲器的值为最小量。</p> <p>在 VC-1 情况下，该值包含旋转输出帧，因此，VC-1 时，当旋转器有效也不需要额外的旋转帧。</p> <p>例如，EIT 返回[FrameBufNeed]为 5，并且主机准备 7 帧缓冲器，通过设置 SET_FRAME_BUF 指令告知帧缓冲器地址。BIT 处理器安排 7 帧缓冲器和分派合适的帧缓冲器地址到解码图像数据存储。如果非 MMCO（存储管理控制操作）和输出再排序，BIT 处理器将分派解码帧缓冲器 0、1、2、3、4、5、6、0、1、2、… 而且参考帧覆盖不会发生。</p>	
--	--	--	---	--

#### RET\_DEC\_SEQ\_FRAME\_DELAY (0x1D0)

位	名称	类型	功能	指令
4:0	FrameBufDelay	读	<p>对于缓冲解码图片再排序，最大显示帧缓冲器</p> <p>当 H.264、pic_order_cnt_type 是“0”或“1”或者 VC-1 解码情况下，对于显示再排序，BIT 处理器可能延迟解码图片显示。</p> <p>例如，BIT 处理器返回[FrameBufDelay]到 5。在第一条 DEC_PIC_RUN 指令 5 张图片解码后，那么 BIT 处理器返回。因为在第一个 5 帧，没有解码图片要显示。</p> <p>最大的[FrameBufDelay]值可能是 16。如果 H.264 情况下，[ReorderEn]标志是“0”，则该值是“0”。</p> <p>在 VC-1 解码情况下，如果没有 B 图该值是“0”，如果有一个 B 图，不管[ReorderEn]表示是什么值，该值为“1”。</p>	DEC_SEQ_INIT



**RET\_DEC\_SEQ\_INFO (0x1D4)**

位	名称	类型	功能	指令
0	DataPartEn	读	0 - 数据分块无效 1 - 数据分块有效  执行完 DEC_SEQ_INIT 指令后, BIT 从解码队列头的信息, 将数据有效标志写入该寄存器。  在编码情况下不使用该寄存器。	DEC_SEQ_INIT
1	RevVlcEn	读	0 - 正常 VLC 表使用 1 - 可逆的 VLC 表使用  如果 DataPartEn 位是“0”, 该位忽略。	
2	ShortVideoHeader	读	0 - 正常MPEG4 流 1 - 短视频标头流	
3	H. 263 Annex J	读	0 - 附件J 关闭 1 - 附件 J 打开	

**CMD\_ENC\_SEQ\_BIT\_BUF\_START (0x180)**

位	名称	类型	功能	指令
31:0	BitBufAddr	读/写	位流缓冲器 SDRAM 字节地址  位流缓冲器必须是 512 字节对齐。  执行 ENC_SEQ_INIT 指令前, 主机必须写入该寄存器。	ENC_SEQ_INIT

**CMD\_ENC\_SEQ\_E (0x184)**

位	名称	类型	功能	指令
13:0	BitBufSize	读/写	位流缓冲器大小以千字节计数  在执行 NC_SEQ_INIT 指令前, 主机必须写入该寄存器。  最大的位流缓冲器大小是 4G 字节。	ENC_SEQ_INIT

**CMD\_ENC\_SEQ\_OPTION (0x188)**

	名称	类型	功能	指令
0	MbBitReport	读/写	<p>每个 MB 位的位置存储于 SDRAM 缓冲器</p> <p>如果该标志是“1”，处理器存储每个 MB 的起始位到 SDRAM。编码一张图片后，主机可能会访问该位的位置值。MB 的 BIT 缓冲器位于 [ParaBufAddr] 中。</p> <p>该位的位置从起始图片开始计数。</p>	ENC_SEQ_INIT
1	SliceInfoReport	读/写	<p>在 SDRAM 位缓冲器存储到 SDRAM 缓冲器，有效编码段数和位置。</p> <p>如果该标志是“1”，BIT 处理器存储每一段的编码段结束位置到 SDRAM。编码一张图片后，主机可能访问该段的位置。编码段数存储在 RET_ENC_PIC_SLICE_NUM 寄存器的 [EncSliceNum]。</p> <p>对于 H.263，附件 K 无效，该标志被忽略。因为没有附件 K 流的 H.263 没有段结构。</p>	
2	AUDEnable	读/写	<p>编码 H.264 访问单元分割符 RBSP 有效</p> <p>MPEG4/H.263 编码情况下，该标志被忽略。</p>	

**CMD\_ENC\_SEQ\_COD\_STD (0x18C)**

位	名称	类型	功能	指令
1:0	EncCodStd	读/写	<p>编码译码标准</p> <p>0: MPEG4 简单协议</p> <p>1: MPEG4 短视频标头 / H.263+</p> <p>2: H.264</p> <p>执行 SEQ_INIT 指令前，主机必须写入该寄存器。</p>	ENC_SEQ_INIT

**CMD\_ENC\_SEQ\_SRC\_SIZE (0x190)**

位	名称	类型	功能	指令
9:0	PictureHeight	读	编码源图片高度大小（以像素计）	ENC_SEQ_INIT

			源图片高度必须是 16 的倍数，小于或等于 576。	
19:10	PictureWidth	读	编码源图片宽度大小（以像素计） 源图片宽度必须是 16 的倍数，小于或等于 720。	

#### CMD\_ENC\_SEQ\_SRC\_F\_RATE (0x194)

位	名称	类型	功能	指令
15:0	FrameRateRes	读	编码源帧速余量 $\text{帧速} = [\text{FrameRateRes}] / [\text{FrameRateDiv}]$ 如果 $[\text{EncCodStd}] = 1$ ， 并且没有附件(I, J, K, T) 打开 (不带有 PLUSPTYPE的 H. 263)， 编码源帧速必须是 29.97 ( $[\text{FrameRateRes}] = 30000$ , $[\text{FrameRateDiv}] = 1001$ ) 因为H. 263 没有 PLUSPTYPE 支持，只有 29.97 Hz 源帧速	ENC_SEQ_INIT
31:16	FrameRateDivMinus1	读	编码源帧速单位数量，以 Hz-1 计 $[\text{FrameRateDiv}]$ 源于加该值到 1。	

#### CMD\_ENC\_SEQ\_MP4\_PARA (0x198)

位	名称	类型	功能	指令
0	DataPartEn	读/写	0 - 数据块无效 1 - 数据块有效	ENC_SEQ_INIT
1	RevVlcEn	读/写	0 - 正常 VLC 表使用 1 - 可逆 VLC 表使用 如果DataPartEn位是 ‘0’，该位被忽略。	
4:2	IntraDcVlcThr	读/写	MPEG4 内部 DC VLC 阈值码 允许的范围是[0~7]	

### CMD\_ENC\_SEQ\_263\_PARA (0x19C)

位	名称	类型	功能	指令
0	Annex T	读/写	0 - 附件T 关闭 1 - 附件 T 打开	ENC_SEQ_INIT
1	Annex K	读/写	0 - 附件 K 关闭 1 - 附件 K 打开	
2	Annex J	读/写	0 - 附件J 关闭 1 - 附件 J 打开	
3	Annex I	读/写	0 - 附件I 关闭 1 - 附件I 打开  * 对于编码模式，当前设计不支持附件 I。 因此该表示不许设置为 0。	

### CMD\_ENC\_SEQ\_AVC\_PARA (0x1A0)

位	名称	类型	功能	指令
4:0	ChromaQpOffset	读/写	图片参量 chroma_qp_index_offset 设置范围是-12 到 +12  2 的补码为有符号的 5 位  1_0100 : -12 1_0101 : -11  ...  0_1100 : +12	ENC_SEQ_INIT
5	ConstIntraFlag	读/写	图片参量 constrained_intra_pred_flag 设置  0 - 帧内预测使用内部 MB 数据 1 - 帧内预测不使用内部 MB 数据	
7:6	DisableDeblk	读/写	段标头的 disable_deblocking_filter_idc  0 - 解锁滤波器有效 1 - 解锁滤波器无效 2 - 除了段边界的解锁滤波器有效	

11:8	DeblkAlphaOffset	读/写	段标头 slice_alpha_c0_offset_div2 的范围是-6 到 +6 2 的有符号补码是 4 位	
15:12	DeblkBetaOffset	读/写	段标头 slice_beta_offset_div2 范围是-6 到 +6 2 的有符号补码是 4 位	

#### CMD\_ENC\_SEQ\_SLICE\_MODE (0x1A4)

位	名称	类型	功能	指令
0	SliceMode	读/写	0 - 每张图片一段 1 - 每张图片多段 MPEG4 模式，重置同步标志，并且信息包标头在段边界之间被插入 Annex K = 0 的H.263模式，GOB 被插入到每个 GOB层的开始 Annex K = 1的H.263模式，多段产生 H.264 模式，多段层 RBSP 产生	ENC_SEQ_INIT
1	SliceSizeMode	读/写	0 - 段被多码段位数改变 1 - 段被编码macro-block时钟数改变 SliceMode 设置为 0，该位被忽略。Annex K = 0 的 H.263 模式下，该位该位被忽略。	
15:2	SliceSizeNum	读/写	如果 SliceSizeMode 设置为 0，每段的 macro-block 数量必须设置到该寄存器。 如果 SliceSizeMode 设置为 1，每段编码位的计数必须设置到该寄存器。 如果 SliceMode 位是 0，该位被忽略。 Annex K = 0 的 H.263 模式下，该位忽略。	

**CMD\_ENC\_SEQ\_GOP\_NUM (0x1A8)**

位	名称	类型	功能	指令
5:0	EncGopNum	读/写	编码 GOP 数  I 图被插入在每个GOP图的数量  最大GOP 数 is 60.  0 - I, P, P, P, ... (只第一张图片是 I)  1 - I, I, I, ... (无 P 图)  2 - I, P, I, P, ...  3 - I, P, P, I, P, P, I, ...	ENC_SEQ_INIT

**CMD\_ENC\_SEQ\_RC\_PARA (0x1AC)**

位	名称	类型	功能	指令
0	RcEnable	读/写	读控制有效  如果该位设置为 0, 寄存器的值 PictureQs 在整个队列中作为量化间距。	ENC_SEQ_INIT
15:1	BitRate	读/写	目标比特率, 以千字节每秒计 (kbps)  如果 RcEnable = 0 该值被忽略。最大允许值是 32767 (0x7FFF)	
30:16	InitDelay	读/写	参考解码器初始缓冲器移动延迟, 以百万秒 (ms) 计  如果 RcEnable = 0, 该值被忽略  允许的最大值是 32767 (0x7FFF)	
31	SkipDisable	读/写	速度控制跳跃无效  如果该标志是 “0”, BIT 处理器可能跳跃一张图片, 如果对于供应来说当前位不足, 则该编入。  如果该标志是 “1”, BIT 处理器不跳跃过该图, 但是编码位流可能溢出目标比特率。  如果[RcEnable]是 “0”, 则该标志被忽略。	

**CMD\_ENC\_SEQ\_RC\_BUF\_SIZE (0x1B0)**

位	名称	类型	功能	指令
31:0	VbvBufSize	读/写	参考解码器缓冲器大小，以位计  如果 RcEnable = 0 或 InitDelay = 0 该值被忽略。  允许的最大值是 0x7FFF_FFFF  0 : 不检查参考解码器缓冲器大小限制	ENC_SEQ_INIT

**CMD\_ENC\_SEQ\_INTRA\_MB (0x1B4)**

位	名称	类型	功能	指令
15:0	IntraMbRefreshNum	读/写	内部 MB 刷新量必须少于被编码的 (PictureHeight×PictureWidth/)  0 - 不使用内部MB刷新  N - 每张图片至少 N 个 MBs 被编码做为内部模式	ENC_SEQ_INIT

**CMD\_ENC\_SEQ\_FMO (0x1B8)**

位	名称	类型	功能	指令
0	FmoEnable	读/写	0: FMO 无效  1: FMO 有效	ENC_SEQ_INIT
4:1	FmoSliceNr	读/写	段组的数量（必须是 2 到 8 之间的数）	
5	FmoType13	读/写	0: 类型0（交叉）  1: 类型 1（分散）	

**CMD\_ENC\_SEQ\_TMP\_BUF\_1 (0x1D0)**

位	名称	类型	功能	指令
31:0	TempBufAddr	读 / 写	临时缓冲器 SDRAM 字节地址  临时缓冲器必须是 256 字节对齐。  在执行 ENC_SEQ_INIT 指令之前，主机必须写入该寄存器。  临时缓冲器 1 被用做 ACDC 预测缓冲器（对于 Mpeg4）和内部预测 Y 缓冲器（对于 AVC）。	ENC_SEQ_INIT

			临时缓冲器 1 必须大于 $\text{picwidth} \times 8$ （对于 mpeg4）和 $\text{stride} \times \text{picheight} / 16$ （对于 AVC）。	
--	--	--	--	--

#### CMD\_ENC\_SEQ\_TMP\_BUF\_2 (0x1D4)

位	名称	类型	功能	指令
31:0	TempBufAddr	读/写	临时缓冲器 SDRAM 字节地址 临时缓冲器必须是 256 字节对齐。 执行 ENC_SEQ_INIT 指令前主机必须写入该寄存器。 临时缓冲器 2 用作数据分区部分 2 保存缓冲器（对于 Mpeg4）和内部预测 Cb 缓冲器（对于 AVC）。 临时缓冲器 2 必须比 $(\text{stride}/2) \times (\text{picheight}/2)/8$ （对于 AVC）大。 对于 mpeg4 情况，不能估计出临时缓冲器 2 大小。	ENC_SEQ_INIT

#### CMD\_ENC\_SEQ\_TMP\_BUF\_3 (0x1D8)

	名称	类型	功能	指令
31:0	TempBufAddr	读/写	临时缓冲器 SDRAM 字节地址 临时缓冲器必须是 256 字节对齐。 执行 ENC_SEQ_INIT 指令之前，主机必须写入该寄存器。 临时寄存器 3 用作数据分区部分 3 保存缓冲器（对于 Mpeg4）和内部预测 Cr 缓冲器（对于 AVC）。 临时缓冲器 3 必须比 $(\text{stride}/2) \times (\text{picheight}/2)/8$ （对于 AVC）大。 对于 mpeg4，不能估计出临时缓冲器 3 的大小。	ENC_SEQ_INIT

#### CMD\_ENC\_SEQ\_TMP\_BUF\_4 (0x1DC)

位	名称	类型	功能	指令
31:0	TempBufAddr	读 / 写	临时缓冲器 SDRAM 字节地址 临时缓冲器必须是 4 字节对齐。 执行 ENC_SEQ_INIT 指令前，主机必须写入该寄存器。 在 H.264 编码器中，当 FMO 选项有效时，临时缓冲器 4 被使用。该缓冲器	ENC_SEQ_INIT



			的大小通过 (32 KB × FmoSliceNr) 得出。例如，如果段组数是 8，该缓冲器的大小应该是 32x8 KB. 。	
--	--	--	---	--

**RET\_ENC\_SEQ\_SUCCESS (0x1C0)**

	名称	类型	功能	指令
0	RetStatus	读	0 - ENC_SEQ_INIT 指令执行错误 1 - ENC_SEQ_INIT 指令执行成功	ENC_SEQ_INIT

**CMD\_DEC\_PIC\_ROT\_MODE (0x180)**

位	名称	类型	功能	指令
3:0	PostRotMode	读/写	点旋转模式  PostRotMode[3:0] = {HorMir, VerMir, RotAng[1:0]}  HorMir : 水平镜像  VerMir : 垂直镜像  RotAng[1:0]  0 : 逆时针旋转0度  1 : 逆时针旋转90度  2 : 逆时针旋转180度  3 :逆时针旋转 270 度	DEC_PIC_RUN
4	PostRotEn	读/写	点旋转有效  如果该值是“1”，则旋转图像存储到DecPicRotAddrY, DecPicRotAddrCb, DecPicRotAddrCr 地址，用于将来参考。  如果该位是“0”，点旋转无效，并且 PostRotMode 被忽略。	

**CMD\_DEC\_PIC\_ROT\_ADDR\_Y (0x184)**

	名称	类型	功能	指令
31:0	DecRotAddrY	读/写	亮度旋转显示帧地址  如果 PostRotEn 域是“1”，旋转的图像存储到该地址。  在 VC—1 模式，不使用该地址。旋转输出将是 RET_DEC_SEQ_FRAME_NEED(0x1CC) 先前分派的一帧。	DEC_PIC_RUN

**CMD\_DEC\_PIC\_ROT\_ADDR\_CB (0x188)**

位	名称	类型	功能	指令
31:0	DecRotAddrCb	读 / 写	Cb 的旋转显示帧地址  在 VC—1 模式，不使用该寄存器。旋转输出先前 RET_DEC_SEQ_FRAME_NEED(0x1CC) 分派的一帧。	DEC_PIC_RUN

**CMD\_DEC\_PIC\_ROT\_ADDR\_CR (0x18C)**

位	名称	类型	功能	指令
31:0	DecRotAddrCr	读 / 写	Cr 的旋转显示帧地址  在 VC—1 模式，不使用该寄存器。旋转输出将是先前 RET_DEC_SEQ_FRAME_NEED(0x1CC) 分派的一帧。	DEC_PIC_RUN

**CMD\_DEC\_PIC\_DBK\_ADDR\_Y (0x190)**

	名称	类型	功能	指令
31:0	DecDbkAddrY	读/写	亮度解锁的显示帧地址  如果 Mp4DbkOn 域是“1”，解锁的图像存储在该地址。	DEC_PIC_RUN

**CMD\_DEC\_PIC\_DBK\_ADDR\_CB (0x194)**

位	名称	类型	功能	指令
31:0	DecDbkAddrCb	读/写	Cb 解锁显示帧地址	DEC_PIC_RUN

**CMD\_DEC\_PIC\_DBK\_ADDR\_CR (0x198)**

位	名称	类型	功能	指令
31:0	DecDbkAddrCr	读/写	Cr 解锁显示帧地址	DEC_PIC_RUN

**CMD\_DEC\_PIC\_ROT\_STRIDE (0x19C)**

位	名称	类型	功能	指令
10:0	DecRotStride	读/写	旋转显示帧幅度	DEC_PIC_RUN

**CMD\_DEC\_PIC\_CHUNK\_SIZE (0x1A8)**

位	名称	类型	功能	指令
31:0	DecPicChunkSize	读/写	图片流数据字节大小  该值只在文件演示模式有效，并且当 BIT 处理器更新图片流缓冲器写指针时用作数据流的大小。	DEC_PIC_RUN

**CMD\_DEC\_PIC\_BB\_START (0x1AC)**

位	名称	类型	功能	指令
31:0	DecPicBitBufStart	读/写	解码器输入如片流缓冲器的 4 字节对齐字节的地址  在文件演示模式下，如果解码器动态缓冲器分派选项有效，该值有效。在这种情况下，当 BIT 处理器更新图片流缓冲器的写指针，该地址将用作位流数据的其始地址。	DEC_PIC_RUN

**CMD\_DEC\_PIC\_START\_BYTE (0x1B0)**

位	名称	类型	功能	指令
1:0	DecPicValidByteStart	读/写	数据图片流缓冲器的有效位流的字节地址	DEC_PIC_RUN

**RET\_DEC\_PIC\_FRAME\_NUM (0x1C0)**

位	名称	类型	功能	指令
15:0	DecFrameNum	读/写	解码帧数  BIT 解码一帧，BIT 增加帧数和存储到该寄存器的帧数。	DEC_PIC_RUN

**RET\_DEC\_PIC\_IDX (0x1C4)**

位	名称	类型	功能	指令
15:0	DecPicIdx	读 / 写	显示帧索引  BIT 解码一帧之后，BIT 返回显示帧索引到该寄存器。  该帧索引是帧缓冲器地址数组的索引，是主机通过 SET_FRAME_BUF 指令发出的。  如果 BIT 返回-1 (0xFFFF)，它表示所有图片已经都被解码，如果主机给的一个位流包含 20 张图片，主机将从第 21 个 PICTURE_RUN 得到-1 值。  在文件演示模式下，当 BIT 没有图片要显示时，将返回-3 (0xFFFD)。例如，当 BIT 解码 reorderEn 位有效的 H.264 流，在解码最多 16 张图片前，BIT 没有图片来显示。并且当 BIT 解码 VC-1 流，解码 2 张图片前，BIT 没有图片来显示。	DEC_PIC_RUN

**RET\_DEC\_PIC\_ERR\_MB\_NUM (0x1C8)**

位	名称	类型	功能	指令
15:0	ErrMbNum	读/写	当前解码图片中错误的 MB 数  如果 BIT 识别出该流错误，BIT 在 MB 基础上执行错误隐蔽，在整个图片返回正确的 MB 数。  如果该值是“0”，解码的帧没有错误。	DEC_PIC_RUN

**RET\_DEC\_PIC\_TYPE (0x1CC)**

位	名称	类型	功能	指令
0	DecPicType	读/写	当前解码的图片的图片类型  0 - I (内部) 图片  1 - P (内部) 图片	DEC_PIC_RUN

**RET\_DEC\_PIC\_SUCCESS (0x1D8)**

位	名称	类型	功能	指令
0	RetStatus	读	0 - DEC_PIC_RUN 指令被执行，带有错误 1 - DEC_PIC_RUN 指令被成功执行	DEC_PIC_RUN
1	Invalid PPS	读	无效 PPS  当给出的流中没有有效的 PPS 该位被设置。  该位只在文件演示模式下被设置，并且当 RetStatus 位为 0 时有效。	

**CMD\_ENC\_PIC\_SRC\_ADDR\_Y (0x180)**

位	名称	类型	功能	指令
31:0	SrcAddrY	读/写	亮度编码源帧地址  执行 ENC_PIC_RUN 指令前主机必须写入该寄存器。  在每一张编码图片前，主机必须设置 SDRAM 帧缓冲器起始地址到该寄存器。	ENC_PIC_RUN

**CMD\_ENC\_PIC\_SRC\_ADDR\_CB (0x184)**

位	名称	类型	功能	指令
31:0	SrcAddrCb	读/写	Cb 的编码源帧地址	ENC_PIC_RUN

**CMD\_ENC\_PIC\_SRC\_ADDR\_CR (0x188)**

位	名称	类型	功能	指令
31:0	SrcAddrCr	读/写	Cr 的编码源帧地址	ENC_PIC_RUN

**CMD\_ENC\_PIC\_QS (0x18C)**

位	名称	类型	功能	指令
31:0	PictureQs	读/写	编码过程图片量化幅度参数  MPEG4/H. 263 模式，允许的范围是 1 到 31。  H. 264 模式，允许的范围是 0 到 51。	ENC_PIC_RUN

			如果速率控制有效，该寄存器被忽略。如果速率控制无效，BIT 在该值范围内，对当前图片编码整个 MB。	
--	--	--	--	--

**CMD\_ENC\_PIC\_ROT\_MODE (0x190)**

位	名称	类型	功能	指令
3:0	PreRotMode	读/写	预旋转模式  PreRotMode[3:0] = {HorMir, VerMir, RotAng[1:0]}  HorMir : 水平镜像  VerMir : 垂直镜像  RotAng[1:0]  0 : 逆时针旋转0度  1 : 逆时针旋转90度  2 : 逆时针旋转180度  3 : 逆时针旋转 270 度  如果该域是 4' b0000，预旋转无效。	ENC_PIC_RUN
4	PreRotEn	读/写	预旋转有效  如果该域是“1”，则源图像旋转优先于编码； 如果该域是“0”，则预旋转无效，并且 PreRotMode 被忽略。	

**CMD\_ENC\_PIC\_OPTION (0x194)**

位	名称	类型	功能	指令
0	PicSkipEn	读/写	图片跳跃标志  如果该域是“1”，EncSrcAddrY、EncSrcAddrCb、EncSrcAddrCr 被忽略，并且跳跃图片被编码。在这种情况下，重新组成的图片在解码中是先前图片的备份。不论[EncGopNum]是什么，该被跳过的图片被编码成 P 类型（内部）。	ENC_PIC_RUN

			<p>当没有得到下一个要被编码的源帧时，主机可能设置该域为“1”。</p> <p>例如，在编码帧速为 5 Hz 的情况下，如果在第四张图片照相机输出没有获得，主机必须在 1 秒内设置 ENC_PIC_RUN 指令 5 次，并且设置 PicSkipEn 标志为“0、0、0、1、0”</p>	
1	IdrPic	读/写	<p>如果该域是“1”，则不管 [EncGopNum] 的值，源图片被编码成 IDR IDR(Instantaneous Decoding Refresh) 图片 (H. 264) 或 I (内部) 图片 (MPEG-4/H. 263)。</p> <p>IDR 图是 0 frame_num 值的 I (内部) 图，所有的解码状态 (除了涉及到的图片列表) 被设置。</p> <p>编码 DIR 图片后，I 图周期计算被设置为初始状态。例如，如果主机设置 [IdrPic] 标志设置为第 18 帧，[EncGopNum] 是 15，编码图片类型是：</p> <p>第一帧：I (IDR - 自动)</p> <p>第二 帧：P</p> <p>...</p> <p>第十四帧：P</p> <p>第十五帧：I</p> <p>第十六帧：P</p> <p>第十七帧：P</p> <p>第十八帧：I (IDR - 主机设置)</p> <p>第十九帧：P</p> <p>...</p> <p>第三十二帧：I</p> <p>第三十三帧：P</p> <p>...</p> <p>在 MPEG-4/H. 263 情况下，I (内部) 图片对于解码器恢复是足够的。对于插入编码器刷新点，主机必须周期性地设置该区域为“1”。</p>	

**RET\_ENC\_PIC\_FRAME\_NUM (0x1C0)**

位	名称	类型	功能	指令
15:0	EncFrameNum	读/写	有效帧数  BIT 编码一帧后，BIT 增加帧数，并且将帧数存储到该寄存器。	ENC_PIC_RUN

**RET\_ENC\_PIC\_TYPE (0x1C4)**

位	名称	类型	功能	指令
0	EncPicType	读/写	当前有效的图片的图片类型  0 - I（内部）图片  1 - P（内部）图片	ENC_PIC_RUN

**RET\_ENC\_PIC\_IDX (0x1C8)**

位	名称	类型	功能	指令
15:0	RecPicIdx	读/写	重建的帧索引  BIT编码一帧后，BIT返回重建的帧索引到该寄存器。	ENC_PIC_RUN

**RET\_ENC\_PIC\_SLICE\_NUM (0x1CC)**

位	名称	类型	功能	指令
14:0	EncSliceNum	读/写	如果 CMD_ENC_SEQ_OPTION 寄存器的[SliceInfoReport]标记，则  BIT 返回编码段数到该寄存器。  每一段的编码段结束位置被存储到 SDRAM 中。	ENC_PIC_RUN

**CMD\_SET\_FRAME\_BUF\_NUM (0x180)**

位	名称	类型	功能	指令
4:0	FrameBufNum	读/写	用于参考或者数据再排序的帧数。  在解码条件下或者编码条件下是“2”，该值必须大于或等于 RET_DEC_SEQ_FRAME_NEED 寄存器的[FrameBufNeed]。  主机必须设置相关的帧缓冲器 SDRAM 的地址（Y，Cb，Cr）	SET_FRAME_BUF



			到地址[ParaBufAddr]的 SDRAM 缓冲器。	
--	--	--	------------------------------	--

### CMD\_SET\_FRAME\_BUF\_STRIDE (0x184)

位	名称	类型	功能	指令
10:0	LineStride	读/写	图片帧存储器的线幅度偏移幅度数以字节计数	SET_FRAME_BUF

### CMD\_ENC\_HEADER\_CODE (0x180)

位	名称	类型	功能	指令
2:0	HeaderCode	读/写	有效标头代码 MPEG4,  3' b000 - VOL 标头  3' b001 - VOS 标头  3' b010 - VIS 标头  H. 264,  3' b000 - SPS rbsp  3' b001 - PPS rbsp  H. 263,  ENC_HEADER 指令被忽略	ENC_HEADER

### CMD\_DEC\_PARAM\_SET\_TYPE (0x180)

位	名称	类型	功能	指令
0	DecParamSetType	读/写	参数设置类型  0 - 次序参数设置  1 - 图片参数设置	DEC_PARAM_SET

### CMD\_DEC\_PARAM\_SET\_SIZE (0x184)

位	名称	类型	功能	指令
8:0	DecParamSetSize	读/写	顺序/图片参数设置 Rbsp 字节大小  最大允许 Rbsp 大小为 511 字节。	DEC_PARAM_SET

**CMD\_ENC\_PARA\_SET\_TYPE (0x180)**

位	名称	类型	功能	指令
0	EncParaSetType	读/写	参数设置类型  0 - 次序参数设置  1 - 图片参数设置	ENC_PARA_SET

**RET\_ENC\_PARA\_SET\_SIZE (0x1C0)**

位	名称	类型	功能	指令
8:0	EncParaSetSize	读/写	有效顺序/图片参数设置 Rbsp 字节大小	ENC_PARA_SET

**21.13 BIT 处理器操作**

**21.13.1. BIT 处理寄存器描述**

根据主机接口寄存器的用途，可分成三类：

（1）BIT处理器控制寄存器

该种类的主机接口寄存器用于将更新或者显示BIT处理器状态送到主机处理器。在导入期间，大部分用于初始化BIT。

（2）BIT处理器通用寄存器

该类的主机接口寄存器用来存储所有的通用变量。一般地，所有的缓冲器地址和一些通用选项将安全地存储在这些寄存器中。

（3）BIT处理器指令I/O寄存器

只要新的指令从主机处理器发出，该类的主机接口寄存器将覆盖或更新。带有输入变元的所有指令和带有返回值所有相应的应答将被通过使用这些寄存器操纵。

地址0x000 ~ 0x0FC（64位寄存器地址空间）是H/W寄存器，该寄存器有复位值和确定的功能（不能配置）。地址0x100~0x1FC是通用S/W寄存器。它们没有复位值，可通过BIT固件配置。它们主要用于主机和BIT处理器之间的接口。

高32位寄存器（地址是0x100~0x17C）用于静态参量。对于所有种类的运行指令（EQ\_INIT, SEQ\_END,

PICTURE\_RUN, ...), 它们功能并不改变, 并且应用到整个指令和处理过程。低32位寄存器(地址是0x180~0x1FC)用做暂时指令变元。对于每条指令, 这些寄存器的功能或意义可能发生改变。

例如, BitStreamCtrl寄存器用于整个处理过程。意思是通过BitStreamCtrl寄存器影响到所有的编码/解码操作。但是CMD\_DEC\_SEQ\_STRIDE寄存器只应用到执行DEC\_SEQ\_INIT指令的当前处理过程中。

[LineStride]的值只用于当前处理, 因此所有的处理将有不同的线跳跃偏移, 因为BIT处理器在执行DEC\_SEQ\_INIT 期间读CMD\_DEC\_SEQ\_STRIDE寄存器, 和存储[LineStride]值到内部存储器。

## 21.13.2. BIT 处理器代码下载

BIT 处理器有 6144 个字的内部代码存储器。内部编码存储器用做通过 BIT 固件指示缓存控制。整个代码镜像可能大于 6144 字, 并且必须属于 SDRAM。运行时, BIT 固件从 SDRAM 载入合适的代码镜像。主机必须通过设置[CodeBufAddr]寄存器, 告知 BIT 处理器代码镜像的起始 SDRAM 字节地址。

例如, 两个处理过程(MPEG4 解码器, H.264 编码器)同时运行。如果主机在 DEC\_PIC\_RUN 指令后执行 ENC\_PIC\_RUN 指令, BIT 处理器需要 H.264 编码代码镜像来执行 H.264 编码一个图片, 因此, 它自动从 SDRAM (内容交换) 载入 H.264 编码代码镜像。对于初始 BIT 处理器执行, 主机必须直接下载初始导入代码镜像。通过设置[CodeRun]寄存器执行 BIT 处理器之前, 通过设置[CodeDownload]寄存器, 主机必须直接下载导入代码镜像(一些达到代码镜像的最上部)到最低代码存储器(地址 0)。

BIT 固件的当前版本的编码镜像总的字节数是 80 KB(40 个字), 引导代码镜像是 1024 字节(512 字)。如图 21-42 所示, 显示位处理器的代码下载。

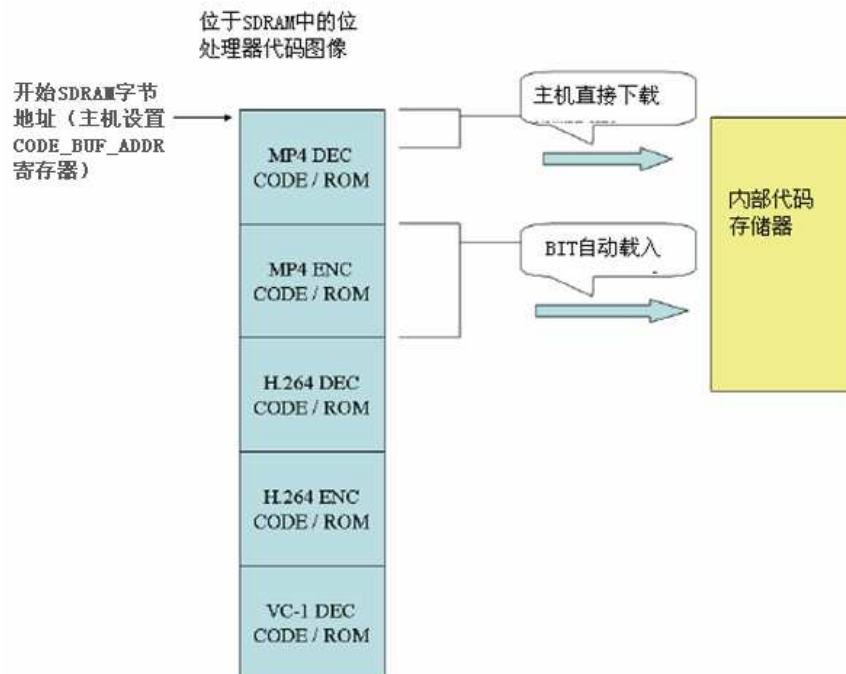


图 21-42 位处理器代码下载

### 21.13.3. 位流缓冲区管理

位于 SDRAM 的外部位流缓冲器由环形缓冲器组成。主机必须将环形缓冲器的起始地址和大小写入 BIT 处理器。

在解码的时候，主机写入要被解码的位流，接下来 BIT 处理器读取该位流。在这种情况下，位流覆盖或者向下溢出可能发生，解码将失败。为了防止覆盖或者向下溢出，当前位流读/写指针必须在主机和 BIT 处理器之间执行。BIT 处理器写环形缓冲器的当前读指针到 BitStreamRdPtr 寄存器，并且主机必须写环形缓冲器的当前写指针到 BitStreamWrPtr 寄存器。BIT 处理器通过比较当前的 BitStreamRdPtr 和 BitStreamWrPtr 来检测位缓冲区空（向下溢出）状态。如果没有得到更多要被解码的位流数据（缓冲区空状态），BIT 处理器停止位流解码来防止误读该位流，并且一直等到主机写入更多的位流数据，更新 BitStreamWrPtr。在写更多位流数据到环形缓冲区前，主机必须核对当前 BitStreamRdPtr 和 BitStreamWrPtr 来防止覆盖位流数据。

在编码时，BIT 处理器和主机改变卷轴，例如，BIT 处理器将位流写入，主机读取该编码的位流。因此 BIT 处理器将环形缓冲器的当前写指针写入 BitStreamWrPtr 寄存器，主机必须将环形缓冲器的当前读指针写入 BitStreamRdPtr 寄存器。如果环形缓冲器为空，BIT 处理器将等待编码位流数据。因此下一次写入将覆盖预存储的数据，主机还没有读取这些数据。如果环形缓冲器为空，主机将一直等到读取被编码的位流数据，因此下一次读取信息的时候，又读取了一次预读数据。

如果 BitStreamCtrl 寄存器中的[BufStsCheckDis]标志是“1”，BIT 处理器禁止位流的向上溢出/向下溢出检测选项。因此编码/解码因位流缓冲区状态的原因，操作不会停止。在这种情况下，主机处理器不需要将当前读/写指针写入 BitStreamRdPtr/BitStreamWrPtr 寄存器里，但是主机保证了不发生位流向上溢出/向下溢出。

BitStreamCtrl 寄存器的[BufPicReset]标志是“1”，在编码条件下，外部位流缓冲器不会作为一个环形缓冲器。被编码的位流将一直写到外部位流缓冲器的起始位。因此，ENC\_PIC\_RUN 指令完成，主机必须读一张图片的整个编码的位流。

如果BitStreamCtrl寄存器的[BufPicFlush]标志是“1”，在每次对一张图片编码结束时，一张图片所有被编码的位流被写入到外部位缓冲器。精确的字节地址被写入到BitStreamWrPtr寄存器中。

BIT处理器中也存在内部位流缓冲器，它是BIT处理器和SDRAM的外部位流缓冲器之间的高速缓冲存储器。在解码的情况下，BIT处理器从SDRAM的外部位流缓冲器读取位流数据，并且存储到内部位流缓冲器。为了提高效率，真正的分解操作是在内部缓冲器中执行的。在编码的时候，被编码的位流数据首先存储在内部缓冲器，然后成块地写入外部缓冲器。内部位流缓冲器有512个字节。因此，位流数据以512个字节从外部位缓冲器读取数据（在编码的情况下，写入数据）。BitStreamRdPtr或BitStreamWrPtr寄存器以512个字节一组增加，由于内部位流数据被读/写只以512个这些剩余字节做为一块。因此，编码一张图片结束时，并不是所有被编码的位流写入到外部位缓冲器。该图片最后编码的位流数据（少于512字节），有一定数量存在于内部位缓冲器，并且当聚集的被编码的数据超过512个字节，这些存在于内部缓冲区的数据被冲掉。为了得到这些剩余的被编码的数据，主机执行ENC\_SEQ\_END指令，BIT处理器将剩余的这些数据存储到外部位缓冲器。在ENC\_SEQ\_END指令之后，BitStreamWrPtr寄存器增长到确定的字节地址，主机将计算被编码的位流的精确的字节大小。

## 21.13.4. 运行指令的描述

指令变元寄存器必须通过主机执行该指令来设置。指令完成后，主机必须设置 BusyFlag 寄存器为“0”。

BIT 处理器接受新的指令，BusyFlag 寄存器是“1”，当指令完成，BusyFlag 设置为“0”。主机通过轮询 BusyFlag 寄存器或者通过 BIT 处理器中断来确认指令完成。

### **21.13.5. DEC\_SEQ\_INIT**

该指令开始一个解码过程。在BIT处理器执行DEC\_SEQ\_INIT指令期间，所有的DEC\_SEQ\_INIT指令变元寄存器的值存储到内部存储器，并且应用到当前处理过程中。因此，DEC\_SEQ\_INIT指令寄存器的值将在后面有指令跟随期间（例如，DEC\_PIC\_RUN指令）不会改变。

在 DEC\_SEQ\_INIT指令中，BIT处理器将找到顺序标头，分解标头来获得位流信息（如图片大小）。接下来将这些信息报告给DEC\_SEQ\_INIT返回寄存器。

### **21.13.6. ENC\_SEQ\_INIT**

该指令开始一个编码过程。在 ENC\_SEQ\_INIT 指令中，BIT 处理器从指令变元寄存器读取编码参数和编码顺序标头。存在于 ENC\_SEQ\_INIT 指令变元寄存器的编码参数被应用到后面的 ENC\_PIC\_RUN 指令，并且不改变。例如，CMD\_ENC\_SLICE\_MODE 寄存器的[SliceMode]应用于所有图片，但是CMD\_ENC\_PIC\_QS寄存器的[PictureQs]在每一条 ENC\_PIC\_RUN 指令该变。如果主机设置的编码参数无效，BIT 处理器将错误标志发送到 RET\_ENC\_SEQ\_SUCCESS 寄存器。

### **21.13.7. DEC\_SEQ\_END**

该指令结束解码过程。执行 DEC\_SEQ\_END 后，不再接收 DEC\_PIC\_RUN 指令。

### **21.13.8. ENC\_SEQ\_END**

该指令结束编码过程。如果[BufPicFlush]标志是“0”，并且[BufPicReset]标志是“0”，BIT 处理器将剩余的内部被编码的位流数据传输到外部位流数据缓冲器。再执行 ENC\_SEQ\_END 指令后，不再接收 ENC\_PIC\_RUN 指令。

### 21.13.9. DEC\_PIC\_RUN

该指令解码一张图片。解码图片后，BIT处理器向RET\_DEC\_PIC\_ERR\_MB\_NUM报告解码成功，并且在RET\_DEC\_PIC\_IDX寄存器显示帧缓冲器索引。

### 21.13.10. ENC\_PIC\_RUN

该指令编码一张图片。编码源帧缓冲器的SDRAM地址必须设置到指令变元寄存器。

### 21.13.11. SET\_FRAME\_BUF

该指令告知BIT处理器，帧缓冲器地址要被用来解码/重组图像。一共63个帧缓冲器可能要用到。通过RET\_DEC\_SEQ\_FRAME\_NEED r寄存器可获知用到的帧缓冲器的最小数量。在编码的情况下，两个帧缓冲器就够了。主机通过设置CMD\_SET\_FRAME\_BUF\_NUM寄存器来设置要用到的帧缓冲器的数量。

解码（解码条件下）/重组（编码条件下）图像应当保留以用作运动补偿参考，直到对于参考无用。因此该解码的/重组的帧缓冲器被再次用到。在编码/解码图片前，BIT处理器通过该指令保存整个帧缓冲器，然后为下一个存储区的编码/解码管理帧缓冲器的分配。

帧缓冲器地址存储到地址[ParaBufAddr]的 SDRAM。必须被存储每一帧亮度和两个色度缓冲器的地址索引。地址格式介绍如图 21-43 所示。

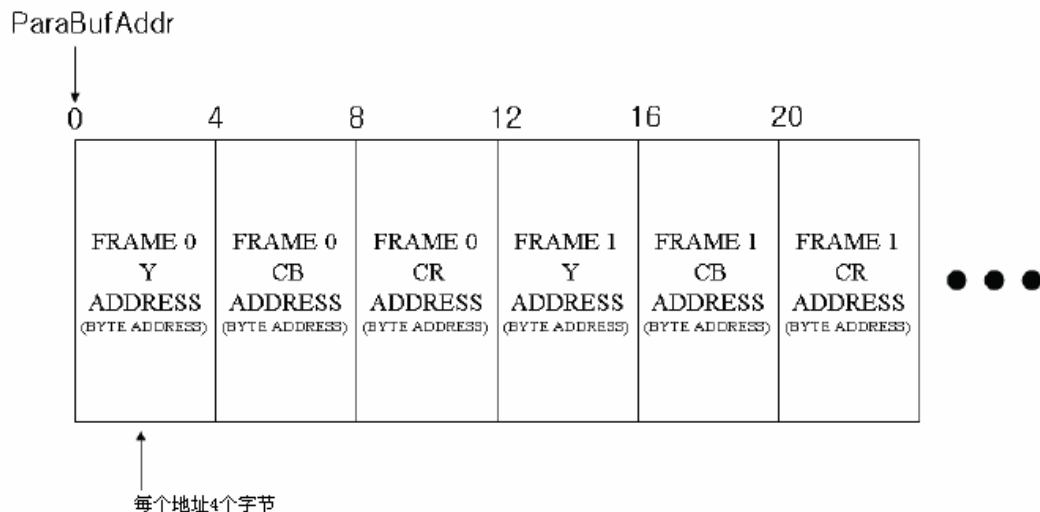


图 21-43 帧缓冲器地址格式

该指令编码特殊的表头。顺序标头必须在位流的起始位编码，在 ENC\_SEQ\_INIT 指令中，BIT 处理器编码顺序标头。在一些应用，如视频电话中，对于错误恢复或解码恢复，顺序标头必须传输。在该情况下，在 ENC\_PIC\_RUN 指令之间设置 ENC\_HEADER 指令来插入精确的标头。输入标头的标头编码必须设置到 CMD\_ENC\_HEADER\_CODE 寄存器中。

### 21. 13. 12. DEC\_PARA\_SET

在 H. 264 情况下，该指令用于将顺序参数设置或者图片参数设置传送到解码器上。

顺序参数设置或者图片参数设置可能通过“out-of-band”传递。在该情况下，主机必须通过该指令传输顺序参数设置或者图片参数设置到解码器。顺序参数设置或图片参数设置必须写入 BIT 处理器的参数缓冲器。BIT 处理器解码传输顺序/图片参数和存储解码内容。解码的顺序/图片参数设置在带有匹配的顺序/图片参数设置 id 的解码部分标头是有效的。

多顺序/图片参数设置可能传递到解码器。它们通过不同的顺序/图片设置 id 来区分。在顺序设置 RBSP 中，顺序参数设置 id 被编码以 5 位（0~31）。图片参数设置 id 以 8 位（0~255）编码。BIT 处理器能处理 32 个顺序参数设置和 256 个图片参数设置。被传递的顺序/图片参数设置的类型（顺序或者图片）和大小（以字节计数）必须通过指令变元寄存器传递到 BIT 处理器。



### 21.13.13. ENC\_PARA\_SET

在 H.264 情况下，该指令编码顺序参数设置或图片参数设置，并且通过参数缓冲器传递到主机。对于通过“out-of-band”传递的顺序/图片参数设置的应用，主机能通过该指令获得顺序/图片参数设置 RBSP。

BIT 处理器编码的顺序/图片参数设置和编码的 RBSP 流被存储到参数缓冲器，代替了正常的位流缓冲器。顺序/图片参数设置的编码的字节数量被指令返回寄存器返回。

#### (1) 参数缓冲器管理

参数缓冲器用于输入指令变元或者输出返回数据。参数缓冲器位于 SDRAM，开始字节地址必须设置到 ParaBufAddr 寄存器。

#### (2) 帧缓冲器地址

这个是输入指令变元，用于 SET\_FRAME\_BUF 指令。

#### (3) 编码的宏块位数

这是输出返回数据，用于 ENC\_PIC\_RUN 指令。如果 CMD\_ENC\_SEQ\_OPTION 寄存器的 [MbBitReport] 标志设置为“1”，ENC\_PIC\_RUN 指令完成后，BIT 处理器存储每一个宏块的起始位置到参数缓冲器。对于每一个宏块，开始位置以无符号的 16 位被存储。宏块的行幅度偏移是 128 个字节，因此最后  $128 - 720 / 16 \times 2 = 38$  字节不会用到。关于宏块位置的字节地址  $\langle \text{MbX}, \text{MbY} \rangle$  是  $\{\text{ParaBufAddr} + \text{MbY} \times 128 + \text{MbX} \times 2\}$ 。最大编码源图片高度是 576，所以宏块位数缓冲器的最大大小是  $128 \times 576 / 16 = 4608$  字节（4.5 KB）。详细格式如图 21-44 所示。

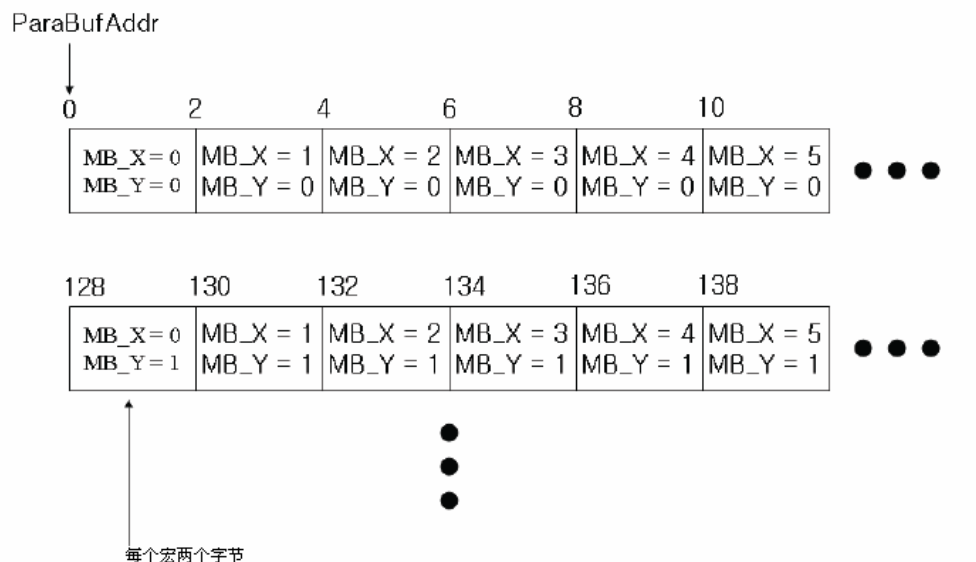


图 21-44 码的宏块位数

### 1. 编码的段的信息

这是输出返回数据，用于 ENC\_PIC\_RUN 指令。ENC\_PIC\_RUN 完成后，如果 CMD\_ENC\_SEQ\_OPTION 寄存器中的 [SliceInfoReport] 标志设置位“1”，BIT 处理器将每一个被编码的段的外部位流缓冲器的结束 SDRAM 地址存储到参数缓冲器。产生的段的总数存储到 RET\_ENC\_PIC\_SLICE\_NUM 寄存器，每一个段的结束位置的 SDRAM 地址存储到参数缓冲器。从每段的最后的位置，主机将计算每一段的精确的字节数。段位置缓冲器的起始地址是 {ParaBufAddr + 4608 (0x1200)}，仅在宏块位数缓冲器的下面。段的 SDRAM 地址结束位置是以无符号 32 位数表示。详细格式介绍如图 21-45 所示。

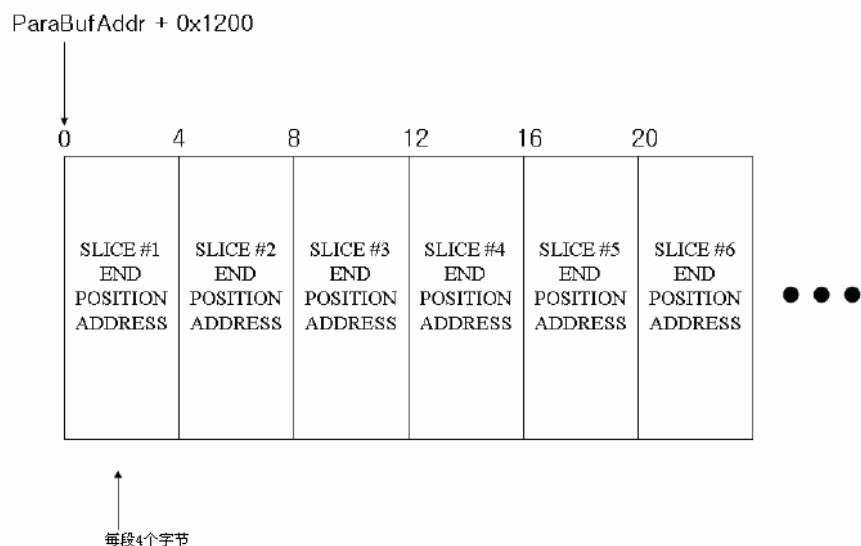


图 21-45 编码的段的信息

## 2. 顺序/图片参数设置RBSP

这是输入变元（对于DEC\_PARA\_SET指令）或者输出被编码的RBSP流（对于ENC\_PARA\_SET指令）。

## 3. 运转的缓冲器管理

对于编码/解码操作，该区域用于 SDRAM 内部运转的缓冲器。例如，用于 MPEG4 AC/DC 预测或者 H. 264 内部预测的重组的像素行缓冲器、用于运行多处理器的内容保存缓冲器、用于 MPEG4 数据分区的位流再排序缓冲器或者 H. 264 FMO/ASO 等等。例如，运转的缓冲器的大小是根据编码/解码大小和容量变化的。例如，AC/DC 预测缓冲区大小由图片宽度决定的，用于数据分区的最大位流再排序缓冲器由一张图片的最大位流大小决定。

固件的当前版本支持全 D1 大小图片（(720 x 576)）的图片解码和编码，位速率达到 10 Mb/s。在 720 x 576 情况下，如果运转的缓冲器配置选项被禁止，运转的缓冲器配置如表 21-19 所示。

表 21-19 运转的缓冲器配置

类型	名称	描述	大小（KB）
STATIC	STATIC_PRC_DMEN	用于内容交换的每个处理过程的BIT处理器数据存储器。	40 <sup>14</sup>
	STATIC_PRC_SEQ	每个队列的静态数据存储。	160 <sup>15</sup>
TEMP_PIC	MP4_DEC_ACDC	Y/Cb/Cr的AC/DC预测缓冲器。	picWidth×8

MP4_DEC	MP4_DEC_DP1	数据分区的位流再排序缓冲器。	48
	MP4_DEC_DP2	数据分区的位流再排序缓冲器。	48
TEMP_PIC 48	MP4_ENC_ACDC	Y/Cb/C的AC/DC预测缓冲器。	picWidth×8
MP4_ENC	MP4_ENC_DP1	数据分区的位流再排序缓冲器。	48
	MP4_ENC_DP2	数据分区的位流再排序缓冲器。	48
TEMP_PIC AVC_DEC	AVC_DEC_IP	Y/Cb/Cr的内部预测缓冲器。	72
	AVC_DEC_FMO	FMO组状态缓冲器。	5.5
	AVC_DEC_SLICE_INFO	段信息缓冲器 每张图片最多1620段。	12.66
	AVC_DEC_NAL_BUF	保存NAL单元缓冲器。	0.5
	AVC_DEC_SLICE	段数据RBSP缓冲器 一张图片的所有的段数据RBSP被存储。 最坏情况下，图片的原始数据大小。	XSIZE×YSIZE×1.5/1024
TEMP_PIC	VC1_DEC_ACDC	Y/Cb/Cr的AC/DC预测缓冲器	6
VC1_DEC	VC1_DEC_DEBLK	交叠/解块过滤器运转缓冲器。	8
	VC1_DEC_DIRECTMV	直接MV 运转缓冲器。	32
TEMP_PIC	AVC_ENC_IP	Y/Cb/Cr内部预测缓冲器。	72
AVC_ENC	AVC_ENC_FMO	FMO组状态缓冲器。	256

在整个处理/编解码过程，静态缓冲器被普遍使用，并且临时图片缓冲器被每个处理过程（编解码器）重用。

当使运转缓冲器配置选项有效，为了避免不可预料的故障，必须谨慎配置它。运转缓冲器能分成三个部分：固定静态缓冲器、可配置的静态缓冲器和可配置的临时缓冲器。固定静态缓冲器用于内容交换和静态数据存储，其大小是78 KB。可配置的静态缓冲器是在寄存器描述部分定义的处理缓冲器，在处于解码的AVC和VC-1情况下使用。每种情况都必须分派可配置的静态缓冲器，在每中情况之前不做其它应用。可配置的临时缓冲器是临时性的缓冲器，在寄存器描述部分定义了它，并且必须分派说明。一张图片处理结束并且被每中状况重用后，使用可配置的临时缓冲器。

例如，如果应用执行 MPEG4 解码的全 D1 处理，两个 AVC 解码、一个 VC-1 解码、一个 MPEG4 编和一个 AVC 编码同时进行，那么建议将运转缓冲器做如表 21-20 所示的配置。

表 21-20 运转缓冲器配置

类型	名称	描述	大小 (KB)
STATIC	STATIC_PRC_DMED	用于内容交换的每个处理的BIT处理器数据存储器。	40
	STATIC_PRC_SEQ	每个队列的静态数据存储	32
	STATIC_AVC_DEC_FMO	Dec FMO组缓冲器。	5.5
	STATIC_AVC_DEC_NAL_BUF	保存NAL单元缓冲器。	0.5
STATIC (configurable)	AVC_DEC_PS	PS数据保存缓冲器。	128
	VC1_DEC_DirectMV	直接MV预测缓冲器。	6.33
TEMP_PIC C MP4_DEC	MP4_DEC_ACDC	Y/Cb/Cr的AC/DC预测缓冲器。	picWidth×8
	MP4_DEC_DP1	用于数据分区的位流再排序缓冲器。	48
	MP4_DEC_DP2	用于数据分区的位流再排序缓冲器。	48
TEMP_PIC C MP4_ENC	MP4_ENC_ACDC	Y/Cb/Cr的AC/DC预测缓冲器。	picWidth×8
	MP4_ENC_DP1	用于数据分区的位流再排序缓冲器。	48
	MP4_ENC_DP2	用于数据分区的位流再排序缓冲器。	48
TEMP_PIC C AVC_DEC	AVC_DEC_IP_Y	Y的内部预测缓冲器。	$(\text{FrameBufferStride} \times \text{picHieght} / 16) / 1024$
	AVC_DEC_IP_Cb	Cb的内部预测缓冲器。	$\lceil \{ \text{FrameBufferStride} / 2 \times (\text{picHieght} / 2) \} / 8 \rceil / 1024$
	AVC_DEC_IP_Cr	Cr的内部预测缓冲器。	$\lceil \{ \text{FrameBufferStride} / 2 \times (\text{picHieght} / 2) \} / 8 \rceil / 1024$
	AVC_DEC_SLICE_INFO	段信息缓冲器 每张图片最大1620段。	12.66
	AVC_DEC_SLICE	段数据RBSP缓冲器	$\text{XSIZE} \times \text{YSIZE} \times 1.5 / 1024$

		存储一张图片的所有段数据。最坏情况下 图片原始数据的大小。	
TEMP_PIC	AVC_ENC_IP_Y	Y的内部预测缓冲器。	$(\text{FrameBufferStride} \times \text{picHieght}/16)/1024$
AVC_ENC	AVC_ENC_IP_Cb	Cb的内部预测缓冲器。	$[\{\text{FrameBufferStride}/2 \times (\text{picHieght}/2)\}/8]/1024$
	AVC_ENC_IP_Cr	Cr的内部预测缓冲器。	$[\{\text{FrameBufferStride}/2 \times (\text{picHieght}/2)\}/8]/1024$
	AVC_ENC_FMO	Enc FMO组缓冲器。	$32 \times \text{SliceGroupNum}$
TEMP_PIC	VC1_DEC_ACDC	Y/Cb/Cr的AC/DC预测缓冲器。	6
C VC1_DEC	VC1_DEC_DEBLK	交叠/解块过滤器运转缓冲器。	8

根据目标应用的要求，通过选择当前缓冲器的总线设置，应用软件能保留运转缓冲区空间。

运行过程的描述：

BIT处理器能同时执行最多8个进程。每个进程有不同的编解码标准（MPEG4 DECODE、H. 264 ENCODE等）。每个进程有不同的位流缓冲器和相关读/写指针寄存器（BitStreamRdPtr0、BitStreamWrPtr0等）。进程通过DEC\_SEQ\_INIT/ENC\_SEQ\_INIT指令产生，通过DEC\_SEQ\_END/ENC\_SEQ\_END指令结束。每个进程通过运行索引寄存器来鉴别。位处理器通过执行指令确定每个进程的时间。对于BIT处理器内容的交换，交换不同的处理需要一些周期/带宽的系统开销。在每个内容交换大约8 KB的存储数据被读出或写入SDRAM，如果编解码在前面的和当前的进程不相同，另外88 KB的存储代码被从SDRAM读出。因此，最坏的情况下，内容交换的开销花费的SDRAM带宽大约是 $\{8\text{KB}(\text{R}) + 8\text{KB}(\text{W}) + 8\text{KB}(\text{R})\} \times 30 \text{ Hz} \times 8 \text{ 进程} = 5760 \text{ KB} / \text{s}$  (30 帧/ s)。

对于以脉冲的形式读或写SDRAM数据，BIT处理器要求大约0.75周期/字节。因此最坏的情况下，关于系统开销，交换周期大约是 $\{8\text{K} + 8\text{K} + 8\text{K}\} \times 0.75 \times 30 \text{ Hz} \times 8 \text{ 进程} \sim 4.4 \text{ M 周期}$

#### 4. 预/后旋转的描述

编码源图像可能优先于编码处理（预旋转）被旋转，解码输出图像可能在解码处理后被旋转。在每次编码/解码一张图片时，主机告知预/后旋转模式。

在预旋转条件下，在读取源图像处理时，旋转被执行。因此不再需要SDRAM带宽。读取旋转源图像后，保持编码进程和非预旋转情况下相同。

在后旋转情况下，在存储解码图像处理时，旋转被执行。但是解码图像（优先于旋转）用于以后的帧解码，用作运动补偿参考。因此当后旋转有效，解码图像必须被存储，进一步需要SDRAM带宽用于额外的后旋转图像存储。。

预/后旋转模式由4位区域组成。第一位（最有意义的位）是水平镜像，下一位是垂直镜像，最后两位（不太重要的两位）是逆时针旋转角度。每个镜像/旋转域（水平、垂直、旋转）都独立应用。

在预旋转情况下，镜像域优先于旋转使用，接下来逆时针旋转应用到水平/垂直触发镜像。

在后旋转器中，旋转区域优先于水平/垂直镜像旋转，接下来镜像应用到逆时针旋转镜像。

如果逆时针旋转区域是 1（90 度）或者 3（270 度），旋转的镜像图片宽读/高度将被改变。例如，CIF 图像大小是（352 x 288），旋转后，将是 288 x 352，并且在后旋转的情况下，解码图像和旋转图像将存储为不同的格式。如图 21-46 所示，显示预/后旋转的描述图。

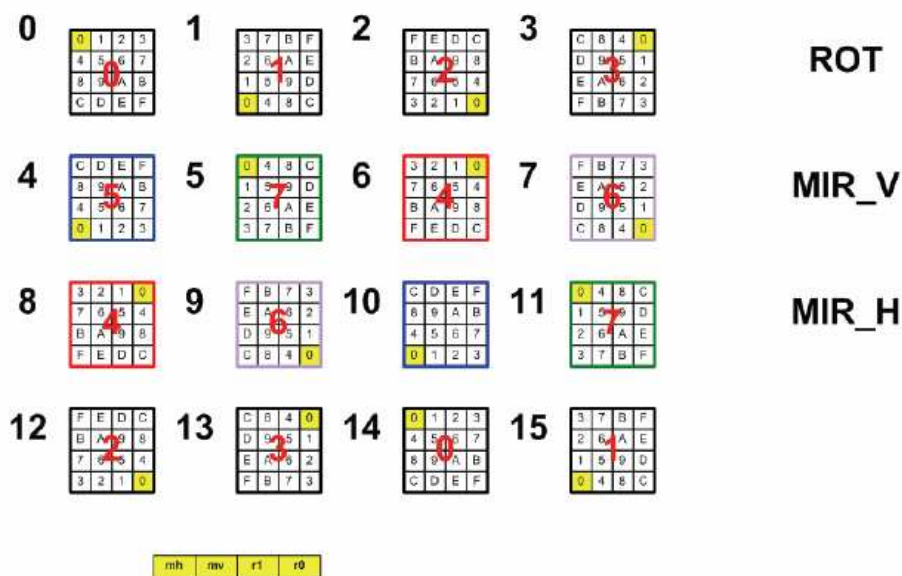


图21-46 预/后旋转的描述图

## 5. 操作流程的实例

对于执行 BIT 处理器的具体运行情况，如图 21-47 所示，显示流程是简单的例子。

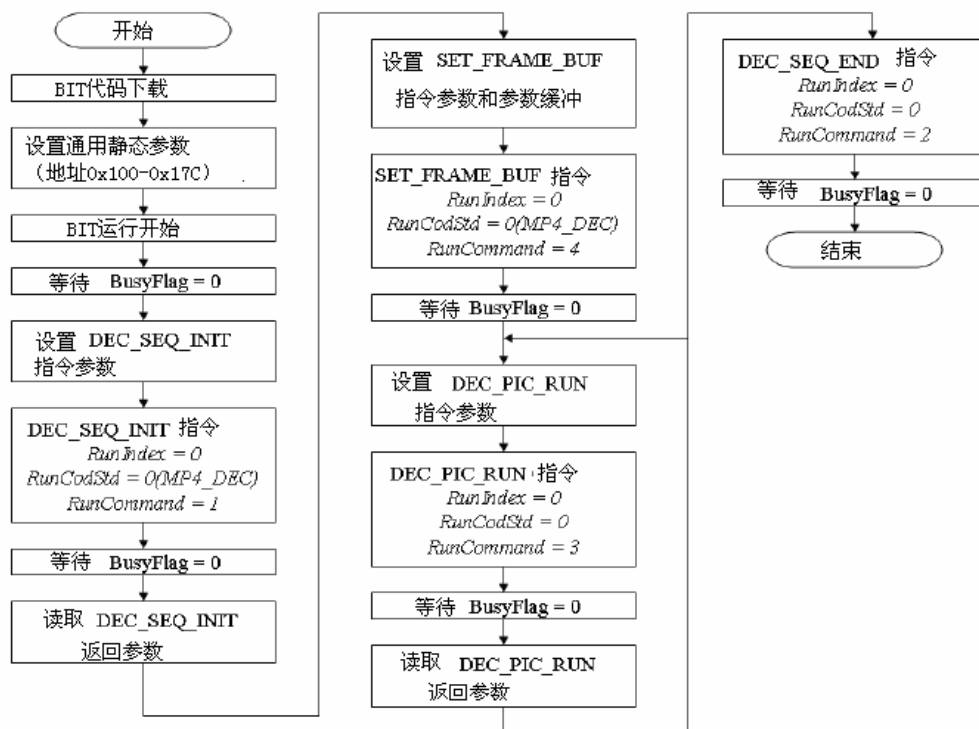


图 21-47 (a) 单个 MPEG4 解码器的例子



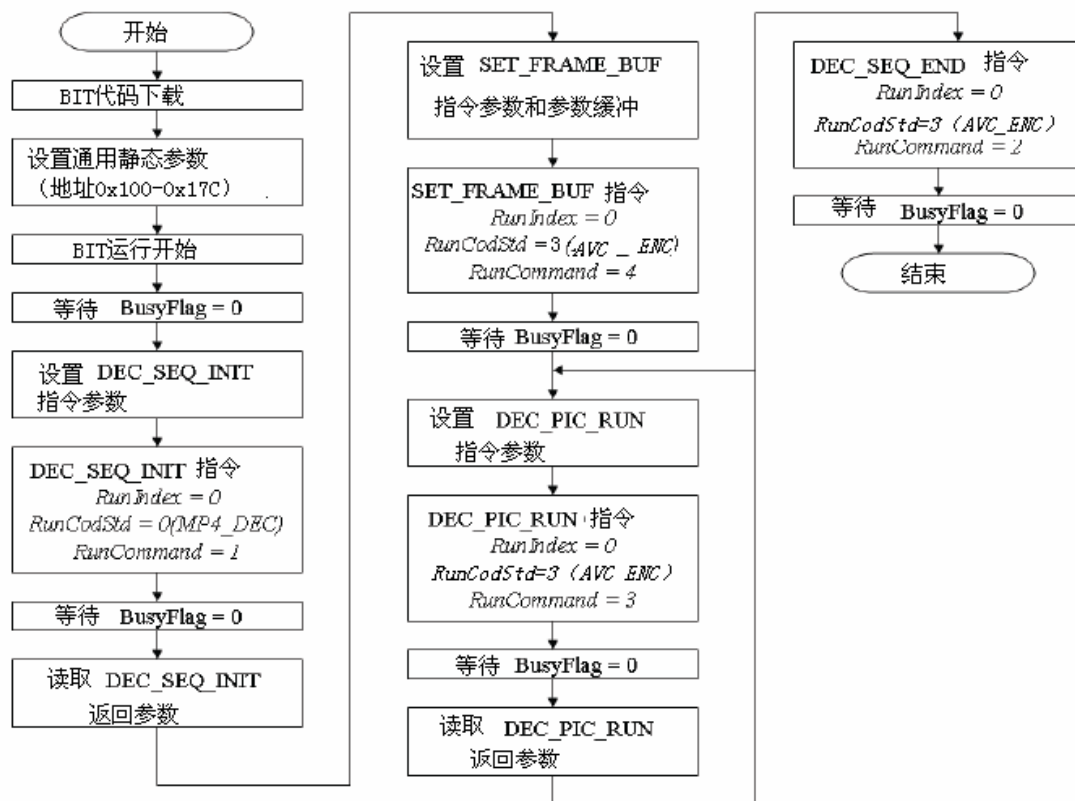


图21-47 (b) 单个H.264 编码器的例子

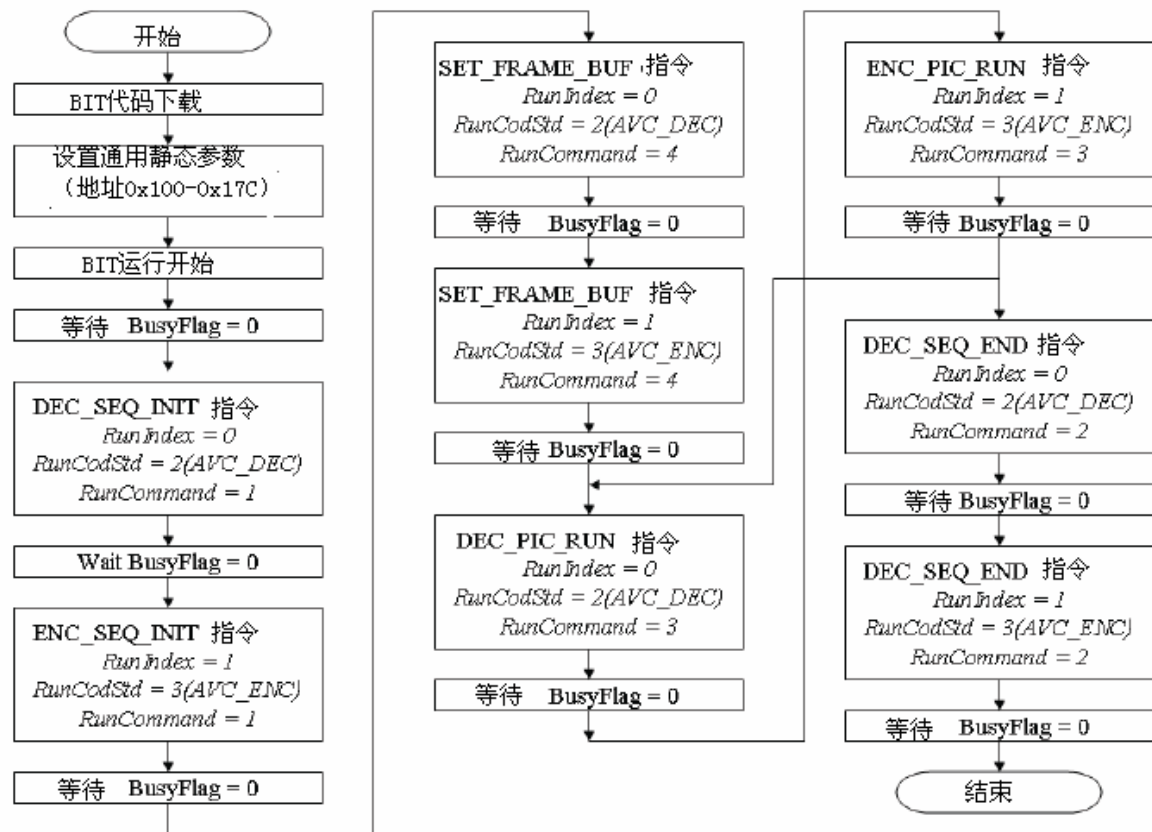


图 21-47 (c) 全双工的例子

### 单个 MPEG4 解码器的例子

以下位多格式视频编解码器的编码解码处理的初始化的一些代码，请大家参考。

多格式视频编解码器的编码处理初始化代码：

```

void MFC_InitProcessForDecoding(
    u32 uProcessIdx, MFC_CODEC_MODE eCodecMode, u32 uStreamBufStAddr, u32 uStreamBufSize,
    u32 uFrameBufStAddr, bool bDecRotEn, bool bMp4DeblkEn, bool bH264ReorderEn)
{
    u32 uStreamBufSizeCeilingToKbMultiple;
    u32 uMp4DecDeblkMode;
    u32 uH264DecReorderMode;

```

```
bool stat;
```

```
float frameRate;
```

```
u32 picX, picY;
```

```
u32 uNumOfRefReconFrame;
```

```
u32 uStride;
```

```
u32 uHeight;
```

```
u32 uFrameBufNumTemp;
```

```
u32 uFrameBufNum;
```

```
Assert(uProcessIdx < MAX_PROCESS_NUM);
```

```
Assert(eCodecMode == MP4_DEC || eCodecMode == AVC_DEC || eCodecMode == VC1_DEC);
```

```
oMfc.m_eCodecMode[uProcessIdx] = eCodecMode;
```

```
uStreamBufSizeCeilingToKbMultiple = (uStreamBufSize+1023)/1024*1024;
```

```
oMfc.m_uStreamBufStAddr[uProcessIdx] = uStreamBufStAddr;
```

```
oMfc.m_uStreamBufEndAddr[uProcessIdx] = uStreamBufStAddr +
```

```
uStreamBufSizeCeilingToKbMultiple;
```

```
oMfc.m_uStreamBufByteSize[uProcessIdx] = uStreamBufSizeCeilingToKbMultiple;
```

```
oMfc.m_uBitRdPtr[uProcessIdx] =
```

```
(uProcessIdx == 0) ? BITS_RD_PTR0 :
```

```
(uProcessIdx == 1) ? BITS_RD_PTR1 :
```

```
(uProcessIdx == 2) ? BITS_RD_PTR2 :
```

```
(uProcessIdx == 3) ? BITS_RD_PTR3 :
```

```
(uProcessIdx == 4) ? BITS_RD_PTR4 :
```

```
(uProcessIdx == 5) ? BITS_RD_PTR5 :
```

```
(uProcessIdx == 6) ? BITS_RD_PTR6 : BITS_RD_PTR7;
```

```
oMfc.m_uBitWrPtr[uProcessIdx] =
```

```

(uProcessIdx == 0) ? BITS_WR_PTR0 :
(uProcessIdx == 1) ? BITS_WR_PTR1 :
(uProcessIdx == 2) ? BITS_WR_PTR2 :
(uProcessIdx == 3) ? BITS_WR_PTR3 :
(uProcessIdx == 4) ? BITS_WR_PTR4 :
(uProcessIdx == 5) ? BITS_WR_PTR5 :
(uProcessIdx == 6) ? BITS_WR_PTR6 : BITS_WR_PTR7;

mfcOutp32(oMfc.m_uBitRdPtr[uProcessIdx], oMfc.m_uStreamBufStAddr[uProcessIdx]);
mfcOutp32(oMfc.m_uBitWrPtr[uProcessIdx],
oMfc.m_uStreamBufStAddr[uProcessIdx]+uStreamBufSize);

mfcOutp32(DEC_SEQ_BIT_BUF_ADDR, oMfc.m_uStreamBufStAddr[uProcessIdx]);
mfcOutp32(DEC_SEQ_BIT_BUF_SIZE, oMfc.m_uStreamBufByteSize[uProcessIdx]/1024);    // KB
unit

oMfc.m_bMp4DecDeblkMode[uProcessIdx] = (eCodecMode == MP4_DEC) ? bMp4DeblkEn : false;
uMp4DecDeblkMode    =    (oMfc.m_bMp4DecDeblkMode[uProcessIdx])    ?    MP4_DBK_ENABLE    :
MP4_DBK_DISABLE;
uH264DecReorderMode = (bH264ReorderEn) ? REORDER_ENABLE : REORDER_DISABLE;
mfcOutp32(DEC_SEQ_OPTION, uMp4DecDeblkMode|uH264DecReorderMode);

oMfc.m_bIsNoMoreStream[uProcessIdx] = false;

MFC_IssueCmd(uProcessIdx, SEQ_INIT);

stat = MFC_IsCmdFinished();

if(stat == false)

```

```

{
    Disp("\n There is an error in the SEQ_INIT result\n");
    return;
}

//u32 uFrameBufAddr = oMfc.m_uStreamBufEndAddr[uProcessIdx]+STREAM_WR_SIZE;

MFC_GetDecSrcFormat(&picX, &picY, &frameRate);
oMfc.m_uPicX[uProcessIdx] = picX;
oMfc.m_uPicY[uProcessIdx] = picY;
Disp("%d x %d @%.2f Hz\n", picX, picY, frameRate);
Assert(picX > 0);
Assert(picY > 0);

oMfc.m_bDecRotEn[uProcessIdx] = bDecRotEn;
oMfc.m_uRotFrameIdx[uProcessIdx] = 0;
oMfc.m_uMp4DeblockFrameIdx[uProcessIdx] = 0;
oMfc.m_uFrameIndex[uProcessIdx] = 0;

// H.263 Annex J deblock is in-loop filter, otherwise deblock is out-loop filter
MFC_IsDecH263AnnexJOn(uProcessIdx, &oMfc.m_bAnnexJOn[uProcessIdx]);

oMfc.m_uFrameDelayCount[uProcessIdx] = mfcInp32(RET_DEC_SEQ_FRAME_DELAY);
//DbgMfc("Delay Frame num=%d\n", oMfc.m_uFrameDelayCount[uProcessIdx]);

MFC_GetDecRefFrameNum(uProcessIdx, &uNumOfRefReconFrame);
//DbgMfc("num of RefFrame = %d\n", uNumOfRefReconFrame);

uFrameBufNumTemp    =    (oMfc.m_bDecRotEn[uProcessIdx])    ?    uNumOfRefReconFrame+2    :

```

```

uNumOfRefReconFrame;

    uFrameBufNum = (oMfc.m_bMp4DecDeblkMode[uProcessIdx] && !oMfc.m_bAnnexJ0n[uProcessIdx]) ?
uFrameBufNumTemp+2 : uFrameBufNumTemp;

    uStride = (picX%16 == 0) ? picX : (picX+15)/16*16;
    uHeight = (picY%16 == 0) ? picY : (picY+15)/16*16;
    MFC_InitDecFrameBuffer(uProcessIdx, uFrameBufNum, uStride, uHeight, uFrameBufStAddr);

    MFC_IssueCmdOfSetFrameBuffer(uProcessIdx, uNumOfRefReconFrame, uStride);
}

```

多格式视频编解码器的解码处理初始化代码:

```

void MFC_InitProcessForDecoding(
    u32 uProcessIdx, MFC_CODEC_MODE eCodecMode, u32 uStreamBufStAddr, u32 uStreamBufSize,
    u32 uFrameBufStAddr, bool bDecRotEn, bool bMp4DeblkEn, bool bH264ReorderEn)
{
    u32 uStreamBufSizeCeilingToKbMultiple;
    u32 uMp4DecDeblkMode;
    u32 uH264DecReorderMode;
    bool stat;
    float frameRate;
    u32 picX, picY;
    u32 uNumOfRefReconFrame;
    u32 uStride;
    u32 uHeight;
    u32 uFrameBufNumTemp;
    u32 uFrameBufNum;

    Assert(uProcessIdx < MAX_PROCESS_NUM);
    Assert(eCodecMode == MP4_DEC || eCodecMode == AVC_DEC || eCodecMode == VC1_DEC);
}

```

```

oMfc.m_eCodecMode[uProcessIdx] = eCodecMode;
uStreamBufSizeCeilingToKbMultiple = (uStreamBufSize+1023)/1024*1024;

oMfc.m_uStreamBufStAddr[uProcessIdx] = uStreamBufStAddr;
oMfc.m_uStreamBufEndAddr[uProcessIdx] = uStreamBufStAddr +
uStreamBufSizeCeilingToKbMultiple;
oMfc.m_uStreamBufByteSize[uProcessIdx] = uStreamBufSizeCeilingToKbMultiple;

oMfc.m_uBitRdPtr[uProcessIdx] =
    (uProcessIdx == 0) ? BITS_RD_PTR0 :
    (uProcessIdx == 1) ? BITS_RD_PTR1 :
    (uProcessIdx == 2) ? BITS_RD_PTR2 :
    (uProcessIdx == 3) ? BITS_RD_PTR3 :
    (uProcessIdx == 4) ? BITS_RD_PTR4 :
    (uProcessIdx == 5) ? BITS_RD_PTR5 :
    (uProcessIdx == 6) ? BITS_RD_PTR6 : BITS_RD_PTR7;
oMfc.m_uBitWrPtr[uProcessIdx] =
    (uProcessIdx == 0) ? BITS_WR_PTR0 :
    (uProcessIdx == 1) ? BITS_WR_PTR1 :
    (uProcessIdx == 2) ? BITS_WR_PTR2 :
    (uProcessIdx == 3) ? BITS_WR_PTR3 :
    (uProcessIdx == 4) ? BITS_WR_PTR4 :
    (uProcessIdx == 5) ? BITS_WR_PTR5 :
    (uProcessIdx == 6) ? BITS_WR_PTR6 : BITS_WR_PTR7;

mfcOutp32(oMfc.m_uBitRdPtr[uProcessIdx], oMfc.m_uStreamBufStAddr[uProcessIdx]);
mfcOutp32(oMfc.m_uBitWrPtr[uProcessIdx],
oMfc.m_uStreamBufStAddr[uProcessIdx]+uStreamBufSize);

```

```

mfcOutp32(DEC_SEQ_BIT_BUF_ADDR, oMfc.m_uStreamBufStAddr[uProcessIdx]);
mfcOutp32(DEC_SEQ_BIT_BUF_SIZE, oMfc.m_uStreamBufByteSize[uProcessIdx]/1024);    // KB
unit

oMfc.m_bMp4DecDeblkMode[uProcessIdx] = (eCodecMode == MP4_DEC) ? bMp4DeblkEn : false;
uMp4DecDeblkMode    =    (oMfc.m_bMp4DecDeblkMode[uProcessIdx])    ?    MP4_DBK_ENABLE    :
MP4_DBK_DISABLE;
uH264DecReorderMode = (bH264ReorderEn) ? REORDER_ENABLE : REORDER_DISABLE;
mfcOutp32(DEC_SEQ_OPTION, uMp4DecDeblkMode|uH264DecReorderMode);

oMfc.m_bIsNoMoreStream[uProcessIdx] = false;

MFC_IssueCmd(uProcessIdx, SEQ_INIT);

stat = MFC_IsCmdFinished();

if(stat == false)
{
    Disp("\n There is an error in the SEQ_INIT result\n");
    return;
}

MFC_GetDecSrcFormat(&picX, &picY, &frameRate);
oMfc.m_uPicX[uProcessIdx] = picX;
oMfc.m_uPicY[uProcessIdx] = picY;
Disp("%d x %d @%.2f Hz\n", picX, picY, frameRate);
Assert(picX > 0);
Assert(picY > 0);

```



```

oMfc.m_bDecRotEn[uProcessIdx] = bDecRotEn;
oMfc.m_uRotFrameIdx[uProcessIdx] = 0;
oMfc.m_uMp4DeblockFrameIdx[uProcessIdx] = 0;
oMfc.m_uFrameIndex[uProcessIdx] = 0;

MFC_IsDecH263AnnexJOn(uProcessIdx, &oMfc.m_bAnnexJOn[uProcessIdx]);

oMfc.m_uFrameDelayCount[uProcessIdx] = mfcInp32(RET_DEC_SEQ_FRAME_DELAY);

MFC_GetDecRefFrameNum(uProcessIdx, &uNumOfRefReconFrame);

uFrameBufNumTemp    =    (oMfc.m_bDecRotEn[uProcessIdx])    ?    uNumOfRefReconFrame+2    :
uNumOfRefReconFrame;
uFrameBufNum = (oMfc.m_bMp4DecDeblkMode[uProcessIdx]
&& !oMfc.m_bAnnexJOn[uProcessIdx]) ? uFrameBufNumTemp+2 : uFrameBufNumTemp;

uStride = (picX%16 == 0) ? picX : (picX+15)/16*16;
uHeight = (picY%16 == 0) ? picY : (picY+15)/16*16;
MFC_InitDecFrameBuffer(uProcessIdx, uFrameBufNum, uStride, uHeight, uFrameBufStAddr);

MFC_IssueCmdOfSetFrameBuffer(uProcessIdx, uNumOfRefReconFrame, uStride);
}

```

## 22 JPEG 编解码器

JPEG编解码器的核心是由控制电路，DCT/量化，哈夫曼编码，标志处理块和AHB从接口控制组成的，如图22-1所示。输入/输出图像总线和压缩数据总线是8位，它控制内部的寄存器。

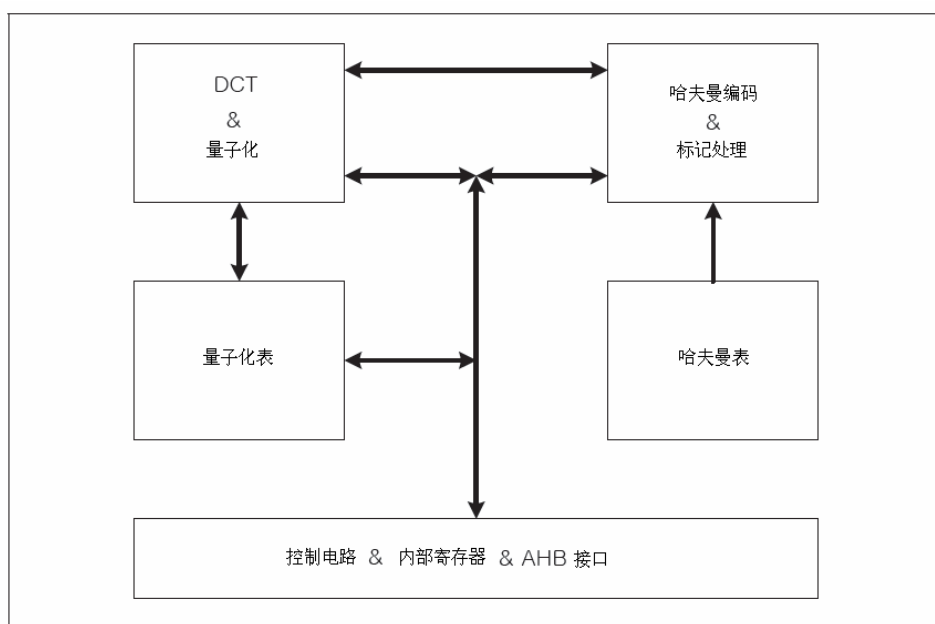


图 22-1 JPEG 编解码器结构框图

### 22.1 JPEG 编码特性

JPEG 编解码器包含有以下特性：

- 压缩/解压缩高达UXGA。
- 编码格式：YCbCr4:2:2或YCbCr4:2:0(JPEG引擎的输入格式)。
- 解码格式：YCbCr4:4:4, YCbCr4:2:2, YCbCr4:2:0, YCbCr4:1:1或格雷码。
- 支持直接压缩，从相机输出。
- 在YCbCr4:2:2或RGB565格式中，支持内存数据的压缩。
- 支持通用颜色转换器。

## 22.2 JPEG 编码定义

### 1. 控制电路和 AHB 接口

该模块设置和初始化操作模式，由内置寄存器组成。它设置操作模式，来确定量化哈夫曼表数目和DRI值。

### 2. DCT/量化

在编码期间，JPEG 编解码器将  $8 \times 8$  图像数据转换为 DCT 系数。因此，量化处理是通过利用量化表来执行 DCT 系数的。在解码期间，反量化将完成，然后 DCT 系数转化成图像数据。

### 3. 哈夫曼编码和标记处理

各种长度的编码和解码都是基于哈夫曼表的。

### 4. 量化表

主要用来存储量化表。这是 RAM 区域，用户可以对其进行分配。

### 5. 哈夫曼表

主要用来存储哈夫曼表。这是RAM区域，用户可以对其进行分配。

### 6. 寄存器访问

该寄存器被修改：

- 复位后，直到一个新的工作开始。
- 处理完成后，中断信号产生，直到新的工作开始。

其他条件表明，核心是在正常运作，因此不允许修改。对于一些寄存器，无论是写或读都可能被禁止。

### 7. 表访问

在压缩前，必须对四个哈夫曼表（AC&DC，每 2 个表）和四个量化表进行配置。设置任何量化表和哈夫曼表，必须首先访问相应的入口寄存器。因此，必须有写传输脉冲跟随。为了更好地了解写传输脉冲，参看图 22-2。每个表的访问顺序显示如下：

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	29	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

图 22-2 量化表的访问顺序

## 8. 中断信号

中断信号将在以下条件下产生（寄存器 JPGIRQ，用来确定原因）：

- 用于一帧的压缩或解压缩处理完成。
- 在解压缩期间，JPGIRQ[3]被设置为高电平。在标记分析之后，寄存器保存图像的大小和取样因子被读出。

中断 1，正常的处理完成。清除待处理的中断请求，读 JPGSTS 寄存器。如果没有编码或解码的错误，JPGIRQ 将以 0x40 被读出。

取消中断 2 也是通过读 JPGIRQ 来完成的。如果没有标题解析错误，其将以 0x08 读出。中断 2 表示解压缩处理被终止。

中断 3 信号用于区别中断发生的条件。

## 9. 中断设置寄存器

当准备解压缩一个图像时，该寄存器用来设置是否允许中断。为了允许中断，在开始解压缩处理之前，设置 JPGIRQS[3]为高电平。当该中断发生时，JPEG 编解码器终止处理，同时驱动 PGSTS[0]为高电平。通过读取 JPGIRQ，设置该寄存器来取消中断。

10. 标记处理

标记是在压缩过程中产生的。如表 22-1 所示。

表 22-1 JPEG 编解码器的标记

标记	编解码器（Hex）	描述
SOI	FFD8	图像开始
SOF0	FFC0	基线 DCT
SOS	FFDA	扫描开始
DQT	FFDB	定义量化表
DHT	FFC4	定义哈夫曼表
DRI	FFDD	重新界定区间
RSTm	FFD0～FFD7	重新开始以模块 8 计数 “m”
EOI	FFD9	图像结束

在解压缩期间，表22-1中的标记得处理。除了SOF1～SOF7和JPG，其他标记将被忽略。

11. 压缩文件的位流

创造JPEG的位流显示如图22-3所示。



图 22-3 JPEG 文件的位流结构框图

12. 程序员模型

如表 22-2 所示，显示了寄存器编码及解码过程。

表 22-2 寄存器编码及解码过程

寄存器	描述	编码过程	解码过程
JPGMOD	流程模式寄存器	必要的	必要的
JPGQHN0	量化和哈夫曼表数目寄存器	必要的	--
JPGDRI	重置区间寄存器	必要的	--
JPGY	垂直大小寄存器	必要的	
JPGX	水平大小寄存器	必要的	

QTBL0	量化表0入口寄存器	必要的	--
QTBL1	量化表1入口寄存器	必要的	--
QTBL2	量化表2入口寄存器	必要的	--
QTBL3	量化表3入口寄存器	必要的	--
HDTBL0, HDCTBLG0	直流哈夫曼表0入口寄存器	必要的	--
HACTBL0, HACTBLG0	交流哈夫曼表0入口寄存器	必要的	--
HDCTBL1, HDCTBLG1	直流哈夫曼表1入口寄存器	必要的	--
HACTBL1, HACTBLG1	交流哈夫曼表1入口寄存器	必要的	--

表 22-2 中任意寄存器的内容都不会改变，除非重新写入或被重置。因此，只有通过执行开始命令过程，才有可能处理下一帧。通过在 SW\_JSTART 寄存器中写 0x1，重新开始。

### 13. JPEG 编解码器设计向导

JPEG 引擎有它自己的内部等待标记，它是通过软件读取 JPGIRQ 来清除的。如果所有的处理都完成了，读 JPGSTS 以清除所有内部中断的等待标记。在 JPEG 处理和硬件控制解码模式下，JPEG 引擎自动地清除所有等待标记。

基本的 JPEG 编码顺序，如图 22-4 所示。

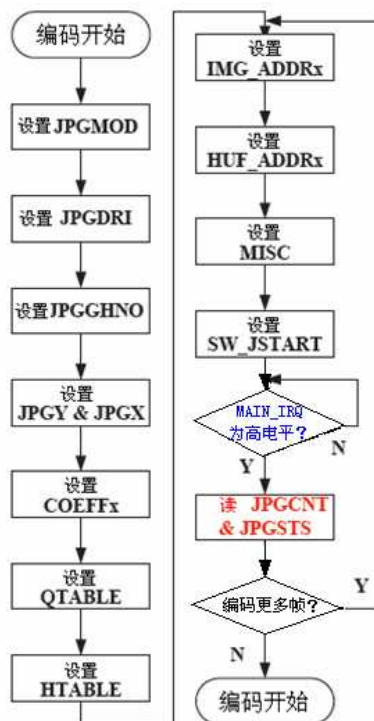


图 22-4 基本编码的流程图实例

基本的 JPEG 编码采取下列步骤：

- (1) 在 JPGMOD 寄存器中，设置流程模式为编码流程和亚抽样模式。
- (2) 设置 MCU 和 RST 标记寄存器 JPGDRI。
- (3) 设置 Q 和 H 表数目寄存器 JPGGHNO。
- (4) 设置 JPGY 和 JPGX 寄存器。
- (5) 设置系数寄存器 COEFF1, COEFF2, COEFF3，用于色彩空间转换。
- (6) 设置 QTABLE 和 HTABLE。
- (7) 设置源图像数据，第一帧地址寄存器 IMG\_ADDR0 和第二帧的 IMG\_ADDR1。
- (8) 设置 JPEG 目的文件地址寄存器 HUFADDR0 和下一编码 JPEG 文件地址寄存器 HUFADDR1。
- (9) 设置混合寄存器 MISC（设置 EMS 位为 0，选择 MODE\_SEL 0x1 或 0x2）。
- (10) 设置 SW\_JSTART 为高电平。
- (11) 必须读 JPGIRQ 和 JPGSTS 寄存器，以清除内部等待中断。

代码实现：

具体在 ARM11 中 JPEG 基本的编码，代码编写如下：

```

void JPEG_InitRegsForEncoding(
    u32 uRawHsz, u32 uRawVsz, u32 uRawAddr, CSPACE eRawType,
    u32 uJpgAddr, JPEG_TYPE uJpgType, bool bIsOnTheFly, bool bIsMotion)
{
    int i;
    Assert(eRawType == YCBYCR || eRawType == RGB16);
    Outp32(JPGMOD, (uJpgType == JPEG_422)? (0x1<<0) : (0x2<<0)); // Encoded to yuv422 or yuv420
    Outp32(JPGDRI, 2); // MCU inserts RST marker
    Outp32(JPGQHN0, 0x0);
    Outp32(JPGX, uRawHsz);
    Outp32(JPGY, uRawVsz);
    Outp32(JIMGADDR0, uRawAddr); // Address of input image
    Outp32(JHUFADDR0, uJpgAddr); // Address of JPEG stream
    Outp32(JIMGADDR1, uRawAddr); // Address of input image
    Outp32(JHUFADDR1, uJpgAddr); // next address of motion JPEG stream
    Outp32(JCOEF1, COEF1_RGB_2_YUV); // Coefficient value 1 for RGB to YCbCr
    Outp32(JCOEF2, COEF2_RGB_2_YUV); // Coefficient value 2 for RGB to YCbCr
    Outp32(JCOEF3, COEF3_RGB_2_YUV); // Coefficient value 3 for RGB to YCbCr
    Outp32(JMISC,
        (bIsOnTheFly ? 0 : (eRawType == YCBYCR ? 1 : 2))<<5 |
        (bIsOnTheFly ? 1 : 0)<<2
    );
    Outp32(JPG_CON, (bIsMotion ? ENABLE_MOTION_ENC : DISABLE_MOTION_ENC));

    // Outp32(JPGIRQEN, 0x10);
    // Outp32(JPGIRQEN, 1<<4); // Deleted @2006.6.8
    // Quantiazation and Huffman Table setting
    //-----

```



```

for (i=0; i<64; i++)
    Outp32((JQTBL0+i*4), (u32)QTBL0[i]);
for (i=0; i<64; i++)
    Outp32((JQTBL1+i*4), (u32)std_chrominance_quant_tbl_plus[i]);
for (i=0; i<16; i++)
    Outp32((JHDCTBL0+i*4), (u32)HDCTBL0[i]);
for (i=0; i<12; i++)
    Outp32((JHDCTBLG0+i*4), (u32)HDCTBLG0[i]);
for (i=0; i<16; i++)
    Outp32((JHACTBL0+i*4), (u32)HACTBL0[i]);
for (i=0; i<162; i++)
    Outp32((JHACTBLG0+i*4), (u32)HACTBLG0[i]);
}

```

JPEG 解码顺序：软件控制解码。如图 22-5 所示。

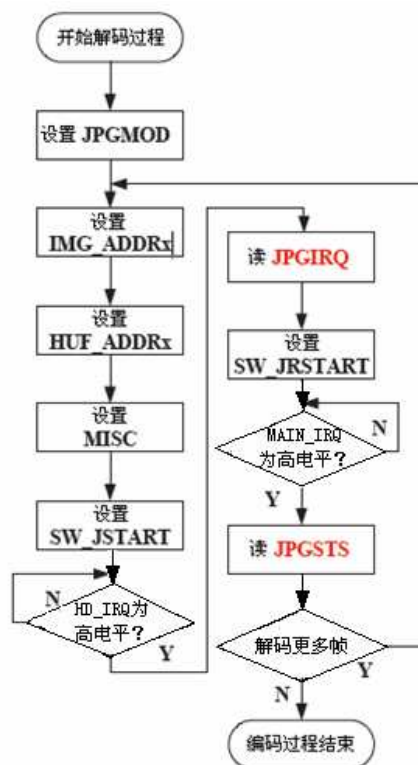


图 22-5 软件控制解码的流程图

采取下列步骤，为软件控制的 JPEG 解码：

- (1) 在 JPGMOD 寄存器中，设置流程模式为解码流程。
- (2) 设置第一帧解码图像数据 IMG\_ADDR0 和第二帧解码图像数据地址 IMG\_ADDR1 的目的地址。
- (3) 设置第一帧 JPEG 文件 HUFADDR0 和下一 JPEG 文件地址 HUFADDR1 的源地址。
- (4) 设置混合寄存器 MISC（设置 MODE\_SEL 为 0x1 或 0x2，DMS 位为 0）。
- (5) 设置 SW\_JSTART 为高电平。
- (6) 如果 HD\_IRQ 为高电平和 ERR\_IRQ 为低电平，读 JPGIRQ 寄存器以清除内部等待 IRQ。
- (7) 设置 SW\_JRSTART 为高电平。
- (8) 如果 MAIN\_IRQ 为高电平和 ERR\_IRQ 为低电平，从 JPGCNT 寄存器读取帧大小（字节）。
- (9) 必须读 JPGIRQ 和 JPGSTS 寄存器以清除内部等待中断。

代码实现：

具体在 ARM11 中 JPEG 的解码，代码编写如下：

```
void JPEG_InitRegsForDecoding(
    u32 uSrcAddr, u32 uDstAddr,
    JPEG_DEC_MODE eMode, bool bIncremental, bool bIsMotion
)
{
    u32 uJpgConVal = 0;
    u32 uMisc = 0;
    if (eMode == HEADER || eMode == HEADER_N_BODY)
    {
        Outp32(JPGMOD, 0x8); // Process mode: Decoding
        Outp32(JHUFADDR0, uSrcAddr); // Address of compressed input data
        Outp32(JHUFADDR1, uSrcAddr); // Address of compressed input data
        Outp32(JPGIRQEN, 0xf<<3); // JPGIRQEN[6:3]=For several error conditions @2006.6.8

        //Outp32(JPGIRQEN, 0xf); // JPGIRQEN[6:3]=For several error conditions @2006.6.8
    }
}
```

```

    if (eMode == HEADER_N_BODY)
    {
        Outp32(JIMGADDR0, uDstAddr); // Address of decompressed image
        Outp32(JIMGADDR1, uDstAddr); // Address of decompressed image
    }
    uJpgConVal = (eMode == HEADER) ? DISABLE_HW_DEC : ENABLE_HW_DEC;
}
else // eMode == BODY
{
    Outp32(JIMGADDR0, uDstAddr); // Address of decompressed image
    Outp32(JIMGADDR1, uDstAddr); // Address of decompressed image
}
if (eMode == BODY || eMode == HEADER_N_BODY)
{
    uJpgConVal |= (bIsMotion == true) ? ENABLE_MOTION_DEC : DISABLE_MOTION_DEC;
    uMisc = (bIncremental == true) ? INCREMENTAL_DEC : NORMAL_DEC;
}
Outp32(JPG_CON, uJpgConVal);
Outp32(JMISC, uMisc);
}

```

JPEG 解码顺序：硬件控制解码。（只有 1 帧）如图 22-6 所示。

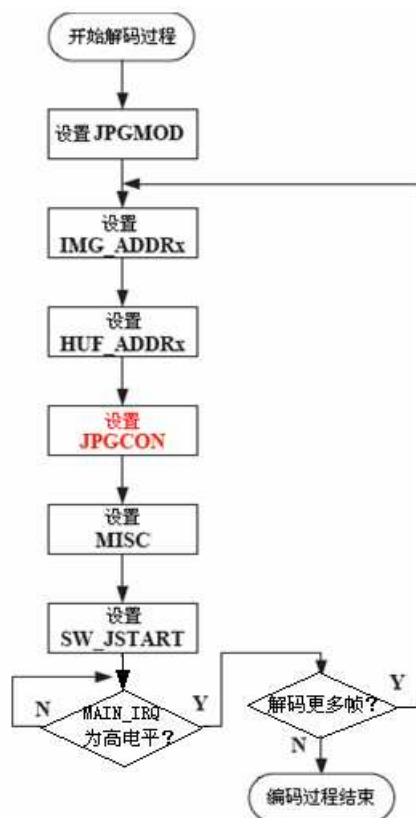


图 22-6 硬件控制解码的流程图

采取下列步骤，硬件控制的 JPEG 解码：

- (1) 在JPGMOD寄存器中，设置流程模式为解码流程。
- (2) 设置第一帧解码图像数据IMG\_ADDR0和第二帧解码图像数据地址IMG\_ADDR1的目的地址。
- (3) 设置第一帧 JPEG 文件 HUFADDR0 和下一 JPEG 文件地址 HUFADDR1 的源地址。
- (4) 在 JPGCON 寄存器中，设置 HW\_DEC 为 1
 

MJ_ENC	HW_DEC	CLK_SEL	MJ_DEC
0	1	0	0
- (5) 设置混合寄存器 MISC（设置 MODE\_SEL 为 0x1 或 0x2，DMS 位为 0）。
- (6) 设置 SW\_JSTART 为高电平。
- (7) 如果 MAIN\_IRQ 为高电平和 ERR\_IRQ 为低电平，从 JPGCNT 寄存器读取帧大小（字节）。
- (8) 必须读 JPGIRQ 和 JPGSTS 寄存器以清除内部等待中断。

代码实现：

具体在 ARM11 中 JPEG 的解码，代码编写如下：

```
void JPEG_InitRegsForDecoding1(
    u32 uSrcAddr, u32 uDstAddr,
    bool bIsHeaderOnly, bool bIncremental, bool bIsMotion
)
{
    u32 uJpgConVal = 0;
    Outp32(JPGMOD, 0x8); // Process mode: Decoding
    Outp32(JHUFADDR0, uSrcAddr); // Address of compressed input data
    Outp32(JIMGADDR0, uDstAddr); // Address of decompressed image
    Outp32(JHUFADDR1, uSrcAddr); // Address of compressed input data
    Outp32(JIMGADDR1, uDstAddr); // Address of decompressed image
    Outp32(JPGIRQEN, 0xf<<3); // JPGIRQEN[6:3]=For several error conditions @2006.6.8
#ifdef 0
    Outp32(JPG_CON, (bIsMotion ? ENABLE_MOTION_DEC : DISABLE_MOTION_DEC));
    u32 uJpgConVal;
    jpgInp32(JPG_CON, uJpgConVal);
    if (bIsHeaderOnly)
        uJpgConVal &= ~(1<<2);
    else
        uJpgConVal |= (1<<2);
    Outp32(JPG_CON, uJpgConVal);
#else
    uJpgConVal = (bIsMotion ? ENABLE_MOTION_DEC : DISABLE_MOTION_DEC);
    uJpgConVal |= (bIsHeaderOnly ? DISABLE_HW_DEC : ENABLE_HW_DEC);
    Outp32(JPG_CON, uJpgConVal);
#endif
    if (bIncremental)
        Outp32(JMISC, (1<<3));
```

```

else
    Outp32(JMISC, (0<<3));
}

```

Motion JPEG 编码顺序，如图 22-7 所示。

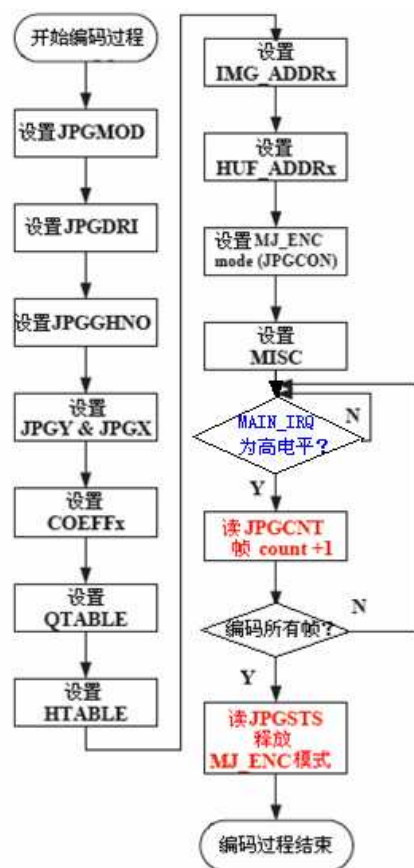


图 22-7 Motion JPEG 编码的流程图

采取下列步骤，为 Motion JPEG 编码：

- (1) 在JPGMOD寄存器中，设置流程模式为编码流程和亚抽样模式。
- (2) 设置 MCU 和 RST 标记寄存器 JPGDRI。
- (3) 设置 Q 和 H 表数目寄存器 JPGQHNO。
- (4) 设置 JPGY 和 JPGX 寄存器。
- (5) 设置系数寄存器 COEFF1、COEFF2、COEFF3，用于色彩空间转换。
- (6) 设置 QTABLE 和 HTABLE。

- (7) 设置源图像数据，第一帧地址寄存器 IMG\_ADDR0 和第二帧的 IMG\_ADDR1。
- (8) 设置 JPEG 目的文件地址寄存器 HUFADDR0 和下一编码 JPEG 文件地址寄存器 HUFADDR1。
- (9) 在JPGCON 寄存器中，设置运动JPEG编码模式。

MJ\_ENC, HW\_DEC, CLK\_SEL, MJ\_DEC

1            0            0            0

- (10) 设置混合寄存器 MISC（设置 MODE\_SEL 为 0x1 或 0x2，EMS 位为 0）。

(11) 如果 MAIN\_IRQ 为高电平和 ERR\_IRQ 为低电平，从 JPGCNT 寄存器和增加编码帧数量中，读帧大小（字节）。

- (12) 如果所有帧都被编码了，读JPGIRQ和JPGSTS 寄存器，以清除内部等待中断。

- (13) 在 JPGCON 寄存器中，MJ\_ENC 被禁用。

代码实现：

具体在 ARM11 中 Motion JPEG 的编码，代码编写如下：

//Motion JPEG 编码

```
void JPEG_StartEncodingMotionJPEG(
    u16 usHSz, u16 usVSz, u32 uSrcAddr, CSPACE eRawType,
    u32 uDestAddr, JPEG_TYPE eJpgType)
{
    Assert(eRawType == YCBYCR || eRawType == RGB16);

    JPEG_Reset();
    //基本的编码函数 JPEG_InitRegsForEncoding
    JPEG_InitRegsForEncoding(usHSz, usVSz, uSrcAddr, eRawType, uDestAddr, eJpgType, false,
true);
    // DisableMotionEncoding();
    Outp32(JSTART, 0); // 开始
}
```

```
void JPEG_InitIpForMotionEncoding(
    u16 uRawHsz, u16 uRawVsz, u32 uRawAddr, CSPACE eRawType,
```

```

u32 uJpgAddr, JPEG_TYPE eJpgType, u32 uMJpegMaxSize)
{
    Assert(eRawType == YCBYCR || eRawType == RGB16);
    printf(" Enc: x=%d, y=%d, yuv=0x%08x~0x%08x, YUV%d\n",
        uRawHsz, uRawVsz, uRawAddr, uRawAddr+(uRawHsz*uRawVsz*2), (eJpgType == JPEG_422) ? 422:
420);
    JPEG_InitRegsForEncoding(uRawHsz, uRawVsz, uRawAddr, eRawType, uJpgAddr, eJpgType, false,
true); // 配置寄存器
    JPEG_SetNextFrameStartAddr(uJpgAddr+uMJpegMaxSize);
    Outp32(JPGCNT, 0);
    Outp32(JPG_CON, (1<<3)); // 开始 Motion Jpeg 编码
    //+daedoo
    Outp32(JSTART, 0); //开始
}

```

Motion JPEG 解码顺序，如图 22-8 所示。



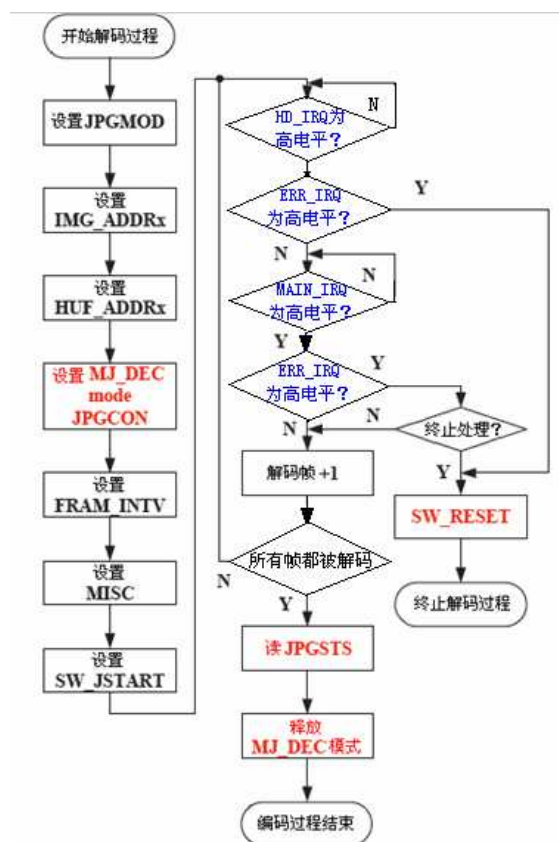


图 22-8 运动 JPEG 解码的流程图

采取下列步骤，为运动的 JPEG 解码：

- (1) 在JPGMOD寄存器中，设置流程模式为解码流程。
- (2) 设置第一帧解码图像数据IMG\_ADDR0和第二帧解码图像数据地址IMG\_ADDR1的目的地址。
- (3) 设置第一帧JPEG文件HUFADDR0和下一JPEG文件地址HUFADDR1的源地址。
- (4) 在JPGCON寄存器中，设置运动JPEG解码模式。

MJ\_ENC, HW\_DEC, CLK\_SEL, MJ\_DEC

0	0	0	1	←基于HCLK的帧速率计数
0	0	1	1	←基于PCLK的帧速率计数

这里，PCLK是像素时钟，HCLK是系统时钟。

- (5) 在FRAM\_INTV中，设置帧解码速率。该值必须大于一帧解码的时间。
- (6) 设置混合寄存器MISC（设置MODE\_SEL 为0x1或 0x2，EMS位为0）。
- (7) 设置SW\_JSTART为高电平。这是初始化开始命令。初始化开始命令以后，当FRAME\_INTV寄存器的

显示的时间过去时，JPEG将产生下一个开始命令。

代码实现：

具体在 ARM11 中 Motion JPEG 的解码，代码编写如下：

```
void JPEG_StartDecodingMotion(u32 uJpgAddr, u32 uRawAddr, u32 uMjpegMaxSize)
{
    printf(" Dec: jpeg=%08x, raw=%08x\n", uJpgAddr, uRawAddr);
    JPEG_Reset();
    // Delay(1000);
    //=====//
    // For compling below func. is changed to InitRegsForDecoding(uJpgAddr, uRawAddr,
    HEADER_N_BODY)
    // so, below code shoule be modified when motion decoding func. is written
    #if 0
        InitRegsForDecoding(uJpgAddr, uRawAddr);
    #else
        JPEG_InitRegsForDecoding(uJpgAddr, uRawAddr, HEADER_N_BODY, false, false);
    #endif
    //=====//
    JPEG_SetNextFrameStartAddr(uJpgAddr + uMjpegMaxSize);
    Outp32(JPG_CON, (1<<0)); // Enable auto decoding
    // Outp32(JFRAME_INTV, 0x800000);
    Outp32(JFRAME_INTV, 0x300000);
    Outp32(JSTART, 1);
}
```

## 22.3 JPEG 编码寄存器

寄存器	偏移量	读/写	描述	复位值
JPGMOD	0x00	读/写	处理模式寄存器。	0x00000000
JPGSTS	0x04	读	操作状态寄存器。	0x00000000

JPGQHNO	0x08	读/写	量化表数目寄存器和哈夫曼表数目寄存器。	0x00000000
JPGDRI	0x0c	读/写	微控制器嵌入 RST 标记。	0x00000000
JPGY	0x10	读/写	垂直分辨率。	0x0000
JPGX	0x14	读/写	水平分辨率。	0x0000
JPGCNT	0x18	读	压缩数据的字节数目。	–
JPGIRQS	0x1c	读/写	中断设置寄存器。	0x0
JPGIRQ	0x20	读	中断状态寄存器。	–
QTBL0	0x400 0x404 ..... 0x4FC	写	8 位量化表数 0（在地址上具有 4 个远程交换的 64 数据）。	–
QTBL1	0x500 0x504 ..... 0x5FC	写	8 位量化表数 1（在地址上具有 4 个远程交换的 64 数据）。	–
QTBL2	0x600 0x604 ..... 0x6FC	写	8 位量化表数 2（在地址上具有 4 个远程交换的 64 数据）。	–
QTBL3	0x700 0x704 ..... 0x7FC	写	8 位量化表数 3（在地址上具有 4 个远程交换的 64 数据）。	–
HDCTBL0	0x800 0x804 ..... 0x83C	写	每个代码长度的数量（在地址上具有 4 个远程交换的 16 数据）。	–
HDCTBLG0	0x840	写	用于发生率的群阶数（在地址上具有 4 个	–

	0x844 ..... 0x86C		远程交换的 12 数据)。	
HACTBL0	0x880 0x884 ..... 0x8BC	写	每个代码长度的数量 (在地址上具有 4 个远程交换的 16 数据)。	—
HACTBLG0	0x8C0 0x8C4 ..... 0xB44	写	用于发生率/组数量的群阶数 (在地址上具有 4 个远程交换的 162 数据)。	—
HDCTBL1	0xC00 0xC04 ..... 0xC3C	写	每个代码长度的数量 (在地址上具有 4 个远程交换的 16 数据)。  8 位寄存器	—
HDCTBLG1	0xC40 0xC44 ..... 0xC6C	写	用于发生率的群阶数 (在地址上具有 4 个远程交换的 12 数据)。  8 位寄存器	—
HACTBL1	0xC80 0xC84 ... 0xCBC	写	每个代码长度的数量 (在地址上具有 4 个远程交换的 16 数据)。  8 位寄存器	—
HACTBLG1	0xCC0 0xCC4 ..... 0xF44	写	用于发生率/组数量的群阶数 (在地址上具有 4 个远程交换的 162 数据)。  8 位寄存器	—
IMG_ADDR0	0x1000	读/写	源或目的图像地址 1。	0x00000000
IMG_ADDR1	0x1004	读/写	源或目的图像地址 2。	0x00000000

HUFADDR0	0x1008	读/写	源或目的 JPEG 文件地址 1。	0x00000000
HUFADDR1	0x100C	读/写	源或目的 JPEG 文件地址 2。	0x00000000
SW_JSTART	0x1010	读/写	开始 JPEG 过程： 0：没有开始过程 1：开始过程	0x00000000
SW_JRSTART	0x1014	读/写	重新开始 JPEG 过程： 0：没有重新开始过程 1：重新开始过程	0x00000000
S_RESET_CON	0x1018	读/写	0：软复位使能 1：软复位禁用	0x00000001
JPG_CON	0x101C	读/写	JPEG 控制寄存器。	0x00000000
COEF1	0x1020	读/写	系数值 RGB ↔ YcbC 的转换器。	0x00000000
COEF2	0x1024	读/写	系数值 RGB ↔ YcbC 的转换器。	0x00000000
COEF3	0x1028	读/写	系数值 RGB ↔ YcbC 的转换器。	0x00000000
MISC	0x102C	读/写	杂项。	0x00000000
FRAM_INTV	0x1030	读/写	帧间的间隔时间值。 (基础时钟周期)	0x00000000

### 22.3.1. JPEG 过程模式寄存器(JPGMOD)

Off=0x00, R/W, Reset=0x0000\_0000

JPGMOD	位	描述	初始状态
Reserved	[31:4]	保留。	0x00000000
Process_Mode	[3]	过程模式： 0：编码过程。 1：解码过程。	
Sub_sampling_Mode	[2:0]	亚抽样模式： 0x0：色度4:4:4格式 0x1：色度4:2:2格式	

		0x2: 色度4:2:0格式 0x6: 色度4:1:1格式 0x3: 灰色格式(单一组成) 其他被禁止。在解码过程中，它们是只读的。 在编码过程中，只有 0x1 或 0x2 被允许。	
--	--	---	--

22.3.2. JPEG 状态寄存器(JPGSTS)

Off=0x04, R, Reset=0x0000\_0000

JPGSTS	位	描述	初始状态
Reserved	[31:1]	保留。	0x00000000
Status	[0]	过程模式：  1: 没有进行正常的操作  0: 进行正常的操作	

22.3.3. JPEG 量化表和哈夫曼表数目寄存器(JPGQHN0)

Off=0x08, R/W, Reset=0x0000\_0000

JPGQHN0	位	描述	初始状态
Reserved	[31:14]	保留。	0x00000000
Q_table_num3	[13:12]	量化表数目用于第三种色彩分量。	
Q_table_num2	[11:10]	量化表数目用于第二种色彩分量。	
Q_table_num1	[9:8]	量化表数目用于第一种色彩分量。	
Reserved	[7:6]	保留。	
H_table_num3_ac	[5]	哈夫曼表数目用于第三种交流色彩分量。	
H_table_num3_dc	[4]	哈夫曼表数目用于第三种直流色彩分量。	
H_table_num2_ac	[3]	哈夫曼表数目用于第二种交流色彩分量。	
H_table_num2_dc	[2]	哈夫曼表数目用于第二种直流色彩分量。	
H_table_num1_ac	[1]	哈夫曼表数目用于第一种交流色彩分量。	
H_table_num1_dc	[0]	哈夫曼表数目用于第一种直流色彩分量。	

注：灰色模式时，让第一和第三色彩分量有相同的值。（量化表数目和哈夫曼表数目）

22.3.4. JPEG 重置间隔寄存器(JPGDRI)

Off=0x0C, R/W, Reset=0x0000\_0000

JPGDRI	位	描述	初始状态
Reserved	[31:16]	保留。	0x0000_0000
Reset_Interval	[15:0]	重置间隔，确定两个邻近的MCU期间的RST标记的距离。	

注：只有在压缩时才有效。当JPGDRI设置为零时，DRI和RST标记将不被声明。

22.3.5. JPEG 垂直大小寄存器(JPGY)

Off=0x10, R/W, Reset=0x0000\_0000

JPGY	位	描述	初始状态
Reserved	[31:16]	保留。	0x0000_0000
VSIZE	[15:0]	在垂直方向上定义图像大小的值。	

22.3.6. JPEG 水平大小寄存器(JPGX)

Off=0x14, R/W, Reset=0x0000\_0000

JPGX	位	描述	初始状态
Reserved	[31:16]	保留。	0x0000_0000
HSIZE	[15:0]	在水平方向上定义图像大小的值。	

22.3.7. JPEG 字节计数寄存器(JPGCNT)

Off=0x18, R, Reset=0x0000\_0000

JPGCNT	位	描述	初始状态
Reserved	[31:24]	保留。	0x0000_0000

B_COUNT	[23:0]	在24位的宽度上，定义压缩数据计数的字节。	
---------	--------	-----------------------	--

22.3.8. JPEG；中断设置寄存器(JPGIRQS)

Off=0x1C, R/W, Reset=0x0000\_0000

JPGIRQS	位	描述	初始状态
Reserved	[31:4]	保留。	0x0000_0000
Intr_enb	[3]	中断使能控制。  0：解压缩期间。在被压缩数据分析的结果中，禁用读图像的大小和取样因素的值。  1：解压缩期间。在被压缩数据分析的结果中，使能读图像的大小和取样因素的值。	
Intr_ctrl	[2:0]	写0x0到设置中断。	

22.3.9. JPEG 中断暂存寄存器(JPGIRQ)

Off=0x20, R, Reset=0x0000\_0000

JPGIRQ	位	描述	初始状态
Reserved	[31:7]	保留。	0x0000_0000
Result_status	[6]	结果状态：  0：不正常的处理结束  1：正常的处理完成	
Reserved	[5]	保留。	
Bitstre_err_status	[4]	位流的错误状态。只有在解压缩期间有效。  0：在被压缩的文件上，没有语法错误。  1：在被压缩的文件上，有语法错误。	
Header_status	[3]	标题状态。只有在解压缩期间有效。  0：图像大小和取样因素值不可读。  1：图像大小和取样因素值可读。	
Reserved	[2:0]	保留。	



注：该就寄存器的值在读出后将被重置。

22. 3. 10. JPEG 量化表 0 寄存器 (QTBL0)

Off=0x400~0x4FC, R/W, Reset= -

QTBL0	位	描述	初始状态
Reserved	[31:8]	保留。	-
Q_val0	[7:0]	定义量化表0。在这里，用户必须写一些值。	

注：0x04作为地址偏移的增加（文字处理）。

22. 3. 11. JPEG 量化表 1 寄存器 (QTBL1)

Off=0x500~0x5FC, R/W, Reset= -

QTBL1	位	描述	初始状态
Reserved	[31:8]	保留。	-
Q_val1	[7:0]	定义量化表1。在这里，用户必须写一些值。	

注：0x04作为地址偏移的增加（文字处理）。

22. 3. 12. JPEG 量化表 2 寄存器 (QTBL2)

Off=0x600~0x6FC, R/W, Reset= -

QTBL2	位	描述	初始状态
Reserved	[31:8]	保留。	-
Q_val2	[7:0]	定义量化表2。在这里，用户必须写一些值。	

注：0x04作为地址偏移的增加（文字处理）。

22. 3. 13. JPEG 量化表 3 寄存器 (QTBL3)

Off=0x700~0x7FC, R/W, Reset= -

QTBL3	位	描述	初始状态
Reserved	[31:8]	保留。	—
Q_val3	[7:0]	定义量化表3。在这里，用户必须写一些值。	

注：0x04作为地址偏移的增加（文字处理）。

22. 3. 14. JPEG 直流哈夫曼表 0 寄存器 (HDCTBL0)

Off=0x800~0x83C, W, Reset= —

HDCTBL0	位	描述	初始状态
Reserved	[31:8]	保留。	—
H_DC_val0	[7:0]	在直流哈夫曼表0中，定义每个编码长度的编码数目。 在这里，用户必须写一些值。	

注：0x04作为地址偏移的增加（文字处理）。

22. 3. 15. JPEG 组直流哈夫曼表 0 寄存器 (HDCTBLG0)

Off=0x840~0x86C, W, Reset= —

HDCTBLG0	位	描述	初始状态
Reserved	[31:8]	保留。	—
H_DC_G_val0	[7:0]	用于发生在直流哈夫曼表0中，定义顺序的组数目。在 这里，用户必须写一些值。	

注：0x04作为地址偏移的增加（文字处理）。

22. 3. 16. JPEG 组交流哈夫曼表 0 寄存器 (HACTBLG0)

Off=0x880~0x8BC, W, Reset= -

HACTBLG0	位	描述	初始状态
Reserved	[31:8]	保留。	—
H_AC_val0	[7:0]	在交流哈夫曼表0中，定义每个编码长度的编码数目。 在这里，用户必须写一些值。	

注：0x04作为地址偏移的增加（文字处理）。

22. 3. 17. JPEG 组交流哈夫曼表 0 寄存器 (HACTBLG0)

Off=0x8C0~0xB44, W, Reset= -

HACTBLG0	位	描述	初始状态
Reserved	[31:8]	保留。	—
H_AC_G_val0	[7:0]	用于发生在交流哈夫曼表0中，定义顺序的组数目。在 这里，用户必须写一些值。	

注：0x04作为地址偏移的增加（文字处理）。

22. 3. 18. JPEG 直流哈夫曼表 1 寄存器 (HDCTBL1)

Off=0xC00~0xC3C, W, Reset= -

HDCTBL1	位	描述	初始状态
Reserved	[31:8]	保留。	—
H_DC_val1	[7:0]	在直流哈夫曼表1中，定义每个编码长度的编码数目。 在这里，用户必须写一些值。	

注：0x04作为地址偏移的增加（文字处理）。

22. 3. 19. JPEG 组直流哈夫曼表 1 寄存器(HDCTBLG1)

Off=0xC40~0xC6C, W, Reset= -

HDCTBLG1	位	描述	初始状态
Reserved	[31:8]	保留。	—
H_DC_G_val1	[7:0]	用于发生在直流哈夫曼表1中，定义顺序的组数目。在这里，用户必须写一些值。	

注：0x04作为地址偏移的增加（文字处理）。

22. 3. 20. JPEG 组交流哈夫曼表 1 寄存器(HACTBLG1)

Off=0xC80~0xCBC, W, Reset= -

HACTBLG1	位	描述	初始状态
Reserved	[31:8]	保留。	—
H_AC_val1	[7:0]	在交流哈夫曼表1中，定义每个编码长度的编码数目。在这里，用户必须写一些值。	

注：0x04作为地址偏移的增加（文字处理）。

22. 3. 21. JPEG 组交流哈夫曼表 1 寄存器(HACTBLG1)

Off=0xCC0~0xF44, W, Reset= -

HACTBLG1	位	描述	初始状态
Reserved	[31:8]	保留。	—
H_AC_G_val1	[7:0]	用于发生在交流哈夫曼表1中，定义顺序的组数目。在这里，用户必须写一些值。	

注：0x04作为地址偏移的增加（文字处理）。

**22. 3. 22. JPEG 图像数据读/写地址 0 寄存器 (IMGADDR0)**

Off=0X1000, R/W, Reset=0x0000\_0000

IMGADDR0	位	描述	初始状态
Image_addr0	[31:0]	源或目的图像地址 0。	0x0000_0000

**22. 3. 23. JPEG 图像数据读/写地址 1 寄存器 (IMGADDR1)**

Off=0X1000, R/W, Reset=0x0000\_0000

IMGADDR1	位	描述	初始状态
Image_addr1	[31:0]	源或目的图像地址 1。	0x0000_0000

IMGADDR0和IMGADDR1是一个图像数据的开始地址。数据压缩前是从该地址的编码模式上读出的，同时解压缩数据从该地址中被保存。当MISC[7:5]为0x1或0x2时是必须的。

如果 (编码或解码) 处理结束，则IMG\_ADDR0或IMG\_ADDR1被交换。

**22. 3. 24. JPEG 数据读/写地址 0 寄存器 (HUFADDR0)**

Off=0X1008, R/W, Reset=0x0000\_0000

HUFADDR0	位	描述	初始状态
Huff_addr0	[31:0]	源或目的 JPEG 文件地址 0。	0x0000_0000

**22. 3. 25. JPEG 数据读/写地址 1 寄存器 (HUFADDR1)**

Off=0X100C, R/W, Reset=0x0000\_0000

HUFADDR1	位	描述	初始状态
Huff_addr1	[31:0]	源或目的 JPEG 文件地址 1。	0x0000_0000

HUFADDR0和HUFADDR1是用于JPEG数据的开始地址。在编码模式和解码模式，JPEG数据从该地址中被保存或读出。

如果 (编码或解码) 处理结束，则HUFADDR0和HUFADDR1被交换。

22. 3. 26. JPEG 软件开始控制寄存器 (SW\_JSTART)

Off=0x1010, R/W, Reset=0x0000\_0000

SW_JSTART	位	描述	初始状态
Reserved	[31:1]	保留。	0x0000_0000
S_JSTART	[0]	0: 没有开始 JPEG 处理 1: 开始 JPEG 处理	

22. 3. 27. JPEG 软件重新开始控制寄存器 (SW\_JRSTART)

Off=0x1010, R/W, Reset=0x0000\_0000

SW_JRSTART	位	描述	初始状态
Reserved	[31:1]	保留。	0x0000_0000
S_JRSTART	[0]	在解码处理中，该值可用。 0: 没有开始主解码处理 1: 开始主解码处理	

在S3C6410X中 JPEG的解码过程是 S\_JSTART命令开始JPEG解码处理，如果JPEG引擎获得JPEG文件的头信息（这意味着HD\_IRQ的高电平值被检测），则S\_JRSTART命令设置为高电平，以开始主解码过程。

22. 3. 28. JPEG 软件复位控制寄存器 (SW\_RESET\_CON)

Off=0x1018, R/W, Reset=0x0000\_0000

SW_RESET_CON	位	描述	初始状态
Reserved	[31:1]	保留。	0x0000_0001
S_RESET	[0]	0: 软件复位使能 1: 软件复位禁用	

如果S\_RESET标志位为低电平，除了SW\_RESET\_CON所有寄存器都被软复位。

22. 3. 29. Motion JPEG 控制寄存器(JPGCON)

Off=0x101C, R/W, Reset=0x0000\_0000

JPGCON	位	描述	初始状态
Reserved	[31:4]	保留。	0x0000_0000
MJ_ENC	[3]	0: Motion JPEG 编码禁用。 1: Motion JPEG 编码使能。	
HW_DEC	[2]	如果该标志位为高电平,SW_JRSTART 命令不能用于解码 JPEG 文件。 0: 硬件解码禁用 1: 硬件解码使能	
CLK_SEL	[1]	选择计数器时钟。 0: HCLK 1: PCLK	
MJ_DEC	[0]	0: Motion JPEG 解码禁用 1: Motion JPEG 解码使能	

22. 3. 30. JPEG 系数 1 寄存器(COEF1)

Off=0x1020, R/W, Reset=0x0000\_0000

COEF1	位	描述	初始状态
Reserved	[31:24]	保留。	0x0000_0000
COEF11	[23:16]	COEF11 的系数值。	
Reserved	[15:8]	保留。	
COEF13	[7:0]	COEF13 的系数值。	

22. 3. 31. JPEG 系数 2 寄存器(COEF2)

Off=0x1024, R/W, Reset=0x0000\_0000

COEF2	位	描述	初始状态
Reserved	[31:24]	保留。	0x0000_0000
COEF21	[23:16]	COEF21 的系数值。	
COEF22	[15:8]	COEF22 的系数值。	
COEF23	[7:0]	COEF23 的系数值。	

22. 3. 32. JPEG 系数 3 寄存器(COEF3)

Off=0x1028, R/W, Reset=0x0000\_0000

COEF3	位	描述	初始状态
Reserved	[31:24]	保留。	0x0000_0000
COEF31	[23:16]	COEF31 的系数值。	
COEF32	[15:8]	COEF32 的系数值。	
Reserved	[7:0]	保留。	

YcbCr 转换器中的计算是基于以下矩阵的，如图 22-9 所示。在编码模式上，SWSEL =0x2 (RGB565)时，YcbC 转换器被使用。如图 22-9 所示。

R

G

B

=

COEF11

0

COEF13

COEF21

COEF22

COEF23

COEF31

COEF32

0

\*

Y - c1

Cb - 128

Cr - 128

图 22-9 RGB 和 YcbCr 格式之间的相关矩阵

22. 3. 33. JPEG 杂项寄存器(MISC)

Off=0x102C, R/W, Reset=0x0000\_0000



MISC	位	描述	初始状态
Reserved	[31:8]	保留。	0x0000_0000
MODE_SEL	[7:5]	模式选择：  0x0：专用引脚(YCbCr4:2:2)  0x1：内存(YCbCr4:2:2)  0x2：内存(RGB 565)  其他被禁止。	
Reserved	[4]	保留。	
DMS	[3]	解码模式选择位(DMS)。  0：正常解码方法被选择  1：增量的解码模式被选择	
EMS	[2]	编码模式选择位(EMS)。  0：在内存中，JPEG 单位编码一个图像存储。图像和编码流结果被读出，通过 AHB DMA 操作被保存到内存中。  1：JPEG 单位编码的图像，通过 local 信号。  警告：S3C6410X 中没有 local 通道，所以，必须设置为 0。	
MODE_Y16	[1]	MODE_Y16。  0：c1 = 0  1：c1 = 16	
ECS	[0]	JPEG 引擎时钟选择(ECS)。  0：JPEG 引擎时钟同系统时钟是相等的。  1：JPEG 引擎时钟比率是系统时钟比率的一半。	

EMS 位决定是否以内存为基础的编码被使用。在基于内存的编码模式下，有可能设定 JPEG 引擎的时钟率的一半，即系统时钟（设置 ECS 位为 1）。当 EMS 位为 1 时，用户不必设置 ECS 位为 1。

DMS 位的选择决定解码模式是否被使用。当位为 0 时，正常的解码程序将被使用。当 DMS 位为 1 时，增量的解码模式被选择。在这种模式下，JPEG 单位解码一个最初为 8 线或 16 线的编码图像，然后使能 partial\_line\_ready 信号。在此之后，JPEG 单位处于等待，直到输入 next\_partial\_lines 信号被设置为高电平。设置该信号为高电平，使 JPEG 单位解码下一个 8 或 16 线的解码图像。重复该过程，直到解码完成。

通过 IMG\_ADDRESS SFR 从内存地址说明，JPEG 单位总是保存每个组的部分线路。

编码图像的图像源格式决定每个组的部分线路的大小。如果图像格式为 420，每个组的部分线路由 16 线组成。否则，它始终是 8 线的。

ECS 位决定 JPEG 引擎时钟的速度。当位为 0 时，时钟比率与系统时钟比率相同。当该位为 1 时，时钟比率则是系统时钟比率的一半。当 EMS 位为 1 时，用户不可以使用半时钟模式。

22. 3. 34. JPEG 帧间隔寄存器 (FRAM\_INTV)

Off=0x1030, R/W, Reset=0x0000\_0000

FRAM_INTV	位	描述	初始状态
Frame_interval	[31:0]	在Motion JPEG解码模式中，间隔时间值在每个SW_JSTART命令之间。  在Motion JPEG解码模式中，这些值被用于设置解码帧的比率。	0x0000_0000

# 23 调制解调器接口

## 23.1 概述

本章描述了在基本板块与进行数据交换的应用处理器之间的接口。为了进行数据交换，应用处理器有一个 DPSRAM 缓冲区，调制解调芯片可以用典型的异步 SRAM 接口访问 DPARAM 缓冲区。

SRAM 缓冲区的大小为 8 个字节。本节对 SRAM 缓冲区的状态以及中断请求的预先定义特殊地址进行了说明。

调制解调器芯片可以像数据缓冲区内输入数据，向中断端口地址写入中断控制数据进行中断请求。当中断请求被接受以后，AP 读取中断数据；当 AP 访问中断端口地址时，中断清除。同样的方式，AP 可以在数据缓冲区内写入数据，可以向中断端口地址写入中断控制数据，向调制解调器芯片发出的中端请求。

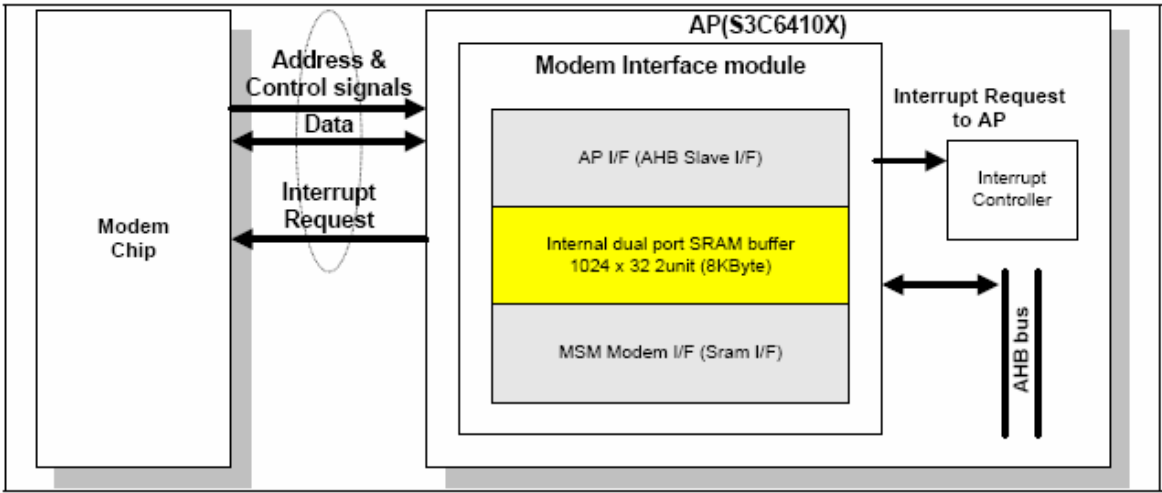


图 23-1 调制解调芯片和 MODEM I/F 模块之间的接口图

---

## 23.2 性能

- (1) 调制解调器芯片采用异步 **SRAM** 接口
- (2) 进行数据转换的 16 位平行总线
- (3) 8k 字节的内部双端口 **SRAM** 缓冲区
- (4) 可以发出进行数据转换的中断请求
- (5) 可程式化的中断端口地址
- (6) 输入/输出电压范围是 1.8v-3.3v
- (7) 调制解调进程的 **AP Boot** 提供一个双端口存储器作为一个调制解调器引导存储器。

## 23.3 外部接口

名称	类型	描述
XhiADR[12:0]	输入	地址总线，通过调制解调器芯片控制
XhiCSn	输入	芯片选择，通过调制解调器芯片控制
XhiCSn_main	输入	LCD 主旁路芯片选择，通过调制解调器芯片控制
XhiCSn_sub	输入	LCD 子旁路芯片选择，通过调制解调器芯片控制
XhiVEn	输入	写使能，通过调制解调器芯片控制
XhiOEn	输入	读使能，通过调制解调器芯片控制
XhiDATA[17:0]	输入/输出	数据总线，通过调制解调器芯片控制
XhiNTR	输出	向调制解调器芯片发出中断请求

## 23.4 内存映射

寄存器	地址	读/写	描述	复位值
MSBM	0x74100000~0x74101FFF	读/写	MODEM I/F SRAM 缓冲区内存	0x00000000

寄存器	地址	读/写	描述	复位值
INT2AP	0x74108000	读/写	向 AP 寄存器发出中断请求	0x00001FFF
INT2MODEM	0x74108004	读/写	向 MODEM 调制解调寄存器发出中断请求	0x00001FFC
MIFCON	0x74108008	读/写	调制解调器接口控制寄存器	0x00000008
MIFPCON	0x7410800C	读/写	调制解调器接口端口控制寄存器	0x00000008
MODEMINTCLR	0x74108010	写	调制解调器接口悬挂中断请求清除	-

## 23.5 中断端口

如果调制解调器芯片或者 AP 访问中断端口时发出中断，或清除中断。通过 AP 可以进行特殊中断地址的设置。默认的地址映射如表

表 23-1 中断请求和清除条件

中断	地址	发出中断请求	清除中断
To AP	0x1FFE	调制解调芯片写入	在 MODEM 接口模块内，AP 向 MODEMINTCLR 寄存器写入
To Modem	0x1FFC	AP 写入或者在 AP 初始化之后	调制解调器芯片向相应的位写入高电平

注意：

- （1）上面的地址是默认的地址值。通过 SFR 可以设置其他的地址值。
- （2）调制解调器接口模块有一个中断清除寄存器-MODEMINTCLR.通过调制解调器接口模块可以产生电平类型中断请求，并可以继续保持中断请求，直到 AP 向寄存器内写入数值进行终端清除。
- （3）有三种情况可以抽出 AP 初始化进程，这三种情况分别是 H/W 复位、停止模式苏醒，以及睡眠模式苏醒。在 AP 初始化进程结束之后，系统模块将稳定并释放复位信号。另外，AP 可以像调制解调器发送中断信号。
- （4）XhiNTR 电平保持低电平状态，因为在系统复位之后 AP 会立即向调制解调器写入中断信号，从而，为了进行通信，调制解调器会首先清空 AP 的调制中断。

调制解调器芯片或 AP(S3C6410X)可以读取数据，读取的数据将说明会发生什么情况：从终端端口地址进行数据交换请求，数据交换进程，发出特殊命令等等。

## 23.5 MODEM AP 引导区

MODEM AP 引导区是指 AP 提供一个用与 MODEM 的引导区域。

MODEM 可以使用 MODEM-IF 模块的内部双端口 SRAM 存储器进行引导。这种情况下，AP 提供“MODEM 复位管脚”，并完成引导操作。同样，AP 下载 MODEM 引导代码到其双端口 SRAM。如果 MODEM 的芯片选择连接到 ‘XhiCSn’，MODEM 可以在没有外部引导存储器的情况下进行引导操作。

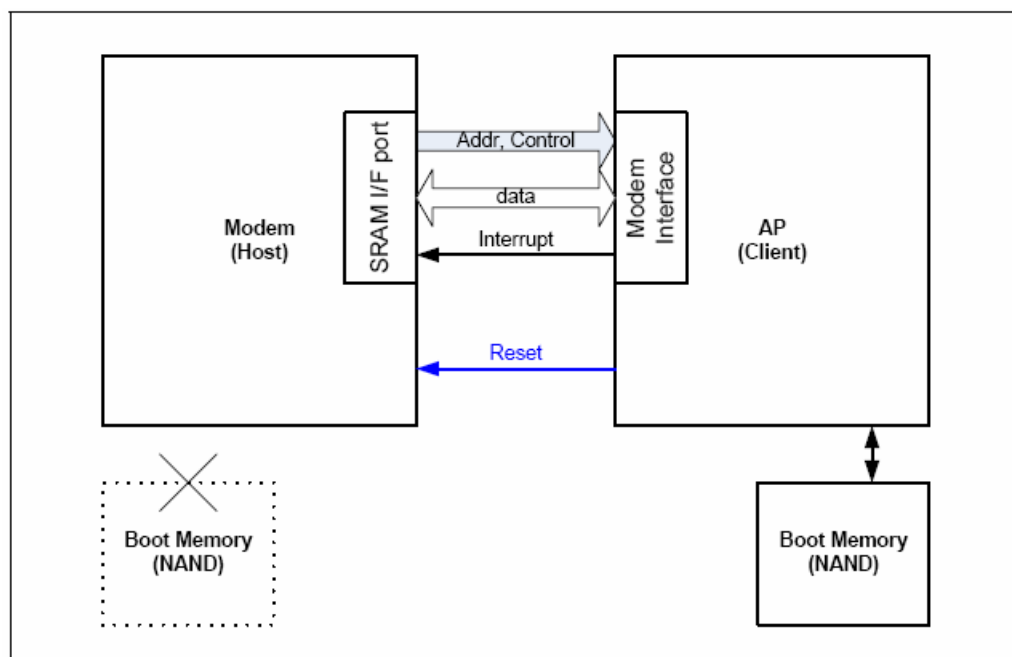


图 23-2 MODEM 接口的 MODEM AP 引导区

在此模式下，AP 可以通过 AP 的 GPIO 输出端口控制“调制解调器的 H/W 复位管脚”。GPIO 输出端口的初始状态是下拉状态。

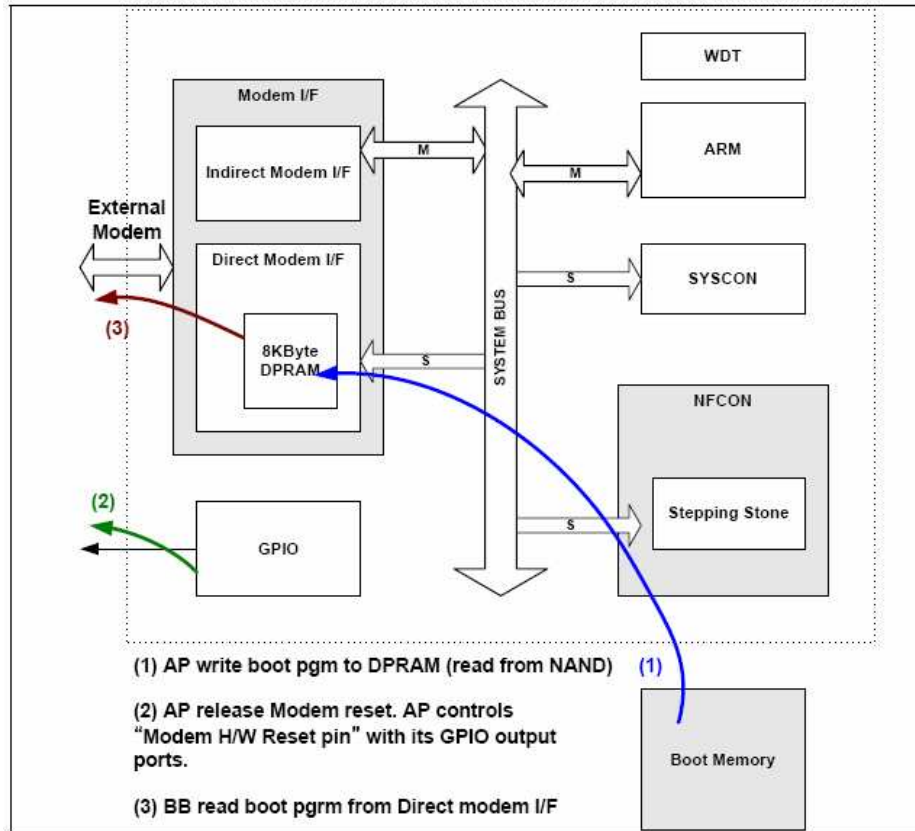


图 23-3 MODEM AP 引导区的引导流程

注意：MODEM 复位管脚有 AP 的 GPIO 管脚控制。当初始电源复位时 MODEM 复位管脚必须是下拉模式。MODEM 的引导地址芯片选择管脚必须与 “XhiCSn” 连接。

## 23.6 地址映射

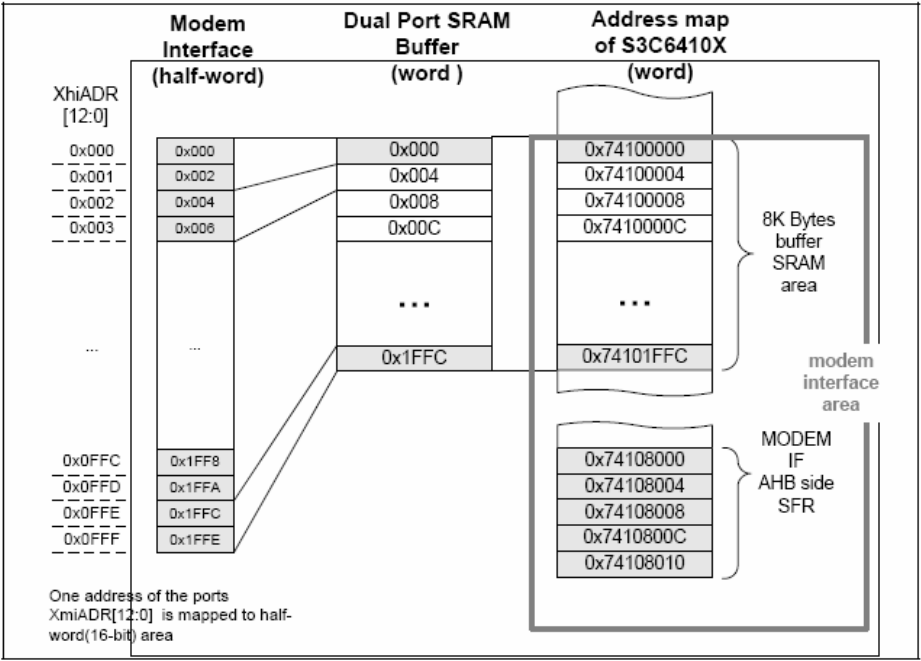


图 23-4 MODEM 接口地址映射

当 XhiCSn= '0'，MODEM 地址映射如下表

XhiADDR			主机/调制解调器接口选择	描述
[12]	[11:8]	[7]		
0	XXX	X	调制解调器接口(直接接口)	
1	0000	X	主机接口接口(间接接口)	Xhi_ADDR[2]=' 0' (固定)
1	0001	0	睡眠/停止模式苏醒判断	写操作
1	0001	1	睡眠/停止模式苏醒清除	写操作
1	100X	X	LCD 主旁路	外部 LCD 设备主旁路模式的调制解调
1	101X	X	LCD 子旁路	外部 LCD 设备子旁路模式的调制解调

注释:

(1) VIC 内的调制解调器中断源在 AP 进入停止模式前必须有效。



- (2) 在进入下一次睡眠/停止模式苏醒操作之前需要清除前一个睡眠/停止模式流程。在停止/睡眠模式苏醒运行之后，将向调制解调器产生自动中断信号。XhiNTR 输出为低电平，‘INT2MODEM\_REG’ 的 bit[0] 设置为高电平。因此调制解调器将进行中断清除（向‘INT2MODEM\_REG 的 bit[0] 写入 1）。调制解调器的自动中断向调制解调器告知 AP 的稳定状态。
- (3) 当调制解调器采用 LCB 旁路模式时，主 LCD 和子 LCD 芯片用 Xhi\_CS<sub>n</sub>\_main，Xhi\_CS<sub>n</sub>\_sub 进行信号选择。

## 23.7 时序图

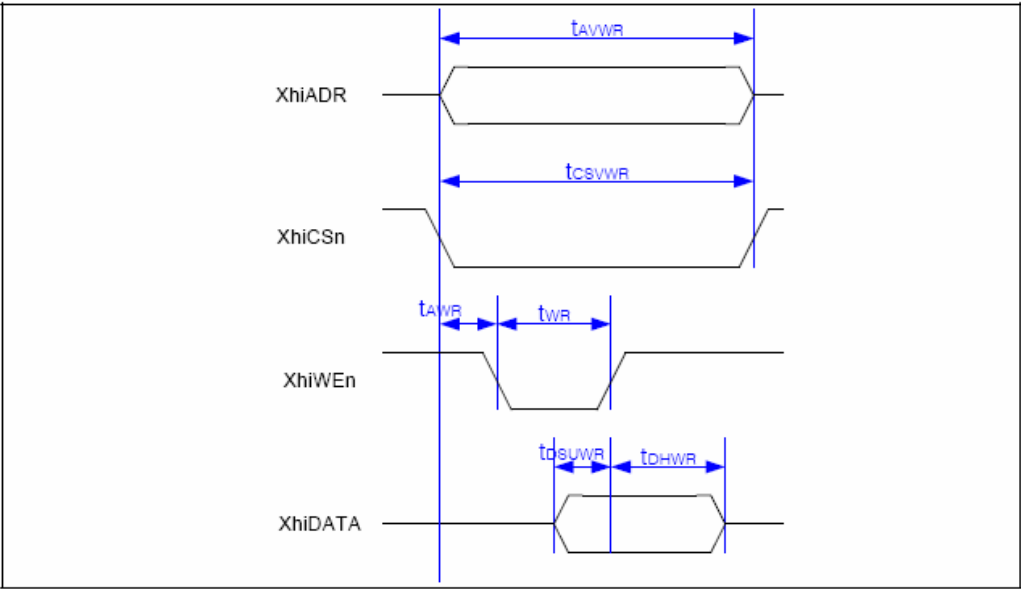


图 23-5 调制解调器接口的写操作时序图

表 23-2 调制解调器接口写时序图

参数	描述	Min（ns）	Max（ns）	注释
tAVWR	有效地址到位无效地址	16ns	-	
tCSVWR	芯片选择操作	16ns	-	
tAWR	有效地址到写操作	4ns	-	
tWR	写操作	8ns	-	
tDSUWR	设置写数据	8ns	-	
tDHVWR	保持写数据	4ns	-	

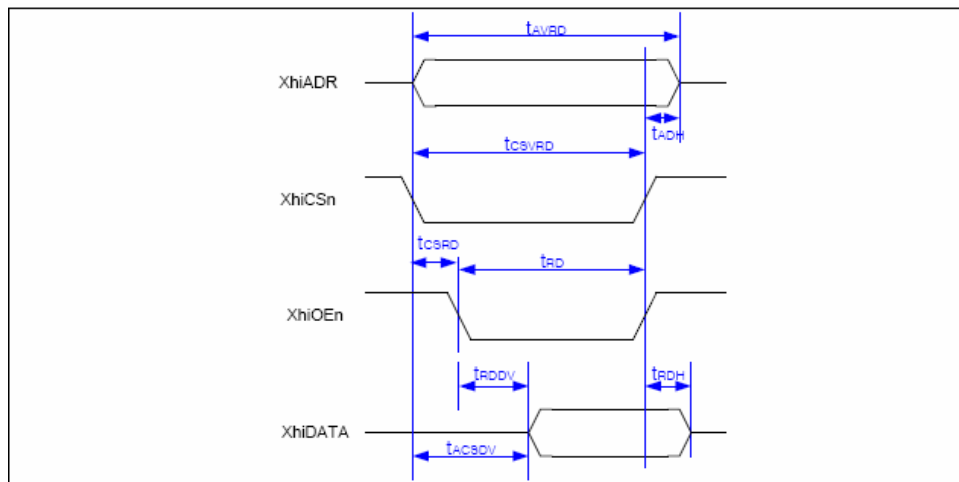


图 23-6 调制解调器接口的读操作时序图

表 23-3 调制解调器接口读时序图

参数	描述	Min (ns)	Max (ns)	注释
$t_{AVRD}$	有效地址到位无效地址	50ns	-	
$t_{ADH}$	保持地址	0ns		
$t_{CSV RD}$	芯片选择操作	50ns	-	
$t_{CSRD}$	有效地址到读操作	14ns	-	
$t_{RD}$	读操作	36ns	-	
$t_{RDDV}$	读操作到有效数据	-	35ns	
$t_{RDH}$	保持读数据	6ns	-	
$t_{ACSCV}$	地址和芯片选择操作到有效数据	-	49ns	

## 23.8 软件接口和寄存器

调制解调器接口提供通用的数据交换方法。调制解调器接口不执行任何复杂的性能，如自动 FIFO 管理等，中断请求、中断清除例外。软件负责完成调制解调器芯片和 AP 之间进行数据交换的请求功能，如数据交换协议，数据缓冲区管理等工作。

## 23.9 调制解调器接口特殊功能寄存器

### 1.AP 寄存器的中断请求（INT2AP）

寄存器	地址	读/写	描述	复位值
INT2AP	0x74108000	读/写	AP 寄存器的中断请求	0x00001FFE

INT2AP	位	描述	初始状态
Reserved	[31:13]	保留	0
INT2AP_ADR	[12:0]	当调制解调器芯片写入此地址时，调制解调器向 AP 发出中断请求。中断请求通过 AP 中断控制器和向 MODEMINTCLR 寄存器写入访问进行清除，	1FFE

### 2. MODEM 寄存器的中断请求（INT2MODEM）

寄存器	地址	读/写	描述	复位值
INT2MODEM	0x74108004	读/写	MODEM 寄存器的中断请求	0x00001FFC

INT2 MODEM	位	描述	初始状态
Reserved	[31:13]	保留	0
INT2MODEM_ADR	[12:0]	AP 通过向该地址写入非 0 值向 MODEM 发出中断请求。 MODEM 通过向相应的位写入高电平进行中断清除	1FFC

### 3. MODEM 接口控制寄存器（MIFCON）

寄存器	地址	读/写	描述	复位值
MIFCON	0x74108008	读/写	中断接口控制接粗阿尼	0x00000008

MIFCON	位	描述	初始状态
Reserved	[31:20]	-	0
DMARXREQEN_1	[19]	调制解调器向 AP（DMA 控制器）写入 DMA 请求（RX1）	0

		使能	
DMARXREQEN_0	[18]	调制解调器向 AP (DMA 控制器) 写入 DMA 请求 (RX0) 使能	0
DMATXREQEN_1	[17]	调制解调器向 AP (DMA 控制器) 能读取 DMA 请求 (RX1) 使	0
DMATXREQEN_0	[16]	调制解调器向 AP (DMA 控制器) 读取 DMA 请求 (RX0) 使能	0
Reserved	[15:4]	-	0
INT2MODEMEN	[3]	调制解调器中断: MODEM_Nirq(XhiINTR)是中断信号使能 '0' =禁止      '1' =使能	1
INT2APEN	[2]	MODEM 向 AP 写入中断 使能	0
Reserved	[1]	保留	0
Fixed	[0]	固定为 0	0

#### 4.调制解调器接口端口控制寄存器（MIFPCON）

寄存器	地址	读/写	描述	复位值
MIFPCON	0x7410800C	读/写	调制解调器接口端口控制寄存器	0x00000008

MIFPCON	位	描述	初始状态
Reserved	[31:6]	-	0
Fixed	[5]	固定为 0	0
INT2M_LEVEL	[4]	调制解调器高电平有效中断：当此位设置为高电平时 MODEM_nIRQ(XhiINTR) 中断信号高电平有效。 '0' =低电平有效      '1' =高电平有效	0
SEL_BYPASS	[3]	为 LCD 旁路选择控制模式 '0' =正常模式      '1' =旁路模式	1
SEL_RS	[2:0]	LCE 旁路路径的 RS 选项 3'b000:XhiADDR[8]->bypass_RS	0

		3'b001: XhiADDR[7]->bypass_RS 3'b010: XhiADDR[6]->bypass_RS 3'b011: XhiADDR[5]->bypass_RS 3'b100: XhiADDR[4]->bypass_RS 3'b101: XhiADDR[3]->bypass_RS	
--	--	---	--

### 5.调制解调器接口清除寄存器（MODEMINTCLR）

寄存器	地址	读/写	描述	复位值
MODEMINTCLR	0x74108010	写	调制解调器接口悬挂中断请求清除	-

MODEMINTCLR	位	描述	初始状态
-	[31:0]	任何数据的写访问此寄存器均会清除调制解调器接口的中断悬挂寄存器	-

注：AP(S3C6410X)的终端控制器 VIC,接收电平触发类型的中断请求。MODEM\_IF 模块发出的中断请求持续保持，直到 ARM 写入高电平清除寄存器的中断。

### 6.DMA 请求 TX 地址寄存器（DMAREQ\_TX\_ADR）

寄存器	地址	读/写	描述	复位值
DMAREQ_TX_ADR	0x74108014	读/写	DMA TX 请求地址寄存器	0x07FE03FE

INT2 AP	位	描述	初始状态
Reserved	[31:29]	保留	0
DMA_TX_ADR_1	[28:16]	当调制解调器芯片读取该地址时，调制解调器接口 DMA 到 AP	07FE
Reserved	[15:13]	保留	0
DMA_TX_ADR_0	[12:0]	当调制解调器芯片读取该地址时，调制解调器接口 DMA 到 AP	03FE

7.DMA 请求 RX 地址寄存器 (DMAREQ\_RX\_ADR)

寄存器	地址	读/写	描述	复位值
DMAREQ_RX_ADR	0x74108018	读/写	DMA RX 请求地址寄存器	0x0FFE0BFE

INT2 AP	位	描述	初始状态
Reserved	[31:29]	保留	0
DMA_RX_ADR_1	[28:16]	当调制解调器芯片写入该地址时，调制解调器接口 DMA 到 AP	07FE
Reserved	[15:13]	保留	0
DMA_TX_ADR_0	[12:0]	当调制解调器芯片写入该地址时，调制解调器接口 DMA 到 AP 源：MODEM_IF RX 0 请求	03FE

## 24 主机接口

### 24.1 概述

S3C6410X 的主机接口模块支持直接访问外部主机设备的功能。通过主机接口协议的选择，可以支持下面所述操作：

- (1) 16 位（协议）寄存器
- (2) SFR 存储器系统内存映射 **single** 读/写操作。
- (3) SFR 存储器系统内存映射 **burst** 读/写操作。
- (4) SFR 存储器系统内存映射重复突发写操作。
- (5) 调制解调区使主机在没有专用的 AP NAND 闪存时可以对 AP 区进行控制。

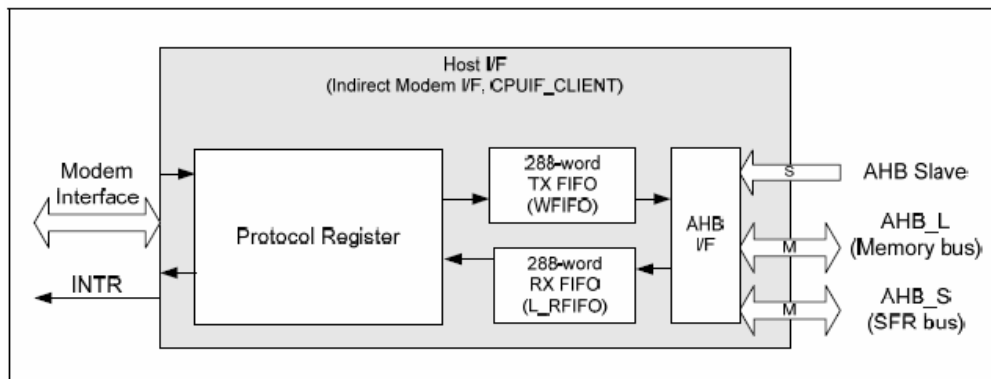


图 24-1 主机 I/F 模块图

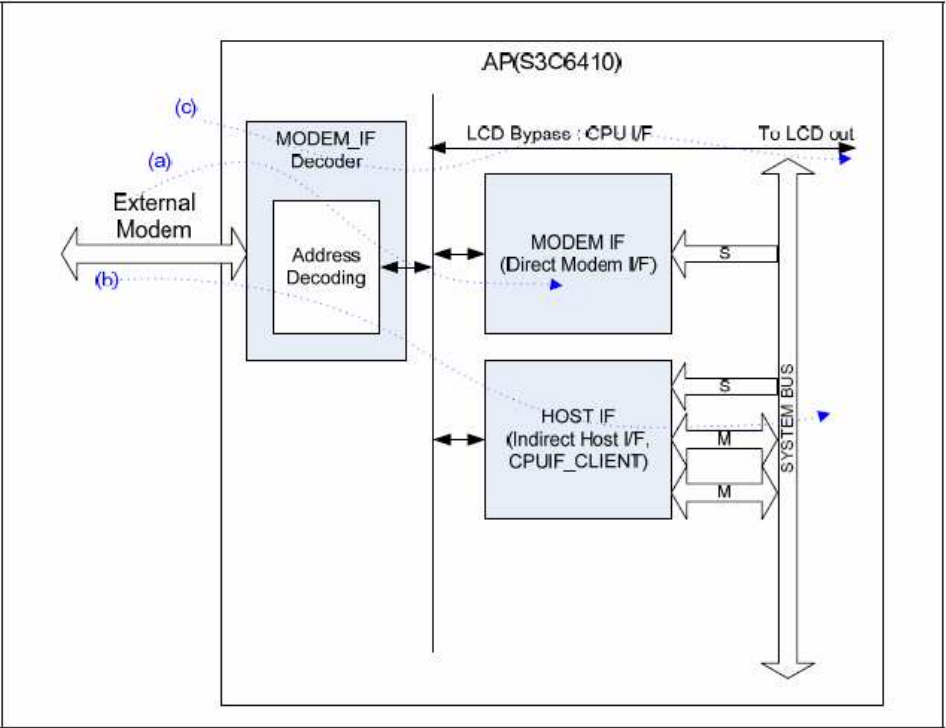


图 24-2 外部主机设备和 AP 区的数据流动图

当 XhiCSn= ‘0’，调制解调区

XhiADDR			主机/调制器 接口选择	描述
[12]	[11:8]	[7]		
0	XXX	X	调制器接口	
1	0000	X	主机接口	Xhi_ADDR[2]='0':间接主机 I/F Xhi_ADDR[2]='1':保留
1	0001	0	睡眠/停止模式复位	写操作
1	0001	1	睡眠/停止模式清除	写操作
1	100X	X	LCD 主旁路	
1	101X	X	LCD 子旁路	

主机接口性能：

- （1）异步间接 16 位 SRAM 型主机接口
- （2）16 位（协议）寄存器



- (3) 写-FIFO 和读 FIFO 支持突发读写交换操作
- (4) 用于数据交换的 32 位的输入、输出寄存器

## 24.2 功能描述

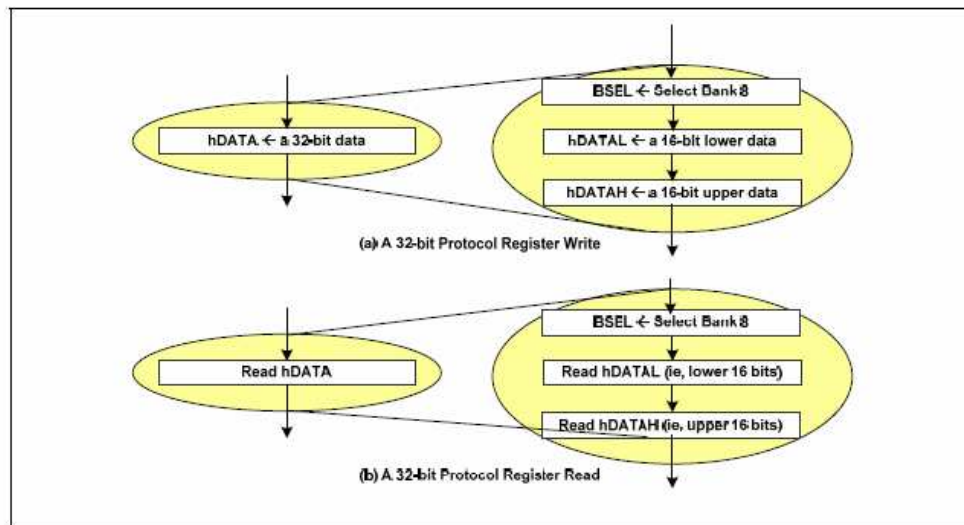


图 24-3 两个 16 位的读/写寄存器访问一个 32 位的（协议）寄存器

### 1. 16 位的读、写（协议）寄存器

访问一个 16 位的（协议）寄存器需要以下步骤：

- (1) 通过写 BSEL 选择一个相应的板块
- (2) 读或写（协议）寄存器

### 2. 同一板块内的 16 位（协议）寄存器读写操作

访问同一板块内的 16 位（协议）寄存器的步骤：

- (1) 写入 BSEL 选择相应的板块
- (2) 读或写低 16 位（协议）寄存器
- (3) 读或写高 16 位（协议）寄存器

### 3 .Single Write

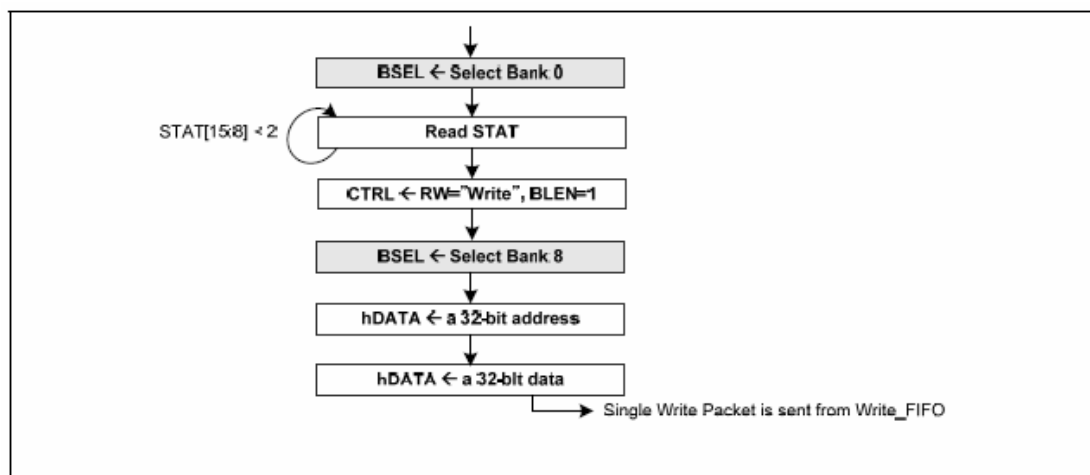


图 24-4 Single Write 流程

### 4.Single Read

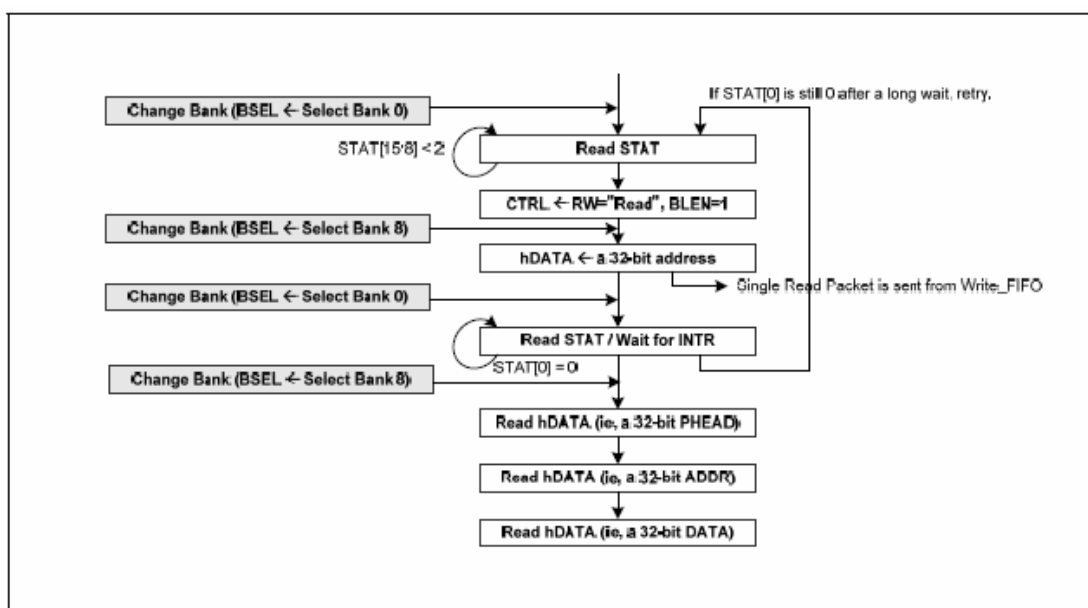


图 24-5 Single Read 流程

这部分里的“HOST I/F”表示主机接口模块；“HOST”表示外部主机设备。

HOST I/F 从 S3C6410 的 AP 中读取结果。HOST I/F 可以在前一个操作完成之前进行一个新的读操作。

结果源不包含任何信息。HOST 必须确保多重读操作的结果不会混淆。例如：从 a 内读取 “source area A” 以后，在还没有等到读取结果出来之前，a 内的 “source area B” 已经被读取，先读取的内容结果会先到达读缓冲区。收到的结果地址与请求的地址保持一致。即使是相同的 “source area”，多重读操作的结果顺序仍然和单个读取的规则一样。

确保 HOST 在下一条读取命令发出之前已经接收到当前命令的读取结果是避免发生未收到读取结果又继续读取下一条指令情况发生的简单方法。

HOST 通过 16 位的状态寄存器或者通过用一个中断计划可以知道 32 位读缓冲区的状态如何。如果使用中断计划，为了知道中断的原因，HOST 必须读取 ISR 内的状态寄存器。

5.Burst Write

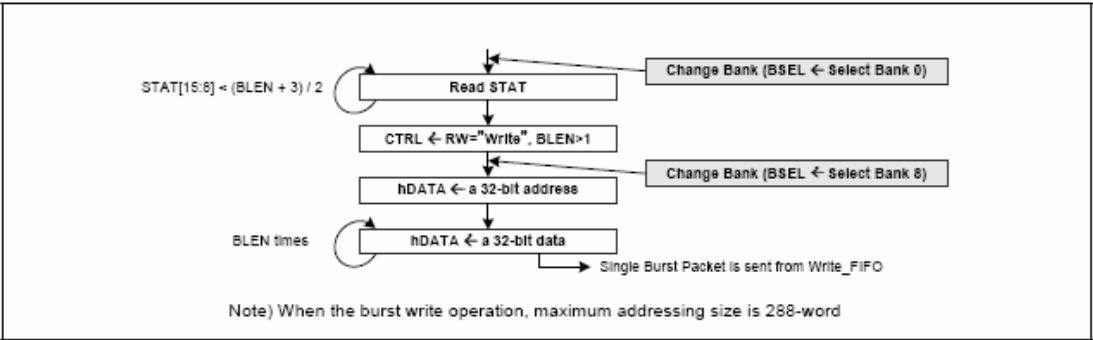


图 24-6 Burst Write 流程

6. Burst Read

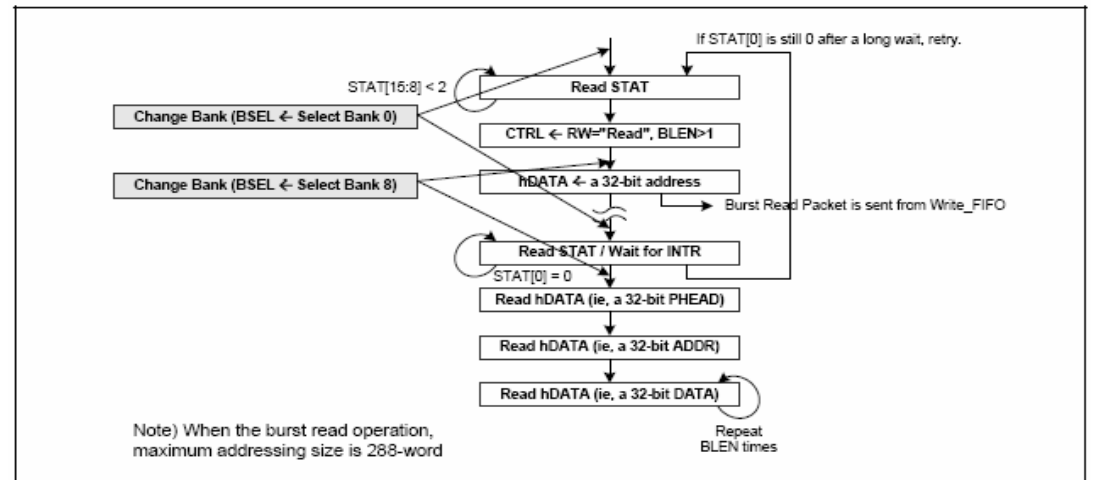


图 24-7 Burst Read 流程

## 7.Modem Booting

Modem Booting 的意思是 Host 控制 AP 区，包括复位。在此情况下，AP 不需要一个外部驱动存储器。调制器在 AP 的驱动存储器内下载代码，并通过 HOST I/F 模块输送到 AP 内部的 Stepping Stone 存储器。

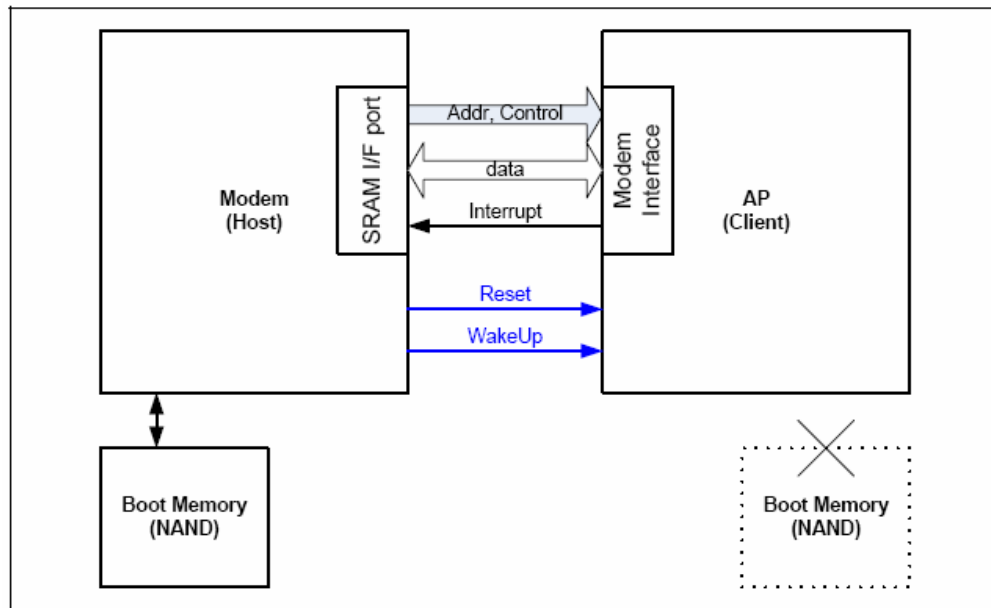


图 24-8 Modem Boot 连接图

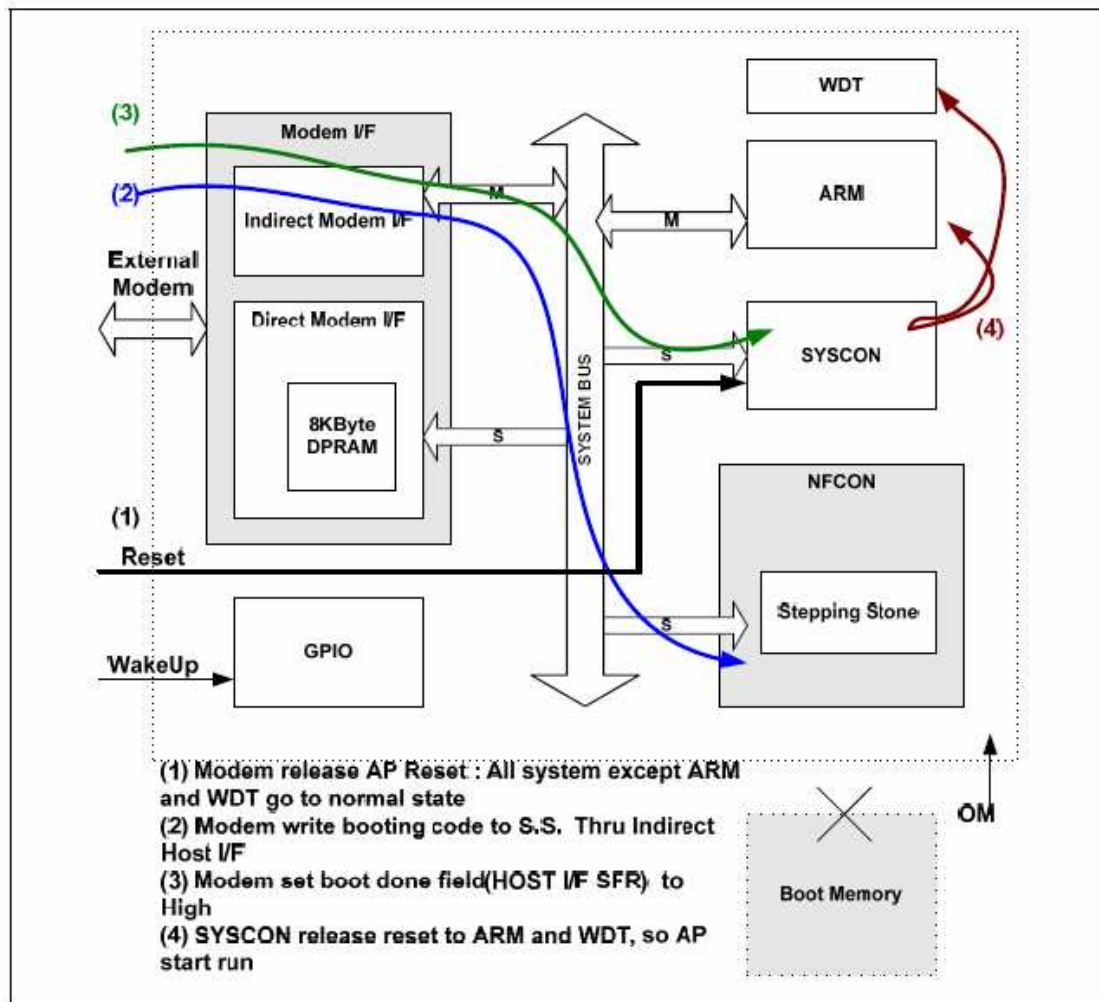


图 24-9 Modem Boot 工作流程

## 8. 信箱接口

调制器用 32 位的输入、输出信箱进行调制器与 AP(S3D6410)之间的内部通信。

## 9. 信箱基本运行

当调制器向输入信箱输入 32 位的数据后，HOST I/F 向 AP (S3C6410) 产生一个中断信号，此时 AP (S3C6410) 可以读取输入信箱的内容，知道调制器发出的请求。输入信箱内的格式可以根据应用自由的定义。当 AP (S3C6410) 读取输入信箱内容是，HOST I/F 内输入信箱标志将自动清除。

当 AP (S3C6410) 向输出信箱写入 32 位的数据时，HOST I/F 想调制器产生中断，此时调试器可以读取输出信箱内的内容，知道 AP (S3C6410) 发出的请求。输出信箱的格式可以根据应用自由定义。当调制

器读取输出信箱时，HOST I/F 内输出信箱标志将自动清除。

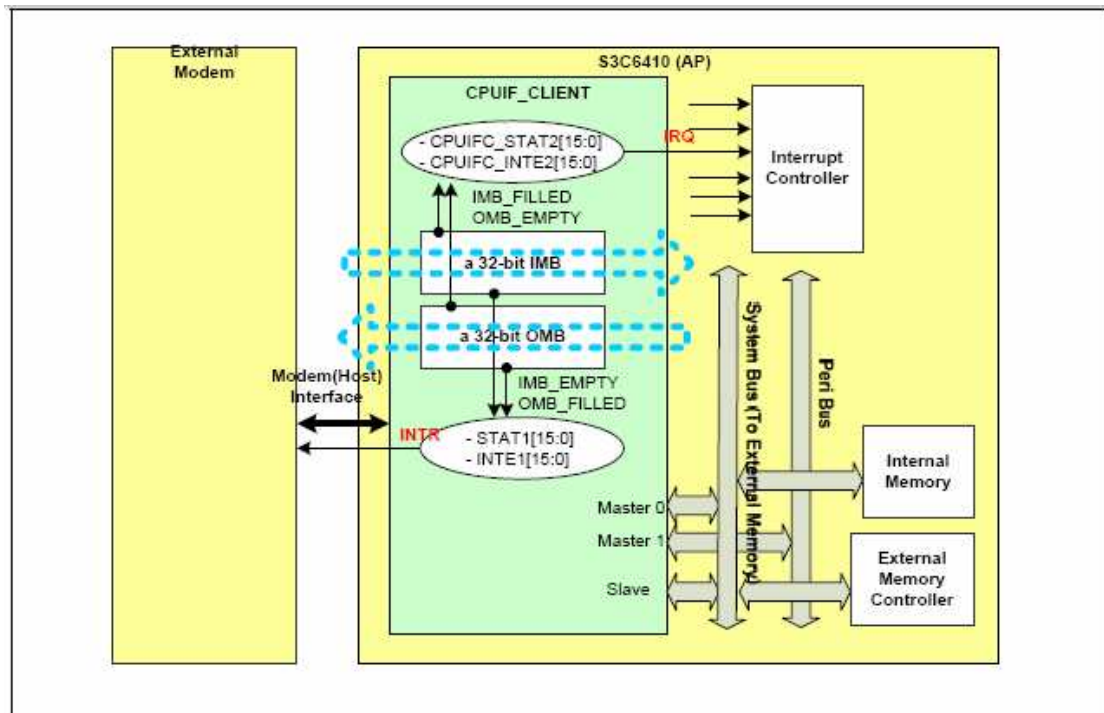


图 24-10 调制器和 AP 之间的输入/输出信箱

## 24.3 编程模型

### 1. 主机接口寄存器可分为：

- (1) (协议) 寄存器：通过调制器访问 16 位主机接口
- (2) 特殊功能寄存器：通过总线主控访问系统总线。

通过 (协议) 寄存器，调制器可以执行以下操作：

- (1) 信号转换
- (2) 突发转换
- (3) 读取 S3C6410 所有地址空间，包括特殊功能寄存器
- (4) 写入 S3C6410 所有地址空间，包括特殊功能寄存器
- (5) 向输入信箱内写入 32 位信息
- (6) 从输出信箱内读取 32 位信息

通过 SFR 可以支持以下操作：

- (1) 向输出寄存器写入 32 位信息
- (2) 从输入信箱内读取 32 位信息

## 24.4 寄存器描述

寄存器	区域	MP_A[1:0]	读/写	描述	复位值
CTRL	0x0	00	读/写	控制寄存器	0x0000
INTE		01	读/写	中断使能寄存器	0x2000
STAT		10	读	状态寄存器	0x90A2
CTRL1	0x1	00	读/写	控制 1 寄存器	0x0000
INTE1		01	读/写	中断使能 1 寄存器	0x0000
STAT1		10	读	状态 1 寄存器	0x0002
IMBL	0x2	00	读/写	输入信箱低位寄存器	0x0000
IMBH		01	读/写	输入信箱高位寄存器	0x0000
OMBL	0x3	00	读	输出信箱低位寄存器	0x0000
OMBH		01	读	输出信箱高位寄存器	0x0000
hDATAH	0x8	00	读/写	主机接口数据低位寄存器	-
hDATAH		01	读/写	主机接口数据高位寄存器	-
SYS_CTRL	0xB	00	读/写	系统控制寄存器	0x0000
Reserved		01	读/写	保留	0x0005
Reserved		10	读	奥六	-
BSEL	所有区域	11	读/写	区域选择寄存器	0x0000

注：MP\_A[1:0]是输入管脚名字 “XhiADDR[1:0]”, MP 表示调制解调器，MP\_A 表示 MP 地址。

### 1.（协议）寄存器矩阵

表 24-1 和表 24-2 列出（协议）寄存器被分成的 16 个板块，在访问之前前，必须正确设置 BSEL[3:0]。

表 24-1 （协议）寄存器 矩阵（板块 1~板块 7）

MP_A[1:0]	（协议）寄存器							
	板块 0	板块 1	板块 2	板块 3	板块 4	板块 5	板块 6	板块 7
	（0000）	（0001）	（0010）	（0011）	（0100）	（0101）	（0110）	（0111）
00	CTRL	CTRL1	IMBL	OMBL	保留	保留	保留	保留
01	INTE	INTE 1	IMBH	OMBH	保留	保留	保留	保留
10	STAT	STAT 1	保留	保留	保留	保留	保留	保留
11	BSEL[3:0]							

表 24-2 （协议）寄存器 矩阵（板块 8~板块 15）

MP_A[1:0]	（协议）寄存器							
	板块 0	板块 1	板块 2	板块 3	板块 4	板块 5	板块 6	板块 7
	（0000）	（0001）	（0010）	（0011）	（0100）	（0101）	（0110）	（0111）
00	hDATAL	保留	保留	SYS-CTRL	保留	保留	保留	保留
01	hDATAH	保留	保留	保留	保留	保留	保留	保留
10	保留	保留	保留	保留	保留	保留	保留	保留
11	BSEL[3:0]							

2.控制寄存器(CTRL)

BSEL[3:0]=0000， MP\_A[1:0]=00， 读/写，复位值=0x0000

领域	位	描述	初始状态
Reserved	[15:13]	保留	00
BLEN[8:0]	[12:4]	突发转换长度  基本单位是一个字节 32 位。最大长度是 256 个字节  0=不 交易  1=单个读或写  N= N-word  注： BLEN 必须是 0，不可以发出新的 HOST I/F 命令	0_0000_0000
REP_WRITE	[3]	重复突发写使能：	0



		0=不 能重复突发写入 1=如果设置 1，突发写入被认为是重复突发写如	
HOSTIF_RESET	[2]	HOST I/F 复位  0=用软复位信号进行复位 1=必须通过软件声明	0
Reserved	[1]	必须固定设置为 00	0
READ_WRITE	[0]	读或写  0=写操作 1=读操作	0

### 3. 中断使能寄存器（INTE）

BSEL[3:0]=0000， MP\_A[1:0]=01， 读/写，复位值=0x2000

领域	位	描述	初始状态
WFIFO-THRES	[15:8]	WFIFO 阈值的空元素  这个值定义了阈值的部分空虚。如，如果 WFIFO-THRES 值为 8，ATAT[7]变成 1，WFIFO 中 16 个或更多的元素是空的。	0x20
WFIFO-PEMPTY	[7]	WFIFO 部分控制中断使能，当 INTE[7]=1, STAT[7]=1 时，发生中断。	0
Reserved	[6:2]	-	0
WFIFO	[1]	WFIFO 中断使能  当 INTE[1]=1, STAT[1]=1 时发生中断	0
RFIFO	[0]	RFIFO 中断使能  当 INTE[0]=1, STAT[0]=1 时发生中断	0

### 4.状态寄存器（STAT）

BSEL[3:0]=0000， MP\_A[1:0]=10， 读/写，复位值=0x90A2

领域	位	描述	初始状态
WRITABLE_CNT	[15:8]	WFIFO 内写入字的数量：只读	0x90

		<p>这个领域可以在没有检测 WFIFO 容量的情况下显示出可以向 WFIFO 内写入多少字，如果是 0x08 领域，主机可以写入 16 个字。</p> <p>注意：复位值是 0x90，在不溢出的情况下可以写入 288 个字。实际上，因为 WFIFO 内包含地址，可接受的最大数据是 256 个字节。</p>	
WFIFO-PEMPTY	[7]	WFIFO 部分空值标志，当空值元素数量大于或等于阈值时，标志变为 1。	0
Reserved	[6:2]	-	0
WFIFO	[1]	WFIFO 部分空值标志： 当突发写入变为空，最大突发写入能力有效的瞬间，标志变为 1。只要 WFIFO 有一个单个写入，标志将复位	1
RFIFO	[0]	RFIFO 读标志 当突发读准备读取数据时，标志变为 1。HOST 读取 DATAH 将复位标志。	0

## 5.中断使能 1 寄存器（INTE1）

BSEL[3:0]=0000， MP\_A[1:0]=10， 读/写，复位值=0x0000

领域	位	描述	初始状态
Reserved	[15:2]		0
IMB_EMPTY	[1]	IMB 空值中断使能 当 INTE[1]=1, STAT[1]=1 时发生中断	0
OMB_FILLED	[0]	OMB 领域中断使能 当 INTE[0]=1, STAT[0]=1 时发生中断	0

## 6.状态 1 寄存器（STAT1）

BSEL[3:0]=0001， MP\_A[1:0]=10， 读/写，复位值=0x0002

领域	位	描述	初始状态
Reserved	[15:2]		0

IMB_EMPTY	[1]	IMB 空值标志： 标示与 IMB_FILLED 值相反	1
OMB_FILLED	[0]	OMB 领域标志： 当通过 SFR 访问写入输出信箱时，标志被设置。当写入 HIGH 值时，标志被清除	0

## 7.输入信箱低位寄存器（IMBL）

BSEL[3:0]=0010， MP\_A[1:0]=00， 读/写，复位值=0x0000

领域	位	描述	初始状态
IMBL	[15:0]	输入信箱寄存器的低 16 位 HOST 向 IMBL 内写入 16 位数据	0x0000

## 8.输入信箱高位寄存器（IMBH）

BSEL[3:0]=0010， MP\_A[1:0]=00， 读/写，复位值=0x0000

领域	位	描述	初始状态
IMBL	[15:0]	输入信箱寄存器的高 16 位 HOST 向 IMBH 内写入 16 位数据后,为了告知 32 位 IMB 包含新值, HOST I/F 需要声明 IMB 标志。当 IMB 通过软件读取时, IMB 标志自动清除。	0x0000

## 9.输出信箱低位寄存器（OMBL）

BSEL[3:0]=0011， MP\_A[1:0]=00， 读/写，复位值=0x0000

领域	位	描述	初始状态
OMBL	[15:0]	输入信箱寄存器的低 16 位 HOST 读取 32 位 OMB 内的低 16 位数据	0x0000

## 10.输出信箱高位寄存器（OMBH）

BSEL[3:0]=0011， MP\_A[1:0]=01， 读/写，复位值=0x0000

领域	位	描述	初始状态
----	---	----	------

OMBH	[15:0]	输出信箱寄存器的高 16 位。 当 HOST 读取 32 位 OMB 内的高 16 位数据时, STAT1[0] 自动清除。	0x0000
------	--------	---	--------

### 11.主机接口数据低位寄存器（HDATAL）

BSEL[3:0]=1000, MP\_A[1:0]=01, 读/写, 复位值=未定义

领域	位	描述	初始状态
DATAL	[15:0]	数据寄存器	—

### 12.主机接口数据高位寄存器（HDATAH）

BSEL[3:0]=1000, MP\_A[1:0]=01, 读/写, 复位值=未定义

领域	位	描述	初始状态
DATAH	[15:0]	数据寄存器	—

### 13.系统控制寄存器（SYS\_CTRL）

BSEL[3:0]=1011, MP\_A[1:0]=00, 读/写, 复位值=0x0000

领域	位	描述	初始状态
Reserved	[15:1]	数据寄存器	0
BOOTDONE	[0]	Boot done for Modem Booting 当此位设置为高电平时, 系统控制器判断 Boot done 信号。S3C6410 (AP) 开始 Boot 操作。 完成 Modem boot 以后, 此位必须清零。	0

### 14.板块选择寄存器（BSEL）

BSEL[3:0]=1011, MP\_A[1:0]=00, 读/写, 复位值=0x0000

领域	位	描述	初始状态
Reserved	[15:4]		0x0000
BSEL	[3:0]	板块选择 0000=板块 0 内的一个（协议）寄存器被选择	0000

		0001=板块 1 内的一个（协议）寄存器被选择	
		.....	
		1111=板块 15 内的一个（协议）寄存器被选择	

## 24.5 特殊功能寄存器描述

基本地址：0x7400\_0000

寄存器	补偿区	读/写	描述	复位值
HOSTIFC_CTRL	0x000	读/写	HOST I/F 控制寄存器	0x20FF_0100
Reserved	0x004	读/写	保留	0x0000_0006
HOSTIFC_TMP	0x008	读/写	HOST I/F 临时寄存器	0x0000_0000
Reserved	0x00C	-	保留	0x0000_0000
HOSTIFC_IMB	0x010	读	HOST I/F IMB 寄存器	0x0000_0000
HOSTIFC_OMB	0x014	读/写	HOST I/F OMB 寄存器	0x0000_0000
HOSTIFC_MR_STAT	0x020	读	HOST I/F 状态镜像寄存器	0x0000_9A02
HOSTIFC_MR_STAT1	0x024	读	HOST I/F 状态镜像 1 寄存器	0x0000_0002
HOSTIFC_STAT2	0x028	读/写	HOST I/F 状态 2 寄存器	0x0001_0000
Reserved	0x02C	-	保留	0x0000_0000
HOSTIFC_MR_INTE	0x030	读	HOST I/F 中断使能寄存器	0x0000_2000
HOSTIFC_MR_INTE1	0x034	读	HOST I/F 中断使能 1 寄存器	0x0000_0000
HOSTIFC_INTE 2	0x038	读/写	HOST I/F 中断使能 2 寄存器	0x0001_0000
Reserved	0x03C	-	保留	0x0000_0000

### 1.HOST I/F 控制寄存器（HOSTIFC\_CTRL）

寄存器	地址	读/写	描述	复位值
HOSTIFC_CTRL	0x74000000	读/写	HOST I/F 控制寄存器	0x20FF0100

HOSTIFC_CTRL	位	描述	初始状态
Reserved	[31:30]	–	0
INV_INIR	[29]	INTR 的极性反转  0: INTR 高电平有效，当发生中断时 INTR 变成高电平 1: INTR 低电平有效，当发生中断时 INTR 变成低电平  注: HOST I/F 模块的“INV_INTR”和 MODEM I/F 模块的“INT2M_LEVEL”极性必须相同	1
Reserved	[28:24]	–	0x0
Reserved	[23:16]	–	0xFF
Reserved	[15:9]	–	0x0
Reserved	[8:0]	–	0x100

## 2.HOST I/F 临时寄存器 (HOSTIFC\_TMP)

寄存器	地址	读/写	描述	复位值
HOSTIFC_TMP	0x74000008	读/写	HOST I/F 临时寄存器	0x000000

HOSTIFC_TMP	位	描述	初始状态
DATA	[31:0]	临时寄存器  临时寄存器可以用来设计版本或者用来验证	0x0000_0000

## 3.HOST I/F IMB 寄存器 (HOSTIFC\_IMB)

寄存器	地址	读/写	描述	复位值
HOSTIFC_IMB	0x74000010	读	HOST I/F IMB 寄存器	0x00000000

HOSTIFC_IMB	位	描述	初始状态
IMB	[31:0]	32 位输入信箱 Shadow 寄存器	0x0000_0000

#### 4. HOST I/F OMB 寄存器 (HOSTIFC\_OMB)

寄存器	地址	读/写	描述	复位值
HOSTIFC_OMB	0x74000014	读	HOST I/F OMB 寄存器	0x00000000

HOSTIFC_OMB	位	描述	初始状态
OMB	[31:0]	32 位输出信箱寄存器	0x0000_0000

#### 5. HOST I/F 状态镜像寄存器 (HOSTIFC\_MR\_STAT)

寄存器	地址	读/写	描述	复位值
HOSTIFC_MR_STAT	0x74000020	读	HOST I/F 状态镜像寄存器	0x000090A2

HOSTIFC_MR_STAT	位	描述	初始状态
Reserved	[31:16]	–	0x0000
STAT	[15:0]	STAT[15:0]镜像（协议）寄存器	0x90A2

#### 6. HOST I/F 状态 1 镜像寄存器 (HOSTIFC\_MR\_STAT1)

寄存器	地址	读/写	描述	复位值
HOSTIFC_MR_STAT1	0x74000024	读	HOST I/F 状态 1 镜像寄存器	0x000090A2

HOSTIFC_MR_STAT1	位	描述	初始状态
Reserved	[31:16]	–	0x0000
STAT1	[15:0]	STAT1[15:0]镜像（协议）寄存器	0x0002

#### 7. HOST I/F 状态 2 寄存器 (HOSTIFC\_MR\_STAT2)

寄存器	地址	读/写	描述	复位值
HOSTIFC_STAT2	0x74000028	读	HOST I/F 状态 2 寄存器	0x000090A2

HOSTIFC_STAT2	位	描述	初始状态
Reserved	[31:20]	–	0x000
Reserved	[19]	–	0
RBURST_DONE	[18]	重复突发写操作标志 当进行重复突发写操作时，将设置此标志。写入 HIGH 值时，将清除该标志。	0
IMB_FILLED	[17]	IMB 领域标志 通过调制器写入输入信箱时，将设置此标志。写入 HIGH 值时，将清除该标志。	0
OMB-EMPTY	[16]	OMB 空值标志 此标志是 OMB_FILLED 的反相值。	1
Reserved	[15:8]	保留	0x00
RX_FIFO_OVER_RUN	[7]	HOST I/F 内的 当地 RX FIFO 过度运行标志	0
RX_FIFO_UNDER_RUN	[6]	HOST I/F 内的 当地 RX FIFO 欠载运行标志	0
TX_FIFO_OVER_RUN	[5]	HOST I/F 内的 当地 TX FIFO 过度运行标志	0
TX_FIFO_UNDER_RUN	[4]	HOST I/F 内的 当地 TX FIFO 欠载运行标志	0
Reserved	[3]	–	0
Reserved	[2]	–	0
Reserved	[1]	–	0
Reserved	[0]	–	0

## 8 HOST I/F 中断使能镜像寄存器（HOSTIFC\_MR\_INTE）

寄存器	地址	读/写	描述	复位值
HOSTIFC_MR_INTE	0x74000030	读	HOST I/F 中断使能镜像寄存器	0x00002000

HOSTIFC_MR_INTE	位	描述	初始状态
Reserved	[31:16]	–	0x0000
INTE	[15:0]	INTE[15:0] 镜像（协议）寄存器	0x2000



9. HOST I/F 中断使能 1 镜像寄存器 (HOSTIFC\_MR\_INTE1)

寄存器	地址	读/写	描述	复位值
HOSTIFC_MR_INTE1	0x74000034	读	HOST I/F 中断使能 1 镜像寄存器	0x00000000

HOSTIFC_MR_INTE1	位	描述	初始状态
Reserved	[31:16]	–	0x0000
INTE1	[15:0]	INTE1[15:0] 镜像（协议）寄存器	0x0000

10. HOST I/F 中断使能 2 镜像寄存器 (HOSTIFC\_INTE2)

寄存器	地址	读/写	描述	复位值
HOSTIFC_INTE2	0x74000034	读	HOST I/F 中断使能 1 镜像寄存器	0x00000000

HOSTIFC_INTE2	位	描述	初始状态
Reserved	[31:16]	–	0x0000
INTE2	[15:0]	INTE2[15:0] 镜像（协议）寄存器	0x0000

# 25 USB 主机控制器

本节主要介绍在 S3C6410 RISC 微处理器上，通用串行总线主机控制器（USB）的执行。

## 25.1 USB 主机控制器概述

S3C6410 支持 2 端口 USB 主机接口如下：

- 兼容 OHCI Rev1.0。
- 兼容 USB Rev1.1。
- 两个向下传输端口。
- 支持低速和全速 USB 设备。

其 USB 主机控制器结构框图，如图 25-1 所示。

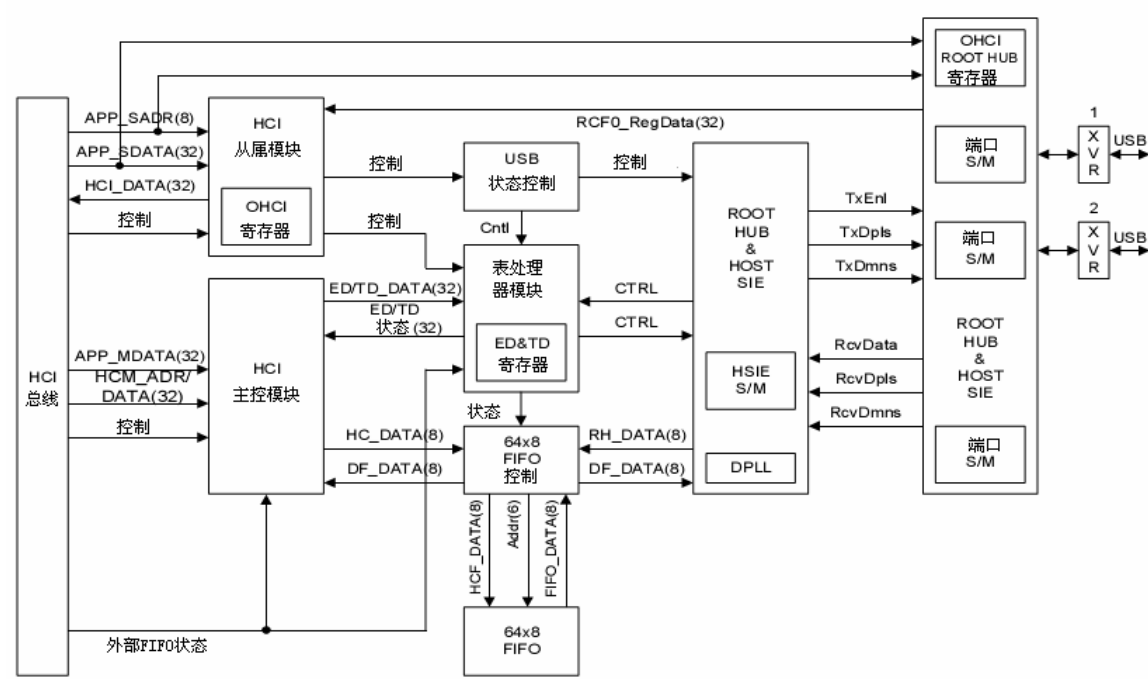


图 25-1 USB 主机控制器结构框图

## 25.2 USB 主机控制器特殊寄存器

该 S3C6410 USB 主机控制器符合 OHCI Rev1.0，参照开放式主机控制器接口 Rev1.0，如表 25-1 所示，显示了 USB 主机控制器的 OHCI 控制器。

表 25-1 USB 主机控制器的 OHCI 控制器

寄存器	基地址	读/写	描述	初始值
HcRevision	0x74300000	读	USB 主机控制器修改寄存器。	0x0000_0010
HcControl	0x74300004	读/写	USB 主机控制器控制寄存器。	0x0000_0000
HcCommonStatus	0x74300008	读/写	USB 主机控制器命令状态寄存器。	0x0000_0000
HcInterruptStatus	0x7430000C	读/写	USB 主机控制器中断状态寄存器。	0x0000_0000
HcInterruptEnable	0x74300010	读/写	USB 主机控制器中断启动寄存器。	0x0000_0000
HcInterruptDisable	0x74300014	读/写	USB 主机控制器中断禁止寄存器。	0x0000_0000
HcHCCA	0x74300018	读/写	USB 主机控制器 HCCA 寄存器。	0x0000_0000
HcPeriodCuttentED	0x7430001C	读	USB 主机控制器当前周期 ED 寄存器。	0x0000_0000
HcControlHeadED	0x74300020	读/写	USB 主机控制器主要 ED 寄存器。	0x0000_0000
HcControlCurrentED	0x74300024	读/写	USB 主机控制器当前控制 ED 寄存器。	0x0000_0000
HcBulkHeadED	0x74300028	读/写	USB 主机控制器主要批量 ED 寄存器。	0x0000_0000
HcBulkCurrentED	0x7430002C	读/写	USB 主机控制器当前批量 ED 寄存器。	0x0000_0000
HcDoneHead	0x74300030	读	USB 主机控制器 Done Head 寄存器。	0x0000_0000
HcRmInterval	0x74300034	读/写	USB 主机控制器调频时间间隔寄存器。	0x0000_2EDF
HcFmRemaining	0x74300038	读	USB 主机控制器保持的帧寄存器。	0x0000_0000
HcFmNumber	0x7430003C	读	USB 主机控制器帧数目寄存器。	0x0000_0000
HcPeriodicStart	0x74300040	读/写	USB 主机控制器定期启动寄存器。	0x0000_0000
HcLSThreshold	0x74300044	读/写	USB 主机控制器低速极限寄存器。	0x0000_0628
HcRhDescriptorA	0x74300048	读/写	USB 主机控制器根集线器描述符寄存器 A。	0x0200_1202
HcRhDescriptorB	0x7430004C	读/写	USB 主机控制器根集线器描述符寄存器 B。	0x0000_0000
HcRhStatus	0x74300050	读/写	USB 主机控制器根集线器端口状态寄存器。	0x0000_0000
HcRhPortStatus1	0x74300054	读/写	USB 主机控制器根集线器端口状态 1 寄存	0x0000_0100
HcRhPortStatus2	0x74300058	读/写	USB 主机控制器根集线器端口状态 2 寄存	0x0000_0100

# USB 2.0 高速 OTG

## 26.1 概述

三星 USB 接口 OTG 是一个双角色的设备控制器，可以支持设备和主机两种功能。它支持高速（HS,480Mbps）、全速（FS,12Mbps，只用于设备）、以及低速（LS,1.5Mbps,只用于主机）转换。高速 OTG 可以作为主机或设备控制器。

USB2.0 高速 OTG 的主要性能包括：

- （1）符合补充的 SUB 2.0 OTG 规范
- （2）操作在高速（480Mbps）、全速（12Mbps，只用于设备）和低速（1.5Mbps，只用于主机）模式
- （3）支持 UTMI+level 3 接口
- （4）支持 SRP 和 HNP
- （5）在 AHB 上，只支持 32 位数据
- （6）控制转换机上有一个控制端点 0
- （7）15 个设备模式可编程端点：
  - 可编程端点类型：块类型，同步类型，或中断类型。
  - 可编程输入/输出方向
- （8）支持 16 个主机通道
- （9）支持基础包，动态 FIFO 存储器分配（6144 depths，35-bit width）

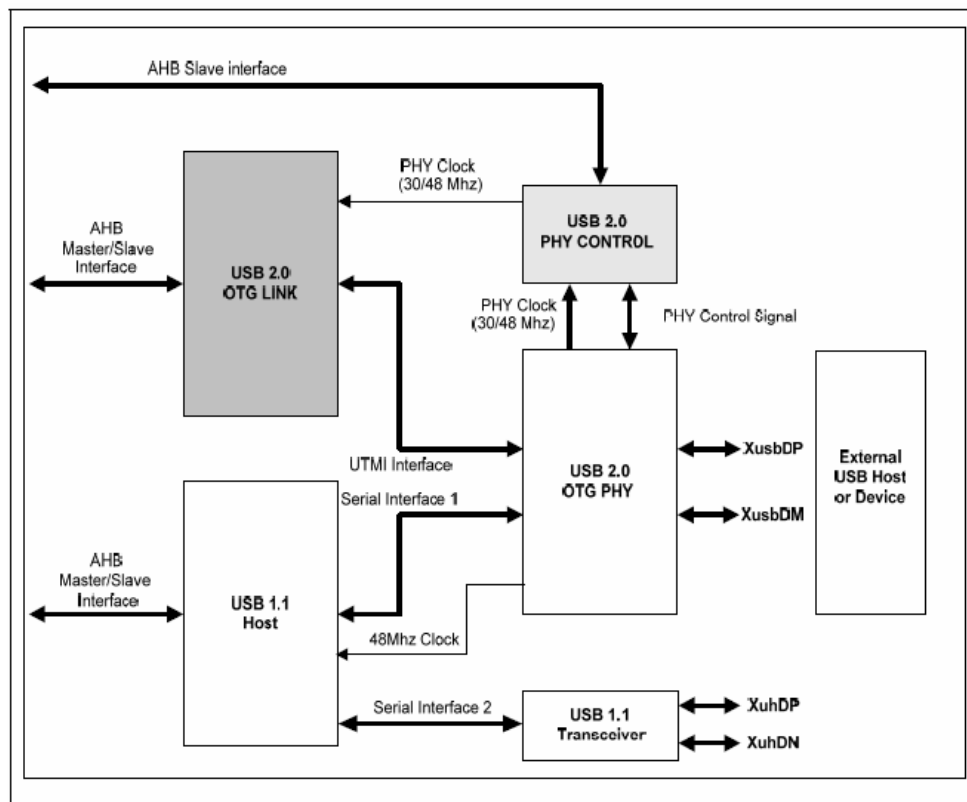


图 26-1 系统级框图

高速 OTG 控制器由两个独立的模块组成, 分别是 USB2.0 OTG 链接核心和 USB 2.0 PHY 控制两部分。每个模块都有一个 AHB Slave, 可以支持微型控制器对控制寄存器和状态寄存器的读写访问。OTG 连接有一个 AHB Mster, 可以使能连接, 在 AHB 上进行数据转换。

在上图所示中的 S3C6400X USB 系统可以根据下面所述进行配置:

- (1) USB 1.1 主机 1 端口和 USB2.0 OTG 1 端口
- (2) USB 1.1 主机 2 端口

为了使能串行接口 1 和使用 2 个主机 1.1 端口, 需要设置 OPHYCLK. 串行模式寄存器位为 1.

### 1. 操作模式

实际应用中, 可以在 DMA 模式下和 Slave 模式下操作链接, 不可以同时使用 DMA 和 Slave 模式运行核心。

- (1) DMA 模式

USB OTG 主机用 AHB master 接口转换包数据, 以及进行接收数据更新。AHB master 用程式化的

DMA 地址访问数据缓冲区。

(2) Slave 模式

USB OTG 可以在 transaction-level 操作和 Pipelined transaction-level 操作下运行。在 transaction-level 操作中，应用可以在每个通道或端口处理一个数据包。在 Pipelined transaction-level 操作中，应用可以运行 OTG 进行多重转换操作。Pipelined 操作功能的提高主要指在包基础上的应用不再需要中断。

2. 系统控制器设置

一个系统控制器内的寄存器应该被设置成为 USB 接口进行适当的工作。

OTHERS	位	读/写	描述	初始状态
USB_SIG_MASK	[16]	读_写	USB 信号屏蔽阻止不希望的泄露	1'b0

在地址 7E00\_F900h 基础上的 OTHER 控制寄存器的第 16 位，根据系统的操作模式被进行不同模式的设置。

(1) 常规模式

USB\_SIG\_MASK 的初始状态是 1'b0。进行开始 USB 转换操作时需要将此位设置为 1'b1。

在常规模式下，如果未使用 USBOTG 功能，USB PHY 电源可以被切断。

(2) 停止/深度停止/睡眠模式

在这些操作模式下，USB PHY 电源可以被切断。因此为了阻止泄露电流，必须在进入这些模式之前设置 USB\_SIG\_MASK 为 1'b0。

26.2 寄存器映射

访问基于地址 7C10\_0000H 上的 OTG PHY 控制寄存器可以控制和监视 OTG PHY。

基于地址 7C00\_0000H OTG 上的链接核心寄存器根据下面所述进行分类：

- (1) 核心全局寄存器
- (2) 主机模式寄存器：主机全局寄存器，主机端口 CSRs，主机特殊通道寄存器
- (3) 设备模式寄存器：设备全局寄存器，设备特殊端点寄存器

只有核心全局寄存器和主机端口寄存器可以在主机模式和设备模式下被访问。当 OTG 链接在设备模式或主机模式下运行时，实际应用不能从其他模式访问寄存器。如果出现非法访问，将产生模式不匹配中断，并且在核心中断寄存器内反射出来。当核心从一种模式转换为另一种模式时，新模式下寄存器的操作

必须被重新程式化，就像在电源开始复位以后一样。

1. OTG 链接 CSR 存储器映射

主机和设备模式寄存器占用不同的地址，所有寄存器都在 AHB 时钟范围内被执行。图 26-2 显示出 OTG 链接 CSR 地址映射。

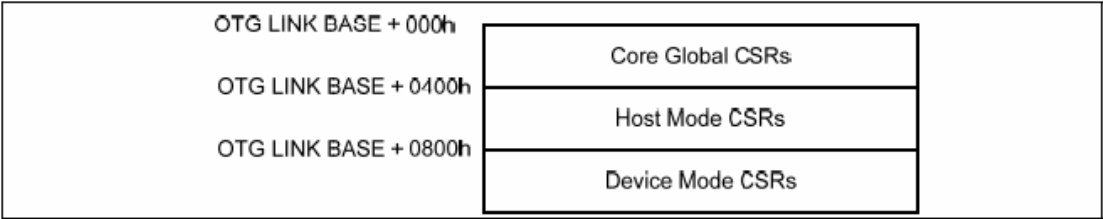


图 26-2 OTG 链接 CSR 存储器映射

2. OTG FIFO 地址映射

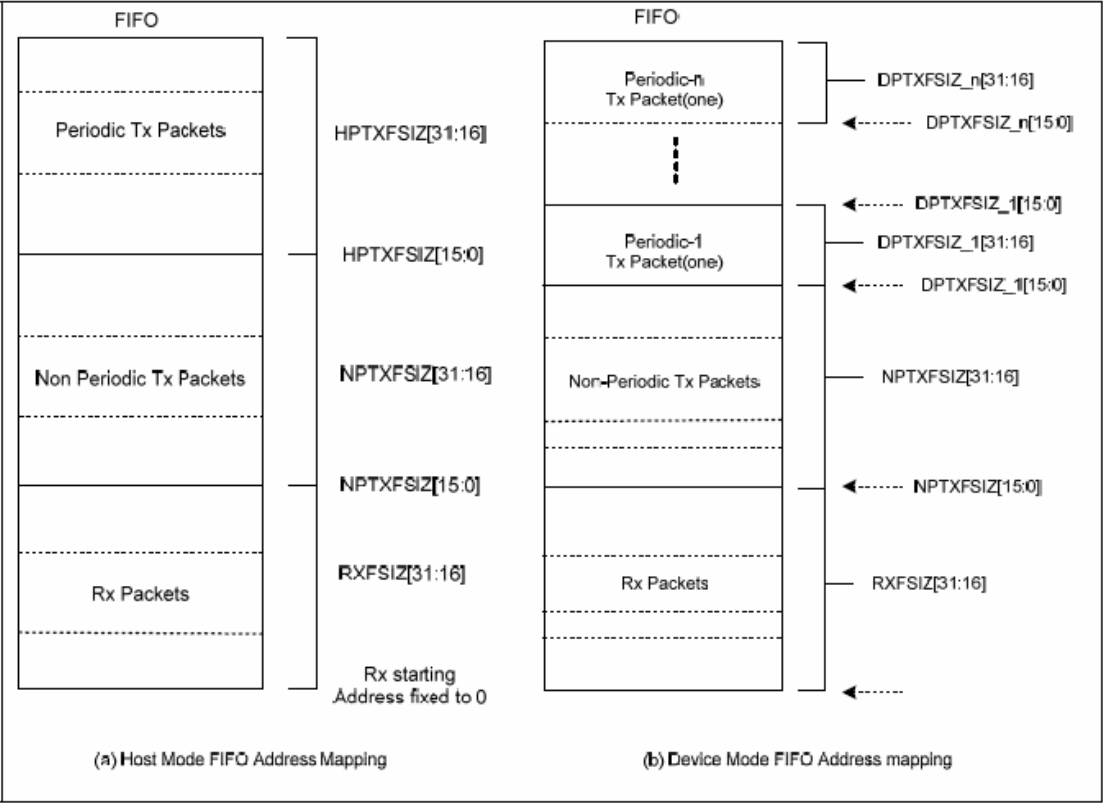


图 26-3 OTG FIFO 映射

图 26-3 中显示了 OTG FIFO 地址映射关系。下面所述寄存器必须根据所示内容进行编程  
主机模式下：

```
RXFSIZ[15:0] = OTG_RX_DFIFO_DEPTH
NPTXFSIZ[15:0] = OTG_RX_DFIFO_DEPTH
NPTXFSIZ[31:16] = OTG_TX_NPERIO_DFIFO_DEPTH
HPTXFSIZ[15:0] = RX_DFIFO_DEPTH+TX_NPERIO_DFIFO_DEPTH
HPTXFSIZ[31:16] = TX_PERIOD_DFIFO_DEPTH
```

设备模式下：

```
RXFSIZ[15:0] = OTG_RX_DFIFO_DEPTH
NPTXFSIZ[15:0] = OTG_RX_DFIFO_DEPTH
NPTXFSIZ[31:16] = OTG_TX_NPERIO_DFIFO_DEPTH
DPTXFSIZ_1[15:0] = OTG_RX_DFIFO_DEPTH+OTG_TX_NPERIO_DFIFO_DEPTH
DPTXFSIZ_1[31:16] = OTG_TX_PERIOD_DFIFO_DEPTH_1
DPTXFSIZ_2[15:0] = DPTXFSIZ_1[15:0]+OTG_TX_NPERIO_DFIFO_DEPTH_1
DPTXFSIZ_2[31:16] = OTG_TX_PERIOD_DFIFO_DEPTH_2
.....
DPTXFSIZ_n[15:0] = DPTXFSIZ_n-1[15:0]+OTG_TX_NPERIO_DFIFO_DEPTH_n-1
DPTXFSIZ_n[31:16] = OTG_TX_PERIOD_DFIFO_DEPTH_n
```

### 3. CSRS 应用访问

每个寄存器访问列描述下面所述特殊功能，包括如何应用，以及中心可以访问的 CSRs 的寄存器领域。

Read Only	RO	应用当中寄存器区域只可以进行读操作，此领域内的写操作无效
Write Only	WO	应用当中寄存器区域只可以进行写操作
READ and Write	R_W	应用当中在寄存器区域内可以进行读和写操作。并且可以设置写入 1'b1 开始操作，设置写入 1'b0 进行清零。
Read Write and Self Clear	R_W_SC	应用当中在寄存器区域内可以进行读和写操作。通过核心（自动清零）。



Read, Write, Self set and Self Clear	R_W_SS_SC	应用当中在寄存器区域内可以进行读和写操作。 在一定的 USB 情况下可以通过核心(自动设置)设置到 1'b1, 通过核心（自动清零）清零到 1'b0.
Read, Self set and Write Clear	R_SS_WC	应用当中在寄存器区域内可以进行读和写操作。 在某些内部或 USB 或 AHB 情况下可以自动设置到 1'b1, 并可以自动清除到 1'b0。应用中不可以清除这种领域类型, 向此位写入 1'b0 对此领域没有影响。
Read, Self set and Clear	R_SS_SC_WC	应用当中在寄存器区域内可以进行读和写 Self Clear or Write 操作在某些内部或 USB 或 AHB 情况下可以自动设置到 1'b 1, 并可以自动清除到 1'b0, 或通过系那个寄存器内写入 1'b1 进行清除。 向此位写入 1'b0 对此领域没有影响。

## 26.3 高速 OTG 控制器特殊寄存器

寄存器	补偿区	读/写	描述	复位值
OTG PHY 控制寄存器（基础地址：0x7C10_0000）				
OPHY_PWR	0x000	读/写	OTG PHY 电源控制寄存器	0x0000_0019
OPHY_CLK	0x004	读/写	OTG PHY 时钟控制寄存器	0x0000_0000
ORSTCON	0x008	读/写	OTG 复位控制寄存器	0x0000_0001
OPHY_TUNE	0x020	读_写	OTG PHY 调谐寄存器	0x0027_1B93
OTG 链接核心寄存器（基础地址：0x7C00_0000）				
核心全局寄存器				
GOTGCTL	0x000	读/写	OTG 控制和状态寄存器	0x0001_0000
GOTGINT	0x004	读/写	OTG 中断寄存器	0x0000_0000
GAHB_CFG	0x008	读/写	核心 AHB 配置寄存器	0x0000_0000

GUSBCFG	0x00C	读/写	核心 UHB 配置寄存器	0x0000_1400
GRSTCTL	0x010	读/写	核心复位寄存器	08000_0000
GINTSTS	0x014	读/写	核心中断寄存器	0x0400_1020
GINTMSK	0x018	读/写	核心中断主寄存器	0x0000_0000
GRXSTSR	0x01C	读	核心状态调谐读寄存器	-
GRXSTSR	0x020	读	设备状态读/POP 寄存器	-
GRXFSIZ	0x024	读/写	设备 FIFO 尺寸寄存器	0x0000_1800
GNPTXFSIZ	0x028	读/写	非周期传播 FIFO 尺寸寄存器	0x1800_1800
GNPTXSTS	0x02C	读	非周期传播 FIFO/Queue 状态寄存器	0x0008_1800
HPTXFSIZ	0x100	读/写	主机周期传播 FIFO 尺寸寄存器	0x0300_5A00
DPTXFSIZ1	0x104	读/写	设备周期传播 FIF-1 尺寸寄存器	0x0300_1000
DPTXFSIZ2	0x108	读/写	设备周期传播 FIF-2 尺寸寄存器	0x0300_3300
DPTXFSIZ3	0x10C	读/写	设备周期传播 FIF-3 尺寸寄存器	0x0300_3600
DPTXFSIZ4	0x110	读/写	设备周期传播 FIF-4 尺寸寄存器	0x0300_3900
DPTXFSIZ5	0x114	读/写	设备周期传播 FIF-5 尺寸寄存器	0x0300_3C00
DPTXFSIZ6	0x118	读/写	设备周期传播 FIF-6 尺寸寄存器	0x0300_3F00
DPTXFSIZ7	0x11C	读/写	设备周期传播 FIF-7 尺寸寄存器	0x0300_4200
DPTXFSIZ8	0x120	读/写	设备周期传播 FIF-8 尺寸寄存器	0x0300_4500
DPTXFSIZ9	0x124	读/写	设备周期传播 FIF-9 尺寸寄存器	0x0300_4800
DPTXFSIZ10	0x128	读/写	设备周期传播 FIF-10 尺寸寄存器	0x0300_4B00
DPTXFSIZ11	0x12C	读/写	设备周期传播 FIF-11 尺寸寄存器	0x0300_4E00
DPTXFSIZ12	0x130	读/写	设备周期传播 FIF-12 尺寸寄存器	0x0300_5100
DPTXFSIZ13	0x134	读/写	设备周期传播 FIF-13 尺寸寄存器	0x0300_5400
DPTXFSIZ14	0x138	读/写	设备周期传播 FIF-1 4 尺寸寄存器	0x0300_5700
DPTXFSIZ15	0x13C	读/写	设备周期传播 FIF-1 5 尺寸寄存器	0x0300_5A00
主机模式寄存器				
主机全局寄存器				

HCFG	0x400	读/写	主机配置寄存器	0x0020_0000
HFIR	0x404	读/写	主机帧间隔寄存器	0x0000_17D7
HFNUM	0x408	读	主机帧数量/帧时间持续寄存器	0x000_0000
HPTXSTS	0x410	读	主机周期传播 FIFO/Queue 状态寄存器	0x0008_0100
HAINT	0x414	读	主机所有通道中断寄存器	0x0000_0000
HAINTMAK	0x418	读/写	主机所有通道中断屏蔽寄存器	0x0020_0000
主机端口控制和状态寄存器				
HPRT	0x440	读/写	主机端口控制和状态寄存器	0x0000_0000
主机特殊通道寄存器				
HCCHAR0	0x500	读/写	主机通道 0 特性寄存器	0x0000_0000
HCSPLT0	0x504	读/写	主机通道 0 分裂控制寄存器	0x0000_0000
HCINT0	0x508	读/写	主机通道 0 中断寄存器	0x0000_0000
HCINTMASK0	0x50C	读/写	主机通道 0 中断屏蔽寄存器	0x0000_0000
HCTSIZE0	0x510	读/写	主机通道 0 转换尺寸寄存器	0x0000_0000
HCDMA0	0x514	读/写	主机通道 0 DMA 地址寄存器	0x0000_0000
HCCHAR1	0x520	读/写	主机通道 1 特性寄存器	0x0000_0000
HCSPLT1	0x524	读/写	主机通道 1 分裂控制寄存器	0x0000_0000
HCINT1	0x528	读/写	主机通道 1 中断寄存器	0x0000_0000
HCINTMASK1	0x52C	读/写	主机通道 1 中断屏蔽寄存器	0x0000_0000
HCTSIZE1	0x530	读/写	主机通道 1 转换尺寸寄存器	0x0000_0000
HCDMA1	0x534	读/写	主机通道 1DMA 地址寄存器	0x0000_0000
HCCHAR2	0x540	读/写	主机通道 2 特性寄存器	0x0000_0000
HCSPLT2	0x544	读/写	主机通道 2 分裂控制寄存器	0x0000_0000
HCINT2	0x548	读/写	主机通道 2 中断寄存器	0x0000_0000
HCINTMASK2	0x54C	读/写	主机通道 2 中断屏蔽寄存器	0x0000_0000
HCTSIZE2	0x550	读/写	主机通道 2 转换尺寸寄存器	0x0000_0000
HCDMA2	0x554	读/写	主机通道 2DMA 地址寄存器	0x0000_0000

HCCHAR3	0x560	读/写	主机通道 3 特性寄存器	0x0000_0000
HCSPLT3	0x564	读/写	主机通道 3 分裂控制寄存器	0x0000_0000
HCINT3	0x568	读/写	主机通道 3 中断寄存器	0x0000_0000
HCINTMASK3	0x56C	读/写	主机通道 3 中断屏蔽寄存器	0x0000_0000
HCTSIZE3	0x570	读/写	主机通道 3 转换尺寸寄存器	0x0000_0000
HCDMA3	0x574	读/写	主机通道 3DMA 地址寄存器	0x0000_0000
HCCHAR4	0x580	读/写	主机通道 4 特性寄存器	0x0000_0000
HCSPLT4	0x584	读/写	主机通道 4 分裂控制寄存器	0x0000_0000
HCINT4	0x588	读/写	主机通道 4 中断寄存器	0x0000_0000
HCINTMASK4	0x58C	读/写	主机通道 4 中断屏蔽寄存器	0x0000_0000
HCTSIZE4	0x590	读/写	主机通道 4 转换尺寸寄存器	0x0000_0000
HCDMA4	0x594	读/写	主机通道 4 DMA 地址寄存器	0x0000_0000
HCCHAR5	0x5A0	读/写	主机通道 5 特性寄存器	0x0000_0000
HCSPLT5	0x5A4	读/写	主机通道 5 分裂控制寄存器	0x0000_0000
HCINT5	0x5A8	读/写	主机通道 5 中断寄存器	0x0000_0000
HCINTMASK5	0x5AC	读/写	主机通道 5 中断屏蔽寄存器	0x0000_0000
HCTSIZE5	0x5B0	读/写	主机通道 5 转换尺寸寄存器	0x0000_0000
HCDMA5	0x5B4	读/写	主机通道 5DMA 地址寄存器	0x0000_0000
HCCHAR6	0x5C0	读/写	主机通道 6 特性寄存器	0x0000_0000
HCSPLT6	0x5C4	读/写	主机通道 6 分裂控制寄存器	0x0000_0000
HCINT6	0x5C8	读/写	主机通道 6 中断寄存器	0x0000_0000
HCINTMASK6	0x5CC	读/写	主机通道 6 中断屏蔽寄存器	0x0000_0000
HCTSIZE6	0x5D0	读/写	主机通道 6 转换尺寸寄存器	0x0000_0000
HCDMA6	0x5D4	读/写	主机通道 6DMA 地址寄存器	0x0000_0000
HCCHAR7	0x5E0	读/写	主机通道 7 特性寄存器	0x0000_0000
HCSPLT7	0x5E4	读/写	主机通道 7 分裂控制寄存器	0x0000_0000
HCINT7	0x5E8	读/写	主机通道 7 中断寄存器	0x0000_0000
HCINTMASK7	0x5EC	读/写	主机通道 7 中断屏蔽寄存器	0x0000_0000

HCTSIZ7	0x5F0	读/写	主机通道 7 转换尺寸寄存器	0x0000_0000
HCDMA7	0x5F4	读/写	主机通道 7DMA 地址寄存器	0x0000_0000
HCCHAR8	0x600	读/写	主机通道 8 特性寄存器	0x0000_0000
HCSPLT8	0x604	读/写	主机通道 8 分裂控制寄存器	0x0000_0000
HCINT8	0x608	读/写	主机通道 8 中断寄存器	0x0000_0000
HCINTMASK8	0x60C	读/写	主机通道 8 中断屏蔽寄存器	0x0000_0000
HCTSIZ8	0x610	读/写	主机通道 8 转换尺寸寄存器	0x0000_0000
HCDMA8	0x614	读/写	主机通道 8DMA 地址寄存器	0x0000_0000
HCCHAR9	0x620	读/写	主机通道 9 特性寄存器	0x0000_0000
HCSPLT9	0x624	读/写	主机通道 9 分裂控制寄存器	0x0000_0000
HCINT9	0x628	读/写	主机通道 9 中断寄存器	0x0000_0000
HCINTMASK9	0x62C	读/写	主机通道 9 中断屏蔽寄存器	0x0000_0000
HCTSIZ9	0x630	读/写	主机通道 9 转换尺寸寄存器	0x0000_0000
HCDMA9	0x634	读/写	主机通道 9DMA 地址寄存器	0x0000_0000
HCCHAR10	0x640	读/写	主机通道 10 特性寄存器	0x0000_0000
HCSPLT10	0x644	读/写	主机通道 10 分裂控制寄存器	0x0000_0000
HCINT10	0x648	读/写	主机通道 10 中断寄存器	0x0000_0000
HCINTMASK10	0x64C	读/写	主机通道 10 中断屏蔽寄存器	0x0000_0000
HCTSIZ10	0x650	读/写	主机通道 10 转换尺寸寄存器	0x0000_0000
HCDMA10	0x654	读/写	主机通道 10DMA 地址寄存器	0x0000_0000
HCCHAR11	0x660	读/写	主机通道 11 特性寄存器	0x0000_0000
HCSPLT11	0x664	读/写	主机通道 11 分裂控制寄存器	0x0000_0000
HCINT11	0x668	读/写	主机通道 11 中断寄存器	0x0000_0000
HCINTMASK11	0x66C	读/写	主机通道 11 中断屏蔽寄存器	0x0000_0000
HCTSIZ11	0x670	读/写	主机通道 11 转换尺寸寄存器	0x0000_0000
HCDMA11	0x674	读/写	主机通道 11DMA 地址寄存器	0x0000_0000
HCCHAR12	0x680	读/写	主机通道 12 特性寄存器	0x0000_0000
HCSPLT12	0x684	读/写	主机通道 12 分裂控制寄存器	0x0000_0000

HCINT12	0x688	读/写	主机通道 12 中断寄存器	0x0000_0000
HCINTMASK12	0x68C	读/写	主机通道 12 中断屏蔽寄存器	0x0000_0000
HCTSIZ12	0x690	读/写	主机通道 12 转换尺寸寄存器	0x0000_0000
HCDMA12	0x694	读/写	主机通道 12DMA 地址寄存器	0x0000_0000
HCCHAR13	0x6A0	读/写	主机通道 13 特性寄存器	0x0000_0000
HCSPLT13	0x6A4	读/写	主机通道 13 分裂控制寄存器	0x0000_0000
HCINT13	0x6A8	读/写	主机通道 13 中断寄存器	0x0000_0000
HCINTMASK13	0x6AC	读/写	主机通道 13 中断屏蔽寄存器	0x0000_0000
HCTSIZ13	0x6B0	读/写	主机通道 13 转换尺寸寄存器	0x0000_0000
HCDMA13	0x6B4	读/写	主机通道 13DMA 地址寄存器	0x0000_0000
HCCHAR14	0x6C0	读/写	主机通道 14 特性寄存器	0x0000_0000
HCSPLT14	0x6C4	读/写	主机通道 14 分裂控制寄存器	0x0000_0000
HCINT14	0x6C8	读/写	主机通道 14 中断寄存器	0x0000_0000
HCINTMASK14	0x6CC	读/写	主机通道 14 中断屏蔽寄存器	0x0000_0000
HCTSIZ14	0x6D0	读/写	主机通道 14 转换尺寸寄存器	0x0000_0000
HCDMA14	0x6D4	读/写	主机通道 14DMA 地址寄存器	0x0000_0000
HCCHAR15	0x6E0	读/写	主机通道 15 特性寄存器	0x0000_0000
HCSPLT15	0x6E4	读/写	主机通道 15 分裂控制寄存器	0x0000_0000
HCINT15	0x6E8	读/写	主机通道 15 中断寄存器	0x0000_0000
HCINTMASK15	0x6EC	读/写	主机通道 15 中断屏蔽寄存器	0x0000_0000
HCTSIZ15	0x6F0	读/写	主机通道 15 转换尺寸寄存器	0x0000_0000
HCDMA15	0x6F4	读/写	主机通道 15DMA 地址寄存器	0x0000_0000
设备模式寄存器				
设备全局寄存器				
DCFG	0x800	读/写	设备配置寄存器	0x0020_0000
DCTL	0x804	读/写	设备控制寄存器	0x0000_0000
DSTS	0x808	读	设备状态寄存器	0x0000_0002
DIEPMSK	0x810	读/写	设备 IN 端点通用中断屏蔽寄存器	0x0000_0000

DOEPMSK	0x814	读/写	设备输出端点通用中断屏蔽寄存器	0x0000_0000
DAIN	0x818	读	设备所有端点中断屏蔽寄存器	0x0000_0000
DAINTMAK	0x81C	读/写	设备所有端点中断屏蔽寄存器	0x0000_0000
DTKNQR1	0x420	读	设备输入象征连续学习队列读寄存器 1	0x0000_0000
DTKNQR2	0x424	读	设备输入象征连续学习队列读寄存器 2	0x0000_0000
DVBUSDIS	0x428	读/写	设备 VBUS 放电时间寄存器	0x0000_17D7
DVBUSPULSE	0x42C	读/写	设备 VBUS 脉冲时间寄存器	0x0000_05B8
DTKNQR3	0x430	读	设备输入象征连续学习队列读寄存器 3	0x0000_0000
DTKNQR4	0x434	读	设备输入象征连续学习队列读寄存器 4	0x0000_0000
设备逻辑输入特殊端点寄存器				
DIEPCTL0	0x900	读/写	设备控制输入端点 0 控制寄存器	0x0000_8000
DIEPINT0	0x908	读/写	设备输入端点 0 中断寄存器	0x0000_0000
DIEPTSIZE0	0x910	读/写	设备输入端点 0 转换尺寸寄存器	0x0000_0000
DIEPDMA0	0x914	读/写	设备输入端点 0 DMA 地址寄存器	0x0000_0000
DIEPCTL1	0x920	读/写	设备控制输入端点 1 控制寄存器	0x0000_0000
DIEPINT1	0x928	读/写	设备输入端点 1 中断寄存器	0x0000_0080
DIEPTSIZE1	0x930	读/写	设备输入端点 1 转换尺寸寄存器	0x0000_0000
DIEPDMA1	0x934	读/写	设备输入端点 1 DMA 地址寄存器	0x0000_0000
DIEPCTL3	0x940	读/写	设备控制输入端点 3 控制寄存器	0x0000_0000
DIEPINT3	0x948	读/写	设备输入端点 3 中断寄存器	0x0000_0080
DIEPTSIZE3	0x950	读/写	设备输入端点 3 转换尺寸寄存器	0x0000_0000
DIEPDMA3	0x954	读/写	设备输入端点 3 DMA 地址寄存器	0x0000_0000
DIEPCTL3	0x960	读/写	设备控制输入端点 3 控制寄存器	0x0000_0000
DIEPINT3	0x968	读/写	设备输入端点 3 中断寄存器	0x0000_0080

DIEPTSIZE3	0x970	读/写	设备输入端点 3 转换尺寸寄存器	0x0000_0000
DIEPDMA3	0x974	读/写	设备输入端点 3 DMA 地址寄存器	0x0000_0000
DIEPCTL4	0x980	读/写	设备控制输入端点 4 控制寄存器	0x0000_0000
DIEPINT4	0x988	读/写	设备输入端点 4 中断寄存器	0x0000_0080
DIEPTSIZE4	0x990	读/写	设备输入端点 4 转换尺寸寄存器	0x0000_0000
DIEPDMA4	0x994	读/写	设备输入端点 4 DMA 地址寄存器	0x0000_0000
DIEPCTL5	0x9A0	读/写	设备控制输入端点 5 控制寄存器	0x0000_0000
DIEPINT5	0x9A8	读/写	设备输入端点 5 中断寄存器	0x0000_0080
DIEPTSIZE5	0x9B0	读/写	设备输入端点 5 转换尺寸寄存器	0x0000_0000
DIEPDMA5	0x9B4	读/写	设备输入端点 5DMA 地址寄存器	0x0000_0000
DIEPCTL6	0x9C0	读/写	设备控制输入端点 6 控制寄存器	0x0000_0000
DIEPINT6	0x9C8	读/写	设备输入端点 6 中断寄存器	0x0000_0080
DIEPTSIZE6	0x9D0	读/写	设备输入端点 6 转换尺寸寄存器	0x0000_0000
DIEPDMA6	0x9D4	读/写	设备输入端点 6DMA 地址寄存器	0x0000_0000
DIEPCTL7	0x9E0	读/写	设备控制输入端点 7 控制寄存器	0x0000_0000
DIEPINT7	0x9E8	读/写	设备输入端点 7 中断寄存器	0x0000_0080
DIEPTSIZE7	0x9F0	读/写	设备输入端点 7 转换尺寸寄存器	0x0000_0000
DIEPDMA7	0x9F4	读/写	设备输入端点 7DMA 地址寄存器	0x0000_0000
DIEPCTL8	0xA00	读/写	设备控制输入端点 8 控制寄存器	0x0000_0000
DIEPINT8	0xA08	读/写	设备输入端点 8 中断寄存器	0x0000_0080
DIEPTSIZE8	0xA10	读/写	设备输入端点 8 转换尺寸寄存器	0x0000_0000
DIEPDMA8	0xA14	读/写	设备输入端点 8 DMA 地址寄存器	0x0000_0000
DIEPCTL9	0xA20	读/写	设备控制输入端点 9 控制寄存器	0x0000_0000
DIEPINT9	0xA28	读/写	设备输入端点 9 中断寄存器	0x0000_0080
DIEPTSIZE9	0xA30	读/写	设备输入端点 9 转换尺寸寄存器	0x0000_0000
DIEPDMA9	0xA34	读/写	设备输入端点 9 DMA 地址寄存器	0x0000_0000
DIEPCTL10	0xA40	读/写	设备控制输入端点 10 控制寄存器	0x0000_0000
DIEPINT10	0xA48	读/写	设备输入端点 10 中断寄存器	0x0000_0080



DIEPTSIZE10	0xA50	读/写	设备输入端点 10 转换尺寸寄存器	0x0000_0000
DIEPDMA10	0xA54	读/写	设备输入端点 10 DMA 地址寄存器	0x0000_0000
DIEPCTL11	0xA60	读/写	设备控制输入端点 11 控制寄存器	0x0000_0000
DIEPINT11	0xA68	读/写	设备输入端点 11 中断寄存器	0x0000_0080
DIEPTSIZE11	0xA70	读/写	设备输入端点 11 转换尺寸寄存器	0x0000_0000
DIEPDMA11	0xA74	读/写	设备输入端点 11 DMA 地址寄存器	0x0000_0000
DIEPCTL12	0xA80	读/写	设备控制输入端点 12 控制寄存器	0x0000_0000
DIEPINT12	0xA88	读/写	设备输入端点 12 中断寄存器	0x0000_0080
DIEPTSIZE12	0xA90	读/写	设备输入端点 12 转换尺寸寄存器	0x0000_0000
DIEPDMA12	0xA94	读/写	设备输入端点 12 DMA 地址寄存器	0x0000_0000
DIEPCTL13	0xAA0	读/写	设备控制输入端点 13 控制寄存器	0x0000_0000
DIEPINT13	0xAA8	读/写	设备输入端点 13 中断寄存器	0x0000_0080
DIEPTSIZE13	0xAB0	读/写	设备输入端点 13 转换尺寸寄存器	0x0000_0000
DIEPDMA13	0xAB4	读/写	设备输入端点 13 DMA 地址寄存器	0x0000_0000
DIEPCTL14	0xAD0	读/写	设备控制输入端点 14 控制寄存器	0x0000_0000
DIEPINT14	0xAC8	读/写	设备输入端点 14 中断寄存器	0x0000_0080
DIEPTSIZE14	0xAD0	读/写	设备输入端点 14 转换尺寸寄存器	0x0000_0000
DIEPDMA14	0xAD4	读/写	设备输入端点 14 DMA 地址寄存器	0x0000_0000
DIEPCTL15	0xAE0	读/写	设备控制输入端点 15 控制寄存器	0x0000_0000
DIEPINT15	0xAE8	读/写	设备输入端点 15 中断寄存器	0x0000_0080
DIEPTSIZE15	0xAF0	读/写	设备输入端点 15 转换尺寸寄存器	0x0000_0000
DIEPDMA15	0xAF4	读/写	设备输入端点 15 DMA 地址寄存器	0x0000_0000
设备逻辑输入特殊端点寄存器				
DOEPTCTL0	0xB00	读/写	设备控制输出端点 0 控制寄存器	0x0000_8000
DOEPINT0	0xB08	读/写	设备输出端点 0 中断寄存器	0x0000_0000
DOEPTSIZE0	0xB10	读/写	设备输出端点 0 转换尺寸寄存器	0x0000_0000
DOEPDMA0	0xB14	读/写	设备输出端点 0 DMA 地址寄存器	0x0000_0000
DOEPTCTL1	0xB20	读/写	设备控制输出端点 1 控制寄存器	0x0000_0000

DOEPINT1	0xB28	读/写	设备输出端点 1 中断寄存器	0x0000_0080
DOEPTSIZE1	0xB30	读/写	设备输出端点 1 转换尺寸寄存器	0x0000_0000
DOEPDMA1	0xB34	读/写	设备输出端点 1 DMA 地址寄存器	0x0000_0000
DOEPTCTL3	0xB40	读/写	设备控制输出端点 3 控制寄存器	0x0000_0000
DOEPINT3	0xB48	读/写	设备输出端点 3 中断寄存器	0x0000_0080
DOEPTSIZE3	0xB50	读/写	设备输出端点 3 转换尺寸寄存器	0x0000_0000
DOEPDMA3	0xB54	读/写	设备输出端点 3 DMA 地址寄存器	0x0000_0000
DOEPTCTL3	0xB60	读/写	设备控制输出端点 3 控制寄存器	0x0000_0000
DOEPINT3	0xB68	读/写	设备输出端点 3 中断寄存器	0x0000_0080
DOEPTSIZE3	0xB70	读/写	设备输出端点 3 转换尺寸寄存器	0x0000_0000
DOEPDMA3	0xB74	读/写	设备输出端点 3 DMA 地址寄存器	0x0000_0000
DOEPTCTL4	0xB80	读/写	设备控制输出端点 4 控制寄存器	0x0000_0000
DOEPINT4	0xB88	读/写	设备输出端点 4 中断寄存器	0x0000_0080
DOEPTSIZE4	0xB90	读/写	设备输出端点 4 转换尺寸寄存器	0x0000_0000
DOEPDMA4	0xB94	读/写	设备输出端点 4 DMA 地址寄存器	0x0000_0000
DOEPTCTL5	0xBA0	读/写	设备控制输出端点 5 控制寄存器	0x0000_0000
DOEPINT5	0xBA8	读/写	设备输出端点 5 中断寄存器	0x0000_0080
DOEPTSIZE5	0xBB0	读/写	设备输出端点 5 转换尺寸寄存器	0x0000_0000
DOEPDMA5	0xBB4	读/写	设备输出端点 5DMA 地址寄存器	0x0000_0000
DOEPTCTL6	0xBC0	读/写	设备控制输出端点 6 控制寄存器	0x0000_0000
DOEPINT6	0xBC8	读/写	设备输出端点 6 中断寄存器	0x0000_0080
DOEPTSIZE6	0xBD0	读/写	设备输出端点 6 转换尺寸寄存器	0x0000_0000
DOEPDMA6	0xBD4	读/写	设备输出端点 6DMA 地址寄存器	0x0000_0000
DOEPTCTL7	0xBE0	读/写	设备控制输出端点 7 控制寄存器	0x0000_0000
DOEPINT7	0xBE8	读/写	设备输出端点 7 中断寄存器	0x0000_0080
DOEPTSIZE7	0xBF0	读/写	设备输出端点 7 转换尺寸寄存器	0x0000_0000
DOEPDMA7	0xBF4	读/写	设备输出端点 7DMA 地址寄存器	0x0000_0000
DOEPTCTL8	0xC00	读/写	设备控制输出端点 8 控制寄存器	0x0000_0000

DOEPINT8	0xC08	读/写	设备输出端点 8 中断寄存器	0x0000_0080
DOEPTSIZE8	0xC10	读/写	设备输出端点 8 转换尺寸寄存器	0x0000_0000
DOEPDMA8	0xC14	读/写	设备输出端点 8 DMA 地址寄存器	0x0000_0000
DOEPCTL9	0xC20	读/写	设备控制输出端点 9 控制寄存器	0x0000_0000
DOEPINT9	0xC28	读/写	设备输出端点 9 中断寄存器	0x0000_0080
DOEPTSIZE9	0xC30	读/写	设备输出端点 9 转换尺寸寄存器	0x0000_0000
DOEPDMA9	0xC34	读/写	设备输出端点 9 DMA 地址寄存器	0x0000_0000
DOEPCTL10	0xC40	读/写	设备控制输出端点 10 控制寄存器	0x0000_0000
DOEPINT10	0xC48	读/写	设备输出端点 10 中断寄存器	0x0000_0080
DOEPTSIZE10	0xC50	读/写	设备输出端点 10 转换尺寸寄存器	0x0000_0000
DOEPDMA10	0xC54	读/写	设备输出端点 10 DMA 地址寄存器	0x0000_0000
DOEPCTL11	0xC60	读/写	设备控制输出端点 11 控制寄存器	0x0000_0000
DOEPINT11	0xC68	读/写	设备输出端点 11 中断寄存器	0x0000_0080
DOEPTSIZE11	0xC70	读/写	设备输出端点 11 转换尺寸寄存器	0x0000_0000
DOEPDMA11	0xC74	读/写	设备输出端点 11 DMA 地址寄存器	0x0000_0000
DOEPCTL12	0xC80	读/写	设备控制输出端点 12 控制寄存器	0x0000_0000
DOEPINT12	0xC88	读/写	设备输出端点 12 中断寄存器	0x0000_0080
DOEPTSIZE12	0xC90	读/写	设备输出端点 12 转换尺寸寄存器	0x0000_0000
DOEPDMA12	0xC94	读/写	设备输出端点 12 DMA 地址寄存器	0x0000_0000
DOEPCTL13	0xCA0	读/写	设备控制输出端点 13 控制寄存器	0x0000_0000
DOEPINT13	0xCA8	读/写	设备输出端点 13 中断寄存器	0x0000_0080
DOEPTSIZE13	0xCB0	读/写	设备输出端点 13 转换尺寸寄存器	0x0000_0000
DOEPDMA13	0xCB4	读/写	设备输出端点 13 DMA 地址寄存器	0x0000_0000
DOEPCTL14	0xCD0	读/写	设备控制输出端点 14 控制寄存器	0x0000_0000
DOEPINT14	0xCC8	读/写	设备输出端点 14 中断寄存器	0x0000_0080
DOEPTSIZE14	0xCD0	读/写	设备输出端点 14 转换尺寸寄存器	0x0000_0000
DOEPDMA14	0xCD4	读/写	设备输出端点 14 DMA 地址寄存器	0x0000_0000
DOEPCTL15	0xCE0	读/写	设备控制输出端点 15 控制寄存器	0x0000_0000

DOEPINT15	0xCE8	读/写	设备输出端点 15 中断寄存器	0x0000_0080
DOEPTSIZ15	0xCF0	读/写	设备输出端点 15 转换尺寸寄存器	0x0000_0000
DOEPDMA15	0xCF4	读/写	设备输出端点 15 DMA 地址寄存器	0x0000_0000
电源和时钟选通寄存器				
PCGCCTL	0Xc00	读/写	电源和时钟选通控制寄存器	0x0000_0000

## 26.4 OTG PHY 控制寄存器

### 26.4.1.OTH PHY 电源控制寄存器（OPHYPPWR）

寄存器	地址	读/写	描述	复位值
OPHYPPWR	0x7C10_0000	读/写	OTG PHY 电源控制寄存器	0x0000000F

OPHYPPWR	位	读/写	描述	初始状态
Reserved	[31:5]		保留	27'h0
Otg_disable	[4]		在 PHY2.0 内 OTG 模块电源下降 1'b0: OTG 模块电源上升 1'b1: OTG 模块电源下降 如果实际应用中不适用 OTG 功能, 可以设置此位输入高电平来保存电源。	1'b1
Analog_powerdown	[3]	读_写	在 PHY2.0 内相似模块电源下降 1'b0: 相似模块电源上升	1'b1

			1'b1: 相似模块电源下降	
Reserved	[2:1]		保留	2 b'
Force_suspend	[0]	读_写	电源保存申请暂停信号 1'b0: 禁止（常规操作） 1'b1: 使能	1'b1

### 26.4.2 OTH PHY 时钟控制寄存器（OPHYCLK）

寄存器	地址	读/写	描述	复位值
OPHYCLK	0x7C10_0004	读/写	OTG PHY 时钟控制寄存器	0x0000000F

OPHYCLK	位	读/写	描述	初始状态
Reserved	[31:7]		保留	25'h0
Serial_mode	[6]	读_写	UTMI/串行接口选择 当次寄存器被声明以后，USB 交通流通过串行接口 1'b0: 通过 UTMI 转换和接收 D+ 行和 D-行数据 1'b1: 通过 USB1.1 串行接口转换和接收 D+行和 D-行数据	1'b0
Xo_ext_clk_enb	[5]	读_写	XO 模块接口时钟选择 1'b0: 外部晶体. 1'b1: 外部时钟/振荡器	1'b0
Common_on_n	[4]	读_写	在暂停期间 XO,Bias, Bandgap 和 PLL 维持驱动 当 USB 2.0 OTG PHY 暂停时，此	1'b1

			位控制共用模块内子模块的电源下降信号。  1'b0: 48MHz 时钟在 clk48m_ohci 上一直有效，暂停模式除外  1'b1: 48MHz 时钟在 clk48m_ohci 上一直有效，暂停模式下也一样有效	
Reserved	[3]		保留	1'b0
Serial_mode	[2]	读_写	相似 ID 输入样本使能  1'b0: id_dig 禁止  1'b1: id_dig 使能	1'b0
clk_sel	[1:0]	读_写	PLL 接口时钟频率选择  2'b00: 48MHz  2'b01: 保留  2'b10: 12MHz  2'b11: 24MHz	2'b00

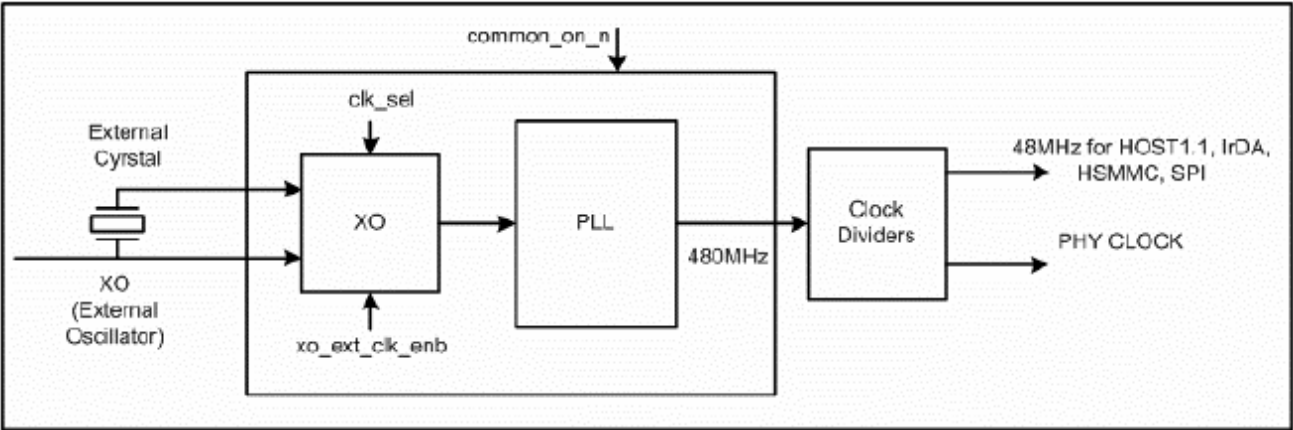


图 26-4 OTG PHY 时钟路径

26.4.3. OTG 复位控制寄存器（ORSTCON）

寄存器	地址	读/写	描述	复位值
OPHYCLK	0x7C10_0008	读/写	OTG 复位控制寄存器	0x00000001

OPHYCLK	位	读/写	描述	初始状态
Reserved	[31:3]		保留	29'h0
Phylnk_sw_rst	[2]	读_写	OTG 链接核心 phy_时钟范围 S/W 复位	1'b0
lnk_sw_rst	[1]	读_写	OTG 链接核心 hclk 范围 S/W 复位	1'b0
Phy_sw_rst	[0]	读_写	OT phy2.0 S/W 复位，Phy_sw_rst 信号必须声明为最少 10us	1'b1

26.4.4.PHY 调谐寄存器（PHY\_TUNE）

寄存器	地址	读/写	描述	复位值
OPHYTUNE	0x7C10_0020	读/写	OTG PHY 协调寄存器	0x00271B98

OPHYCLK	位	读/写	描述	初始状态
Reserved	[31:21]		保留	11'h1
Txpreemphasistune	[20]	读_写	高速传送器预强调使能。这个信号使在高速模式下 J-K 或 K-J 状态转换的预先强调能或不能  1：高速传送器可以预先强调 0：高速传送器不能预先强调	1'b0
Compdistune	[19:17]	读_写	断开阈值调整。这个总线调整阈值的电压值，用于发现主机内的断开	3'b011

			事件  111: +6%  110: +4.5%  101: +3%  100: +1.5%  011: 设计默认  010: -3%  001: -4%  000: -6%	
Otggtune	[16:14]	读_写	VBUS 有效阈值调整。这个总线调整 VBUS 有效阈值的电压值  111: +9%  110: +6%  101: +3%  100: 设计默认  011: -3%  010: -6%  001: -9%  000: -12%	3'b100
Sqrgtune	[13:11]	读_写	限制阈值调整。这个总线调整阈值的电压值,用于发现有效的高速数据。  111: -20%  110: -15%  101: -10%  100: -5%  011: 设计默认-3%  010: +5%  001: +10%	3'b011



			000: +15%	
Txfslstune	[10:7]	读_写	FS/LS 上拉电阻调整。这个总线根据额定功率, 额定电压和温度调整低速和全速上拉电阻值  1111: -2.5%  0111: 设计默认  0011: +2.5%  0001: +5%  0000: +7.5%	4'b0111
Txrisetune	[6]	读_写	高速传送器升/降时间调整。这个总线调整高速的升/降时间  1: -8%  0: 设计默认	2'b01
Txhsxutune	[5:4]	读_写	传送器高速转向调整。这个总线调整高速传送模式下 DP 和 DM 信号交错电压值  11: 转向电压增加 15mv  10: 转向电压增加 30mv  01: 默认设置  00: 保留	2'b01
Txvreftune	[3:0]	读_写	高速 DC 电压值调整。这个总线调整电压值, 调谐高速 DC 电平。  1111: -8.75%  1110: +7.5%  1101: +6.25%  1100: +5%  1011: +3.75%  1010: +2.5%  1001: +1.25%	4'b0011

			1000:  0111: -1.25%  0110: -2.5%  0101: -3.75%  0100: -5%  0011: 设计默认  0010: -7.5%  0001: -8.75%  0000:	
--	--	--	---	--

26.5 OTG 链接核心寄存器

26.5.1 OTG 全局寄存器

在主机模式和设备模式下 OTG 全局寄存器都是可用的，而且当这些 OTG 全局寄存器之间转换时不需要重新程式化。

26.5.1.1. OTG 控制和状态寄存器（GOTGCTL）

OTG 控制和状态寄存器控制 OTG 核心功能的行为，并且影响 OTG 核心功能的状态。

寄存器	地址	读/写	描述	复位值
GOTGCTL	0x7C00_0000	读/写	OTG 控制和状态寄存器	0x00010000

GOTGCTL	位	读/写	描述	初始状态
Reserved	[31:20]		保留	12'h0
BSesVId	[19]	只读	B-Session 有效，指明设备模式收发机状态  1'b0: B-Session 无效	1'b0

			1'b1: B-Session 有效	
ASesVId	[18]	只读	A-Session 有效, 指明主机模式收发机状态  1'b0: A-Session 无效 1'b1: A-Session 有效	1'b0
DbncTime	[17]	只读	长/短去除抖动时间, 只能发现连接的抖动时间  1'b0: 长抖动时间, 用于物理连接 1'b1: 短抖动时间, 用于软件连接	1'b0
ConIDSts	[16]	只读	连接器 ID 状态, 指明连接器 ID 状态  1'b0: OTG 核心在 A 设备模式下 1'b1: OTG 核心在 B 设备模式下	1'b1
Reserved	[15:12]		保留	4'h0
DevHNPEn	[11]	读_写	设备 HNP 使能, 实际应用中当此位成功接收一个 SetFeature 时, 设置此位  1'b0: 实际应用中 HNP 禁止 1'b1: 实际应用中 HNP 使能	1'b0
HNPREq	[10]	读_写	HNP 请求, 实际应用中设置此位向连接的 USB 主机开始一个 HNP 请求。当 HstNegSucStsChng 位被清除时, 核心清除此位。  1'b0: 没有 HNP 请求 1'b1: 发出 HNP 请求	1'b0
HstNegScs	[8]	只读	主机协商成功  当主机协商成功时, 核心设置此位, 当在此寄存器内设置 HNP 请求位是, 核心清除此位。	1'b0

			1'b0: 主机协商失败 1'b1: 主机协商成功	
Reserved	[7:2]		保留	6'h0
SesReq	[1]	读_写	Session 请求 实际应用之设置此位开始一个 USB session 请求。当 HstNegSucStsChng 位清除是，核 心清除此位。 1'b0: 主机协商失败 1'b1: 主机协商成功	1'b0
SesReqScs	[0]	只读	Session 请求成功 当 Session 请求开始成功时核心设 置此位 1'b0: Session 请求失败 1'b1: Session 请求成功	1'b0

### 26.5.1.2. OTH 中断寄存器（GOTGINT）

在实际应用中，无论何时读取这个寄存器，都会有一个 OTG 中断，并且可以通过清除寄存器内的此位置进行清除 OTG 中断。

寄存器	地址	读/写	描述	复位值
GOTGINT	0x7C00_0004	读/写	OTG 中断寄存器	0x00000000

GOTGINT	位	读/写	描述	初始状态
Reserved	[31:20]		保留	12'h0
DbnceDone	[19]	R-SS-WC	抖动操作 当设备连接后完成抖动时，核心	1'b0

			设置此位。只有当核心 USB 配置寄存器内的 HNP 能力和 SRP 能力位设置时此位才有效。	
ADevROUTChg	[18]	R-SS-WC	A 设备超时变化 核心设置此位是为了指明 A 设备等待向 B 设备连接时的超时现象。	1'b0
HstNegDet	[17]	R-SS-WC	主机谈判检测 当检测 USB 的主机谈判请求时，核心设置此位。	1'b0
Reserved	[16:10]		保留	7'h0
HstNegSucStsChng	[9]	R-SS-WC	主机谈判成功状态变化 核心在此位设置 USB 主机谈判请求的成功或失败。	1'b0
SesReqSucStsChng	[8]	R-SS-WC	Session 请求成功状态变化 核心在此位设置 Session 请求的成功或失败。	1'b0
Reserved	[7:3]		保留	5'h0
SesEndDet	[2]	R-SS-WC	Session 结束检测	1'b0
Reserved	[1:0]		保留	2'h0

### 26.5.1.3.OTG AHB 配置寄存器（GAHBCFG）

这个寄存器可以用来配置操作模式下电源打开或改变之后的核心。OTG AHB 配置寄存器主要包含 AHB 系统相关的配置参数。在开始程式化后不要改变这个寄存器。实际应用中，必须在 AHB 或 USB 开始任何转换之前对此寄存器编程。

寄存器	地址	读/写	描述	复位值
-----	----	-----	----	-----

GAHBCFG	0x7C00_0008	读/写	核心 AHB 配置寄存器	0x00000000
---------	-------------	-----	--------------	------------

GAHBCFG	位	读/写	描述	初始状态
Reserved	[31:9]		保留	23'h0
PTxFEmpLvl	[8]	读-写	<p>周期性 TxFIFO 空级别</p> <p>指明当周期性 TxFIFO 空时，核心中断寄存器内的中断位触发。此位只用于 Slave 模式。</p> <p>1'b0: GINTSTS.PtxFEmp 中断指明周期性 TxFIFO 是半空</p> <p>1'b0: GINTSTS.PtxFEmp 中断指明周期性 TxFIFO 全空</p>	1'b0
NPTxFEmpLvl	[7]	读-写	<p>非周期性 TxFIFO 空级别</p> <p>指明当非周期性 TxFIFO 空时，核心中断寄存器内的中断位触发。此位只用于 Slave 模式。</p> <p>1'b0: GINTSTS.PtxFEmp 中断指明非周期性 TxFIFO 是半空</p> <p>1'b0: GINTSTS.PtxFEmp 中断指明非周期性 TxFIFO 全空</p>	1'b0
Reserved	[6]		保留	1'b0
DMAEn	[5]	读-写	<p>DMA 使能</p> <p>1'b0: 核心运行在 Slave 模式下</p> <p>1'b0: 核心运行在 DMA 模式下</p>	1'b0
HBstLen	[4:1]	读-写	<p>突发长度/类型</p> <p>内部 DMA 模式-AHB 主突发类型：</p> <p>4'b0000: single</p> <p>4'b0001: INCR</p> <p>4'b0011: INCR4</p> <p>4'b0101: INCR8</p> <p>4'b0111: INCR16</p>	4'b0

			Others: 保留	
GlbIntrMsk	[0]	读-写	全局中断屏蔽 实际应用中，用此位屏蔽或不屏蔽中断行声明 1'b0: 屏蔽中断声明 1'b0: 不屏蔽中声明	1'b0

#### 26.5.1.4. OTG USB 配置寄存器（GUSBCFG）

OTG USB 配给寄存器用来配置主机模式或设备模式下电源打开或改变之后的核心。OTG USB 配置寄存器主要包含 USB 和 USB-PHY 相关的配置参数。实际应用中，必须在 AHB 或 USB 开始任何转换之前对此寄存器编程。在开始程式化后不要改变这个寄存器。

寄存器	地址	读/写	描述	复位值
GUSBCFG	0x7C00_000C	读/写	核心 USB 配置寄存器	0x00000000

GUSBCFG	位	读/写	描述	初始状态
Reserved	[31:16]		保留	16'h0
PHY Low-Power Clock Select	[15]	读-写	PHY 低电源时钟选择 选择 480MHz 或 48MHzPHY 模式。在 FS 和 LS 模式下，PHY 通常运行在 48HHz 模式下，进行保存电源 1'b0: 480MHz 内部 PLL 时钟 1'b0: 48MHz 外部时钟	1'b0
Reserved	[14: 10]		保留	5'h5
HNPCap	[9]	读-写	HNP 能力 实际应用中,用此位控制OTG 核心的 HNP 能力 1'b0: 没有 HNP 能力 1'b0: HNP 能力有效	1'b0

SRPCap	[8]	读-写	SRP 能力 实际应用中,用此位控制 OTG 核心的 SRP 能力 1'b0: 没有 SRP 能力 1'b0: SRP 能力有效	1'b0
Reserved	[7:4]		保留	4'h0
PHYIf	[3]	读-写	PHY 接口 实际应用中,用此位配置核心用 8 位或 16 位接口支持 UTMI+PHY.如果只支持 16 位接口, 需要将此位设置为 1. 1'b0: 8 位 1'b0: 16	1'b0
ToutCal	[2: 0]	读-写	HS/FS 超时校准 设置此位为 3'h7	3'h0

### 26.5.1.5. 核心复位寄存器（GRSTCTL）

实际应用中，用此位进行核心内部不同银奖的复位。

寄存器	地址	读/写	描述	复位值
GRSTCTL	0x7C00_0010	读/写	核心复位寄存器	0x80000000

GRSTCTL	位	读/写	描述	初始状态
AHBIdle	[31]	只读	闲置的 AHB Master 指明 AHB master 状态机器处于闲置环境	1'b1
DMAReq	[30]	只读	DMA 请求信号 指明 DMA 请求信号在进程中。用于调试	1'b0
Reserved	[38:11]		保留	19'h0
TxFNum	[10:6]	读-写	TxFIFO 序号	5'h0



			<p>FIFO 序号必须用 TxFIFO flush 位进行清除。这个区域直到核心清除 TxFIFO flush 位的时候才可以改变。</p> <p>5'h0: 非周期 TxFIFO flush</p> <p>5'h1: 设备模式内周期 TxFIFO 1 flush</p> <p>5'h2: 设备模式内周期 TxFIFO 2 flush</p> <p>.....</p> <p>5'hF: 设备模式内周期 TxFIFO15 flush</p> <p>5'h10: 清除核心内的所有周期和非周期的 TxFIFO</p>	
TxFFlsh	[5]	R_WS_SC	<p>TxFIFO Flush</p> <p>此位选择性的清除一个或者所有传送 FIFO,但是如果核心位于转换过程的中间时, 不可以进行此操作。在实际应用中, 只有在检查核心即不向向 TxFIFO 写入数据也不从 TxFIFO 读取数据之后才可以写入此位。实际应用当中必须在运行任何操作之前, 核心清除此位之后才可操作。此位用 8 个时钟进行清除</p>	1'b0
ExFFlsh	[4]	R_WS_SC	<p>RxFIFO Flush</p> <p>实际应用中, 用此位可以清楚整个 RxFIFO, 但是必须首先确保核心不再转换过程中的重点位置。在实际应用中, 只有在检查核心即不向向 TxFIFO 写入数据也不从 TxFIFO 读取数据之后才可以写入此位。实际应用当中必须在运行任何操作之前, 核心清除此位之后才可操作。此位用 8 个时钟进行清除</p>	1'b0
INTknQFlsh	[3]	R_WS_SC	<p>输入象征连续学习序列 Flush</p> <p>实际应用中, 写入此位来清除输入象征连续学习序列。</p>	1'b0
FrmCntrRst	[2]	R_WS_SC	<p>主机帧计数器复位</p> <p>实际应用中写入此位进行核心内部帧数量计数器</p>	1'b0

			的复位。当帧计数器复位后，随后通过核心 faculty 的 SOF 将有一个序号为 0 的帧。	
HSftRst	[1]	R_WS_SC	<p>HClk 软件复位</p> <p>实际应用中，用此位清除 AHB 时钟范围的控制逻辑。自会有 AHB 时钟范围管道可以复位。</p> <p>(1) FIFOs 不可以用此位清除</p> <p>(2) AHB 时钟范围内的所有机器在使 AHB 转换中断以后复位到 IDLE 区域。</p> <p>(3) AHB 时钟范围机器用的 DSR 内的控制位被清零。</p> <p>(4) 通过 AHB 时钟范围机器产生的状态屏蔽位控制中断状态，被清零以后清除中断</p> <p>(5) 因为中断状态为未被清零，实际应用中在设置此位以后可以得到任何核心事件的状态，此位为自动清零位，在核心内所有必要的逻辑复位以后，核心自动清零。需要耗费几个时钟，耗费时钟的数量根据核心当前状态决定。</p>	1'b0
CSftRst	[1]	R_WS_SC	<p>核心软件复位</p> <p>根据下面所述复位 HCLK 和 PHY 时钟</p> <p>(1) 清除所有中断和所有 CSR 寄存器，下面所述寄存器位粗外</p> <p>1) HCFG.FSLSPclkSel</p> <p>2) DCFG.DevSpd</p> <p>(2) 所有模版区机器（除了 AHB Slave 单元）都被复位到 IDLE 区，并且所有的转移 FIFOS 和接收 FIFO 被清除。</p> <p>(3) AHB Master 上的任何转换操作被尽可能快的中断，在 AHB 转换器完成最后的数据阶段后。USB 上的任何转换操作被立刻中断。</p>	1'b0

### 26.5.1.6.核心中断寄存器（GINTSTS）

这个寄存器中断应用与在操作当前模式下的系统等级事件。

寄存器	地址	读/写	描述	复位值
GINTSTS	0x7C00_0014	读/写	核心中断寄存器	0x04001020

GINTSTS	位	读/写	描述	初始状态
WkUpInt	[31]	R_SS_WC	设备模式内的恢复/远程唤醒检测中断，当在 USB 内检测恢复时声明此中断。主机模式下，当在 USB 上检测远程唤醒时，声明此中断。	1'b0
SessReqInt	[30]	R_SS_WC	主机模式内的 Session 请求/新的 Session 检测中断，当从设备每检测 Session 请求时，声明此中断。设备模式下，当 b_valid 信号变成高电平时声明此中断。	1'b0
DisconnInt	[29]	R_SS_WC	断开检测中断 当检测到设备断开时，声明此中断	1'b0
ConIDStsChng	[28]	R_SS_WC	连接器 ID 状态变化 当连接器 ID 状态发生变化时，核心设置此位	1'b0
Reserved	[27]		保留	1'b0
PtxFEmp	[26]	RO	空闲的周期 Tx FIFO 当周期传输 FIFO 为半空或全空，有接收最少一个向周期请求队列写入信号的空间时，声明此位。半空或全空状态由核心 AHB 配置寄存器内的周期 Tx FIFO 空闲等级位决定。	1'b1
HchInt	[25]	RO	主机通道中断 核心设置此位来指明一个中断悬挂在核心（主机模式）的一个通道上。实际应用中，必须读取主	1'b0

			机所有通道中断寄存器，以便决定发生中断的通道 的正确序号，然后读取相应的序号的主机通道 总段寄存器，决定正确的中断源。实际应用中必 须在 <b>HCINTn</b> 寄存器内清除适当的状态位以便 清除此位。	
<b>PrtInt</b>	[24]	<b>RO</b>	主机端口中断  核心设置此位用于指明主机模式下 <b>OTG</b> 和弦端 口的端口状态的变化。实际应用中必须读取主机 端口控制和章台寄存器，以便聚顶产生此中断的 正确事件。在应用中，必须清楚主机短空控制和 状态寄存器内适当的状态为来清除此位。	1'b0
<b>Reserved</b>	[23]		保留	1'b0
<b>FetSusp</b>	[22]	<b>R_SS_WC</b>	数据提取暂停。只有在 <b>DMA</b> 模式下，此中断有 效。中断指明因为 <b>TxFIFO</b> 空间或请求队列空间 的不可用核心已经停止向输入端点提取数据。端 点不匹配算法应用 中会用到此中断。  例如，检测端点不匹配以后，将会有如下应用：  (1) 设置一个全局非周期的输入 <b>NAK</b> 交换  (2) 在端点处无效  (3) 刷新 <b>FIFO</b>  (4) 从令牌序列学习队列内确定令牌序列  (5) 重新使能端点  (6) 清除全局非周期输入 <b>NAK</b> 交换  如果全局非周期输入 <b>NAK</b> 交换被清除，核心并 没有从输入端点取得数据，并且输入令牌接收到 核心产生的：“ <b>IN token recervedwhen FIFO</b> <b>empty</b> ”中断。然后 <b>OTG</b> 向主机发送一个 <b>NAK</b> 回应。为了避免推测，应用时可以检测 <b>GINSTS</b> 。 <b>FetSusp</b> 总段确保在清除全局 <b>MAK</b> 交换之前	1'b0

			FIFO 是满的。	
IncomplIP	[21]	R_SS_WC	不完整的周期转换。在主机模式下，当有为了当前的微帧计划仍然未定的不完整用于周期转换时，核心设置此位。	1'b0
IncomplSOOUT			不完全同步的转换。设备模式下，核心设置此中断用于指明在当前微帧内最少有一个同步进程输出转换端点没有完全转换。这个中断伴随着此寄存器内的周期帧中断位的末尾被声明。	
IncomplSOIN	[20]	R_SS_WC	不完全同步输入转换。核心设置此中断用于指明在当前微帧内最少有一个同步进程输入转换端点没有完全转换。这个中断伴随着此寄存器内的周期帧中断位的末尾被声明。	1'b0
OEPInt	[19]	RO	输出端点中断。核心设置此位用于指明有一个终端悬挂在核心的一个输出端点上。实际应用中，需要读取设备的所有端点中断寄存器，以便决定发生中断的输出端点的正确序号。然后，读取相应设备输出端点序号的终端（DOEPINTn）寄存器决定中断的正确原因。应用纵，需要清除相应DOEPINTn 寄存器的适当状态位来清除此位。	1'b0
IEPInt	[18]	RO	输出端点中断。核心设置此位用于指明有一个终端悬挂在核心的一个输出端点上。实际应用中，需要读取设备的所有端点中断寄存器，以便决定发生中断的输入端点的正确序号。然后，读取相应设备输入端点序号的终端（DOEPINTn）寄存器决定中断的正确原因。应用纵，需要清除相应DOEPINTn 寄存器的适当状态位来清除此位。	1'b0
EPMis	[17]	R_SS_WC	端点不匹配中断 指明 IN 令牌应该被接受，用于非周期的断点，但是另一个端点的数据被保存在非周期传送的	1'b0

			FIFO 中，并且应用中输入端点不匹配计数进程已经终止。	
Reserved	[16]		保留	1'b0
EOPF	[15]	R_SS_WC	周期帧中断的末尾 指明了在当前微帧内，已经到达了设备配置寄存器内的周期帧间隔时间领域的特殊期间。	1'b0
ISOutDrop	[14]	R_SS_WC	同步输出包发行中断 当向 R 向 FIFO 写入同步输出包失败时，核心设置此位。写入失败的原因是因为 RxFIFO 没有足够的空间容纳最大包尺寸同步输出端点包。	1'b0
EnumDone	[13]	R_SS_WC	列举操作 核心设置此位用于指明高速列举已经完成。应用中，需要读取设备状态寄存器已达到列举速度。	1'b0
USBRst	[12]	R_SS_WC	USB 复位 核心设置此位用于指明在 USB 中已经检测到一个复位操作。	1'b1
USBSusp	[11]	R_SS_WC	USB 暂停 核心设置此位用于指明在 USB 上检测到一个暂停操作。当很长时间内线状态信号斗没被激活时，核心进入暂停状态。	1'b0
ErlySusp	[10]	R_SS_WC	早期暂停 核心设置此位用于指明在 USB 内发现一个 3ms 的限制状态。	1'b0
Reserved	[9]		保留	1'b0
Reserved	[9]		保留	1'b0
GOUTNakEFF	[7]	RO	有效的全局输出 NAK 指明设备控制寄存器内的设置全局输出 NAK 位（DETL.SGOUTNak），通过应用设置，在核心内有影响。通过在设备控制寄存器内向清除全局	1'b0

			输出 NAK 位写入数据可以清除此位。	
GINNakEff	[6]	RO	<p>有效的全局输入非周期 NAK。</p> <p>指明设备控制寄存器内的设置全局非周期输入 NAK 位。，通过应用设置，在核心内有影响，意思是核心有相似测全局输入 NAK 位通过应用可以设置。可以通过应用向清除清除全局非周期输入 NAK 位进行此位的清除。</p>	1'b0
NPTxFEmp	[5]	RO	<p>空闲的非周期 TxFIO</p> <p>当非周期 FIFO 为半空或全空，且有接收最少一个向非周期传送请求队列写入信号的空间时，声明此位。半空或全空状态由核心 AHB 配置寄存器内的非周期 TxFIFO 空闲等级位决定。</p>	1'b1
RxFLvl	[4]	R_SS_WC	<p>RxFIFO 非空闲</p> <p>指明至少有一个等待从 RxFIFO 读取数据的包。</p>	1'b0
Sof	[3]	R_SS_WC	<p>帧的开始</p> <p>主机模式，核心设置此位用于指明在 USB 上已经传输完一个 SOF,micro-SOF，或 Keep-Alive。</p> <p>应用中，需要向此位写入 1 清除中断。在设备模式下，核心设置此位用于指明 USB 上已经接受到一个 SOF 令牌。应用中，可以读取设备状态寄存器获取当前帧序号。只有当核心运行在 HS 或 FS 时才可以看到此中断。</p>	1'b0
OTGInt	[2]	RO	<p>OTG 中断</p> <p>核心设置此位用于指明一个 OTG 协议事件。应用中，需要读取 OTG 中断状态（GOTGINT）寄存器决定导致中断的正确事件。应用中需要清除 GOTGINT 寄存器内的相应的状态位来清除此位。</p>	1'b0
ModeMis	[1]	R_SS_WC	模式不协调中断	1'b0

			当应用中视图访问下面所述寄存器时，核心设置此位： （1）访问 主机模式寄存器，当核心在设备模式下运行时 （2）访问设备模式寄存器，当核心在主机模式下运行时	
CurMod	[0]	RO	操作的当前模式 指名操作当前模式 1'b0: 设备模式 1'b1: 主机模式	1'b0

### 26.5.1.7.核心中断屏蔽寄存器（GINTMSK）

寄存器使用核心中断寄存器中断应用。当一个中断位被屏蔽时，与中断相联系的位不会产生中断。然而，与中断相对应的核心中断（GINTSTS）寄存器位将被设置。

屏蔽中断：1'b0

不屏蔽中断：1'b1

寄存器	地址	读/写	描述	复位值
GINTMSK	0x7C00_0018	读/写	核心中断寄存器	0x04001020

GINTMSK	位	读/写	描述	初始状态
WkUpIntMsk	[31]	R_W	恢复/远程唤醒检测中断屏蔽	1'b0
SessReqIntMsk	[30]	R_W	Session 请求/新的 Session 检测中断屏蔽	1'b0
DisconnIntMsk	[29]	R_W	断开检测中断屏蔽	1'b0
ConIDStsChngMsk	[28]	R_W	连接器 ID 状态变化屏蔽	1'b0
Reserved	[27]		保留	1'b0
PtxFEmpMsk	[26]	R_W	空闲的周期 Tx FIFO 屏蔽	1'b1
HchIntMsk	[25]	R_W	主机通道中断屏蔽	1'b0



PrtIntMsk	[24]	R_W	主机端口中断屏蔽	1'b0
Reserved	[23]		保留	1'b0
FetSuspMsk	[22]	R_W	数据提取暂停屏蔽	1'b0
IncompIPMsk	[21]	R_W	不完整的周期转换屏蔽	1'b0
IncomplSOOUTMsk			不完全同步的转换屏蔽	
IncompSOINMsk	[20]	R_W	不完全同步输入转换屏蔽	1'b0
OEPIntMsk	[19]	R_W	输出端点中断屏蔽	1'b0
IEPIntMsk	[18]	R_W	输出端点中断屏蔽	1'b0
EPMisMsk	[17]	R_W	端点不匹配中断屏蔽	1'b0
Reserved	[16]	R_W	保留	1'b0
EOPFMsk	[15]	R_W	周期帧中断的末尾屏蔽	1'b0
ISOutDropMsk	[14]	R_W	同步输出包发行中断屏蔽	1'b0
EnumDoneMsk	[13]]	R_W	列举操作屏蔽	1'b0
USBRstMsk	[12]	R_W	USB 复位屏蔽	1'b1
USBSuspMsk	[11]	R_W	USB 暂停屏蔽	1'b0
ErlySuspMsk	[10]	R_W	早期暂停屏蔽	1'b0
Reserved	[9]		保留	1'b0
Reserved	[9]		保留	1'b0
GOUTNakEFFMsk	[7]	R_W	有效的全局输出 NAK 屏蔽	1'b0
GINNakEffMsk	[6]	R_W	有效的全局输入非周期 NAK 屏蔽	1'b0
NPTxFEmpMsk	[5]	R_W	空闲的非周期 TxFIO 屏蔽	1'b1
RxFLvlMsk	[4]	R_W	RxFIFO 非空闲屏蔽	1'b0
SofMsk	[3]	R_W	帧屏蔽的开始	1'b0
OTGIntMsk	[2]	R_W	OTG 中断屏蔽	1'b0
ModeMisMsk	[1]	R_W	模式不协调中断屏蔽	1'b0
Reserved	[0]		保留	1'b0

26.5.1.8.接收状态调试读取/状态读取和 Pop 寄存器（GRXSTSR/GRXSTSP）

接收状态调试读取寄存器内的读操作返回接收的 FIFO 顶层的内容。接收状态读取和 Pop 寄存器的读操作，增添了进入 RxFIFO 的 pops 数据。

接收状态内容在主机模式和设备模式内有不同的解释。当接收 FIFO 为空时，核心忽略接收状态 POP/读取，并且返回 32'h0000\_0000 值。应用中，当核心中断寄存器的接收 FIFO 非空位被声明时，只有 pop 接收状态 FIFO。

1.主机模式接收状态调试读取/状态读取和 Pop 寄存器（GRXSTSR/GRXSTSP）

寄存器	地址	读/写	描述	复位值
GRXSTSR/GRXSTSP	0x7C00_001C 0x7C00_0020	读	主机模式接收状态调试读取/状态读区和 Pop 寄存器	-

GRXSTSR/GRXSTSP	位	读/写	描述	初始状态
Reserved	[31:21]		保留	-
PktSts	[20:17]	RO	包状态 指明接收的包状态 4'b0010: 接收的输入数据包 4'b0011: 输入转换完成（触发一个中断） 4'b0101: 数据切换错误（触发一个终端） 4'b0111: 通道停止 Others: 保留	-
DPID	[16:15]	RO	数据 ID 指明接收包的数据 ID 2'b00: DATA0 2'b10: DATA1 2'b01: DATA2 2'b11: M DATA	-
BCnt	[14:4]	RO	字节数	-

			指明接收的输入数据包的字节数	
ChNum	[3:0]	RO	通道序号 指明属于当前接收包的通道序号	-

## 2.设备模式接收状态调试读取/状态读区和 Pop 寄存器（GRXSTSR/GRXSTSP）

寄存器	地址	读/写	描述	复位值
GRXSTSR/GRXSTSP	0x7C00_001C 0x7C00_0020	读	设备模式接收状态调试读取/状态读区和 Pop 寄存器	0xFFFFFFFF

GRXSTSR/GRXSTSP	位	读/写	描述	初始状态
Reserved	[31:25]		保留	7'h3F
FN	[24:21]	RO	帧序号 这是 USB 接收到得包内帧序号中最重要的 4 位。只有在同步输出端口支持的情况下才支持此区域。	4'hF
PktSts	[20:17]	RO	包状态 指明接收的包状态 4'b0001: 全局输出 NAK(触发一个中断) 4'b0010: 接收包的输出数据 4'b0011: 输出转换完成（触发一个中断） 4'b0100: 设置转换完成（触发一个中断） 4'b0110: 接收的设置数据包 Others: 保留	4'b1111
DPID	[16:15]	RO	数据 ID 指明接收的输出数据包的数据 ID 2'b00: DATA0 2'b10: DATA1 2'b01: DATA2 2'b11: M DATA	2'b11

BCnt	[14:4]	RO	字节数 指明接收的数据包的字节数	11'n3FF
EPNum	[3:0]	RO	端点序号 指明属于当前接收包的端点序号	4'hF

### 26.5.1.9. 接收 FIFO 尺寸寄存器（GRXFSIZ）

应用中可以对 RAM 尺寸进行编程，必须分配到 RxFIFO。

寄存器	地址	读/写	描述	复位值
GRXFSIZ	0x7C00_0024	读/写	接收 FIFO 尺寸寄存器	0x00001800

GRXFSIZ	位	读/写	描述	初始状态
Reserved	[31:16]		保留	16'h0
RxFDep	[15:0]	R_ W	RxFIFO 深度 这个值是 32 位的。 最小值是 16 最大值是 6144 这个寄存器的电源打开复位值是特殊的最大的 Rx 数据 FIFO 深度。 必须向此位写入一个新的值。程式化的值不能超过电源打开值设置。	16'h1800

### 26.5.1.10.非周期传送 FIFO 尺寸寄存器（GNPTXFSIZ）

实际应用中，可以变出 RAM 尺寸和存储器的非周期 TxFIFO 开始地址。

寄存器	地址	读/写	描述	复位值
GNPRXFSIZ	0x7C00_0028	读/写	非周期传送尺寸 FIFO 寄存器	0x18001800

GRXFSIZ	位	读/写	描述	初始状态
NPTxFDep	[31:16]	R_ W	非周期 Tx FIFO 深度  这个值是 32 位的。  最小值是 16  最大值是 32768  这个寄存器的电源打开复位值是特殊的最大的非周期的 Rx 数据 FIFO 深度。  必须向此位写入一个新的值。可程式化的值不能超过电源打开值设置。	16'h1800
NPTxFStADDR	[15:0]	R_ W	非周期转换开始地址  这个区域包含非周期传送 FIFO RAM 的存储器开始地址。  寄存器电源复位值是特殊的最大的 Rx 数据 FIFO 深度（6144）  必须向此位写入一个新的值。可程式化的值不能超过电源打开值设置。	16'h1800

#### 26.5.1.11. 非周期传送 FIFO/Queue 状态寄存器（GNPTXSTS）

这个只读寄存器包括了非周期 Tx FIFO 和非周期传送请求队列的自由空间信息。

寄存器	地址	读/写	描述	复位值
GNPTXSTS	0x7C00_002C	读/写	非周期传送 FIFO/Queue 状态寄存器	0x00081800

GRXFSIZ	位	读/写	描述	初始状态
---------	---	-----	----	------

Reserved	[31]		保留	1'b0
NPTxQTop	[30:24]	RO	<p>非周期 Tx 请求队列中的非周期传送请求队列空闲顶层是 MAC 当前运行的内容。</p> <p>位[30:27]:通道/端点序号</p> <p>位[26:25]:</p> <p>2'b00: 输入/输出令牌</p> <p>2'b01: 零长度传送包 (设备输入/主机输出)</p> <p>2'b10: PING/CSPLIT 令牌</p> <p>2'b11: 通道停止命令</p> <p>位[24]:结束 (最后进入选择的通道/端点)</p>	7'h0
NPTxQSpAvail	[23:16]	RO	<p>可用的非周期 Tx 请求队列空间</p> <p>指明非周期传送请求队列中可用自由空间的数量。这个队列停止主机模式的输入和输出请求。设备模式只有输入请求。</p> <p>8'h0: 非周期传送请求队列已满</p> <p>8'h1: 1 个可用位置</p> <p>8'h2: 2 个可用位置</p> <p>N: n 个可用位置 (<math>0 \leq n \leq 8</math>)</p> <p>其他: 保留</p>	8'h08
NPTxFSpAvail	[15: 0]	RO	<p>可用的非周期 TxFIFO 空间</p> <p>指明非周期 TxFIFO 中可用自由空间的数量。这个值是 32 位的。</p> <p>16'h0: 非周期 TxFIFO 已满</p> <p>16'h1: 1 个可用字</p> <p>16'h2: 2 个可用字</p> <p>16'h<sub>n</sub>: n 个可用字 (<math>0 \leq n \leq 32768</math>)</p> <p>其他: 保留</p>	16'h1800

**26.5.1.12.主机周期传送 FIFO 尺寸寄存器（HPTXFSIZ）**

寄存器停止周期 TxFIFO 的尺寸和存储器开始地址。

寄存器	地址	读/写	描述	复位值
HPRXFSIZ	0x7C00_0100	读/写	主机周期传送 FIFO 尺寸寄存器	0x03005A00

HPRXFSIZ	位	读/写	描述	初始状态
PtxFSize	[31:16]	R_ W	主机周期 TxFIFO 深度  这个值是 32 位的。  最小值是 16  最大值是 6144  必须向此位写入一个新的值。程式化的值不能超过电源打开值设置。	16'h0300
PTxFStAddr	[15:0]	R_ W	主机周期转换开始地址  寄存器电源复位值是特殊的最大的 Rx 数据 FIFO 深度和最大的非周期 Tx 数据 FIFO 深度总和。  如果向 Rx FIFO 或非周期 Tx FIFO 程式化一个新值，可以向这个区域写入总和。被程式化得值不能超过电源打开值。	16'h1800

**26.5.1.13.设备周期传送 FIFO-N 尺寸寄存器（DPTXFSIZN）**

FIFO 序号：1≤n≤15

寄存器停止设备模式下的每个周期 TxFIFO 的存储器开始地址。每个周期 FIFO 停止周期输入端点的数据。

寄存器	地址	读/写	描述	复位值
DPRXFSIZn	0x7C00_0104	读/写	设备周期传送 FIFO-n 尺寸寄存	0x0300XXXX

	+ (n-1) *04h		器	
--	--------------	--	---	--

DPRXFSIZn	位	读/写	描述	初始状态
DPTxFSize	[31:16]	R_ W	<p>设备周期 TxFIFO 深度</p> <p>这个值是 32 位的。</p> <p>最小值是 16</p> <p>最大值是 768</p> <p>寄存器的电源复位值是最大的设备模式周期 Tx 数据 FIFO-n 深度。可以向此位写入一个新的值。</p>	<p>n:1(16'h300)</p> <p>n:2(16'h300)</p> <p>n:3(16'h300)</p> <p>n:4(16'h300)</p> <p>n:5(16'h300)</p> <p>n:6(16'h300)</p> <p>n:7(16'h300)</p> <p>n:9(16'h300)</p> <p>n:9(16'h300)</p> <p>n:10(16'h300)</p> <p>n:11(16'h300)</p> <p>n:12(16'h300)</p> <p>n:13(16'h300)</p> <p>n:14(16'h300)</p> <p>n:15(16'h300)</p>
DPTxFStAddr	[15:0]	R_ W	<p>设备周期 TxFIFO RAM 开始地址</p> <p>在 RAM 内停止周期 FIFO 的开始地址。</p> <p>寄存器电源复位值是特殊的最大的 Rx 数据 FIFO 深度和最大的非周期 Tx 数据 FIFO 深度, 以及所有较低编号最大设备模式周期 Tx 数据 FIFO 深度的总和。</p> <p>如果向 RxFIFO 或非周期 TxFIFO 或设备周期 Tx FIFOs 程式化一个新值, 可以向这个区域写入总和。被程式化得值不能超过电源打开值。</p>	<p>n:1(16'h1000)</p> <p>n:2(16'h3300)</p> <p>n:3(16'h3600)</p> <p>n:4(16'h3900)</p> <p>n:5(16'h3C00)</p> <p>n:6(16'h3F00)</p> <p>n:7(16'h4200)</p> <p>n:9(16'h4500)</p> <p>n:9(16'h4800)</p> <p>n:10(16'h4B00)</p> <p>n:11(16'h4E00)</p>



				n:12(16'h5100) n:13(16'h5400) n:14(16'h5700) n:15(16'h5A00)
--	--	--	--	--

26.5.2 主机模式寄存器

主机模式寄存器影响主机模式下的核心操作。在设备模式下不能访问主机模式寄存器。主机模式寄存器分类如下：

- （1）主机全局寄存器
- （2）主机端口控制和状态寄存器
- （3）主机特殊通道寄存器

26.5.2.1. 主机全局寄存器

1.主机配置寄存器（HCFG）

打开电源以后主机配置寄存器配置核心。初始化主机之后不要改变此寄存器

寄存器	地址	读/写	描述	复位值
HCFG	0x7C00_0400	读/写	OTG 控制和状态寄存器	0x00200000

HCFG	位	读/写	描述	初始状态
Reserved	[31:3]		保留	29'h0040000
FSLSSupp	[2]	R_W	只支持 FS 和 LS  应用中，用此位控制核心的列举速度。用此位，应用个可以使核心列举为一个 FS 主机，甚至是连接的设备支持 HS 通信。在初始化程序后不能改变此位。  1'b0: HS/FS/LS，通过连接的设备支持最大速度 1'b1: 只有 FS/LS，相连的设备可以支持 HS。	1'b0

FSLSPclkSel	[1:0]	R_W	FS/LS PHY 时钟选择  当核心处于 FS 主机模式是  2'b00: PHY 时钟是 30/60MHz  2'b01: PHY 时钟是 48MHz  当核心处于 LS 主机模式时  2'b00: PHY 时钟是 30/60MHz  2'b01: PHY 时钟是 48MHz  2'b10: PHY 时钟是 6MHz  2'b11: 保留	2'b0
-------------	-------	-----	--	------

2.主机帧间隔时间寄存器（HFIR）

主机帧将时间寄存器存储核心已经列举的当前速度的帧间隔时间信息。

寄存器	地址	读/写	描述	复位值
HFIR	0x7C00_0404	读/写	主机帧间隔时间寄存器	0x000017D7

HFIR	位	读/写	描述	初始状态
Reserved	[31:16]		保留	16'h0
FrInt	[15:0]	R_W	帧间隔时间  应用中向此位编程的值指定了两个连续的SOF(FS)或 microSOF(HS)或 Keep-Alive(HS)令牌之间的间隔时间。这个区域包含了 PHY 时钟的序号，构成了需要的帧间隔时间。FS 操作的默认值在这个地方设置，前提是 PHY 时钟频率是 60MHz。应用各种可以在主机宽口控制和状态（HPRT.PrtEnaPort）寄存器的端口使能位设置已有向此寄存器写入值。如果没有值可程式化，核心将根据主机配置寄存器内的 FS/LS PHY 时钟选择领域内特殊 PHY 时钟计算此值。在初始化配置以后不能改变此区域的值。  125μs（HS 的 PHY 时钟频率）	16'h17D7

			1ms（FS/LS 的 PHY 时钟频率）	
--	--	--	-----------------------	--

### 3.主机帧序号/帧剩余寄存器（HFNUM）

这个寄存器指明当前帧的序号，同时指明当前帧的剩余时间。

寄存器	地址	读/写	描述	复位值
HFNUM	0x7C00_0408	读/写	主机帧序号/帧剩余时间寄存器	0x00000000

HFNUM	位	读/写	描述	初始状态
FrRem	[31:16]	RO	帧剩余时间  指明当前微帧（HS）或帧（FS/LS）内剩余时间的数量，这个是针对 PHY 时钟而言的。这个区域在每个 PHY 时钟上都有消耗。当他大道零的时候，用帧间隔时间寄存器内的值转移此位，并在 USB 上传输一个新的 SOF。	16'h0
FrNum	[15:0]	RO	帧序号  当一个新的 SOF 在 USB 上传输的时候，帧序号将增加，序号值增加到 16'h3FFF 时，将复位为 0。	16'h0

### 4.主机周期传送 FIFO/QUEUE 状态寄存器（HPTXSTS）

这是一个制度寄存器，其中包含了周 TxFIFO 和 ZHOUQ 传送请求队列的自由空间信息。

寄存器	地址	读/写	描述	复位值
HPTXSTS	0x7C00_0410	读	主机周期传送 FIFO/Queue 状态寄存器	0x00080100

HPTXSTS	位	读/写	描述	初始状态
PTxQTop	[31:24]	RO	周期传送请求队列中顶部  周期 Tx 请求队列的顶层是 MAC 当前运行的内容。  位[31]:奇数/偶数帧  1'b0:发送偶数帧	8'h0

			1'b0: 发送奇数帧 位[30:27]:通道/端点序号 位[26:25]:类型 2'b00: 输入/输出 2'b01: 零长度包 2'b10: CSPLIT 2'b11: 无效的通道命令 位[24]:结束	
PTxQSpAvail	[23:16]	RO	可用的周期传送请求队列空间 指明周期传送请求队列中可用自由空间的数量。这个队列停止主机模式的输入和输出请求。设备模式只有输入请求。 8'h0: 周期传送请求队列已满 8'h1: 1 个可用位置 8'h2: 2 个可用位置 N: n 个可用位置 ( $0 \leq n \leq 8$ ) 其他: 保留	8'h8
PTxFSpAvail	[15: 0]	RO	可用的周期 Tx FIFO 空间 指明周期 Tx FIFO 中可用自由空间的数量。 这个值是 32 位的。 16'h0: 非周期 Tx FIFO 已满 16'h1: 1 个可用字 16'h2: 2 个可用字 16'h $n$ : n 个可用字 ( $0 \leq n \leq 32768$ ) 其他: 保留	16'h0100

## 5.主机所有通道中断寄存器 (HAINT)

通道内发生一个重要事件时，主机所有通道中断寄存器使用核心中断寄存器内的主机通道中断位中断应用。每个通道有一个中断位，最高可达到 16 位。相应的主机通道-n 中断寄存器内设置或清除此位时，

此寄存器内的位会被设置或清除。

寄存器	地址	读/写	描述	复位值
HAINT	0x7C00_0414	读	主机所有通道中断寄存器	0x00000000

HAINT	位	读/写	描述	初始状态
Reserved	[31:16]		保留	16'h0
HAINT	[15:0]	RO	通道中断 一个通道一个中断位： 通道 0 的中断位是 0 通道 15 的中断位是 15	16'h0

**6.主机所有通道中断屏蔽寄存器（HAINTMSK）**

主机所有通道中断屏蔽寄存器和主机所有通道中断寄存器一起工作，当通道内发生事件时中断应用。  
每个通道有一个中断屏蔽位，最大值可达 16 位。

屏蔽中断：1'b0

不屏蔽中断：1'b1

寄存器	地址	读/写	描述	复位值
HAINTMSK	0x7C00_0418	读/写	主机所有通道中断品比寄存器	0x00000000

HAINTMSK	位	读/写	描述	初始状态
Reserved	[31:16]	读/写	保留	16'h0
HAINTMsk	[15:0]	R_W	通道中断屏蔽 一个通道一个中断屏蔽位： 通道 0 的中断位是 0 通道 15 的中断位是 15	16'h0

26.5.2.2.主机端口控制和状态寄存器

1.主机端口控制和状态寄存器（HPRT）

主机端口控制和状态寄存器在主机模式下和设备模式下都可用。目前，OTG 主机只支持一个端口。A 信号寄存器掌握 USB 端口的相关信息，如 USB 复位、使能、暂停、恢复、连接状态以及每个端口的测试模式。此寄存器内的 R\_SS\_WC 位可以触发一个中断穿过核心中断寄存器内的主机端口中断位。在端口中断时，应用中需要读取此寄存器并清除产生中断的位。如 R\_SS\_WC 位，应用中需要向此位写入 1 来清除中断。

寄存器	地址	读/写	描述	复位值
HPRT	0x7C00_0440	读/写	主机端口控制和状态寄存器	0x00000000

HPRT	位	读/写	描述	初始状态
Reserved	[31:19]		保留	13'h0
PrtSpd	[18:17]	RO	端口速度 指明连接到此位的设备的速度 2'b00: 高速 2'b00: 全速 2'b10: 低速 2'b11: 保留	2'b0
PrtTstCtl	[16:13]	R_W	端口测试控制 应用中项此位写入一个非零数值，将此 端口设置为测试模式 2'b00: 高速 4'b0000: 测试模式无效 4'b0001: 测试 J 模式 4'b0010: 测试 K 模式 4'b0011: 测试 SE0_NAK 模式 4'b0100: 测试包模式 4'b0101: Test_Force_Enable 其他: 保留	4'b0

PrtPwr	[12]	R_W_SC	<p>端口电源</p> <p>应用中用此位控制端口电源, 过流条件下核心清除此位</p> <p>1'b0: 电源关闭</p> <p>1'b1: 电源打开</p>	1'b0
PrtLnSts	[11:10]	RO	<p>端口行状态</p> <p>指明当前逻辑电平 USB 数据行</p> <p>位[10]:D-逻辑电平</p> <p>位[11]:D+逻辑电平</p>	2'b0
Reserved	[9]		保留	1'b0
PrtRst	[8]	R_W	<p>端口复位</p> <p>应用中设置此位后, 在端口将开始复位序列。</p> <p>1'b0: 端口不复位</p> <p>1'b1: 端口复位</p> <p>应用中, 需要保留此位用于最低期限下面开始端口复位。</p> <p>高速: 50ms</p> <p>全速/低速: 10ms</p>	1'b0
PrtSusp	[7]	R_WS_SC	<p>端口暂停</p> <p>应用中设置此位, 将端口放置在暂停模式下。只有设置此位后, 核心才可以停止发送 SOFs。为了停止 PHY 时钟, 应用中需要设置端口时钟停止位。此位读取的值将影响端口但前暂停的状态。在检测到一个唤醒信号后, 或者应用中设置端口复位位或设置寄存器内的端口恢复位, 或设置恢复;远程唤醒检测中断位或设置核心中断寄存器内的断开</p>	1'b0

			<p>检测中断位以后，通过核心可以清楚此位。</p> <p>1'b0: 端口不处于暂停模式</p> <p>1'b1: 端口处于暂停模式</p>	
PrtRes	[6]	R_WS_SC	<p>端口恢复</p> <p>应用中设置此位后用于驱动端口的恢复信号。在应用清除此位之前，核心一直驱动恢复信号。如果核心发现一个 USB 远程唤醒序列，核心在没有应用参与的情况下开始驱动恢复信号，在发现未连接时清除此位。此位读取的值指明核心是否是当前驱动恢复信号</p> <p>1'b0: 没有恢复驱动</p> <p>1'b1: 恢复驱动</p>	1'b0
PrtOvrCurrChng	[5]	R_WS_SC	<p>端口过流变化</p> <p>当寄存器内的端口过流活动位状态变化时，核心设置此位。</p>	1'b0
PrtOvrCurrAct	[4]	RO	<p>端口过流活动</p> <p>指明端口过流条件</p> <p>1'b0: 没有过流条件</p> <p>1'b1: 有过流条件</p>	1'b0
PrtEnChng	[3]	R_WS_SC	<p>端口 Enable/Disable 变化</p> <p>当寄存器内的端口使能位[2]的状态变化时，核心设置此位。</p>	1'b0
PrtEna	[2]	R_SS_SC_WC	<p>端口使能</p> <p>在复位序列之后通过核心可以端口使能，过流条件下端口 disable，未连接条件下或清除此位时端口同样 disable。通过寄存器写操作不能设置此位。此位不</p>	1'b0



			向应用触发中断。  1'b0: 端口 disabled  1'b1: 端口 enabled	
PrtConnDet	[1]	R_SS_WC	端口连接测试  当检测到设备连接用核心中断寄存器内的主机端口中断位向应用中触发中断时，核心设置此位。在应用中向此位写入 1 可以清除中断。	1'b0
PrtConnSts	[0]	RO	端口连接状态  1'b0: 没有设备连接到端口上  1'b1: 有设备 连接到端口上	1'b0

26.5.2.3.主机特殊通道寄存器

1.主机通道-N 特性寄存器（HCCHARn）

通道序号：0≤n≤15

寄存器	地址	读/写	描述	复位值
HCCHARn	0x7C00_0500  +n*20h	读/写	主机端口控制和状态寄存器	0x00000000

HCCHARn	位	读/写	描述	初始状态
ChEna	[31]	R_WS_SC	通道使能  通过应用设置此区域，通过 OTG 主机清除此区域	1'b0

			1'b0: 通道 disabled 1'b1: 通道 enabled	
ChDis	[30]	R_WS_SC	端口 disable  应用中设置此位来停止通道内的传送/接收数据，甚至在通道转换完成之间便停止。在触发通道禁止之前需要等待通道使能信号。	1'b0
OddFrm	[29]	R_W	计数帧  通过应用设置此区域，以便指明 OTG 主机必须用奇数帧运行转换。这个区域指那个在周期传送中可用。  1'b0: 偶数帧 1'b1: 奇数帧	1'b0
DevSddr	[28:22]	R_W	设备地址  此区域选择作为数据源或转换器的特殊的设备服务。	7'h0
MC/EC	[21:20]	R_W	多计数/错计数  当主机通道-n 分裂控制寄存器的分裂使能位复位时，这个区域向主机指明处理事务的数量。  2'b00: 保留 2'b01: 处理一事物 2'b10: 每个帧处理两件事物 2'b11: 每帧处理三件事物	2'b0
EPTYPE	[19:18]	R_W	端点类型  指明选择的转换类型  2'b00: 控制 2'b01: 同步 2'b10: 块	2'b0

			2'b11: 中断	
LspdDev	[17]	R_W	低速设备 应用中设置此区域来指明此通道与低速设备通信	1'b0
Reserved	[16]		保留	1'b0
EPDir	[15]	R_W	端点方向端点类型 指明选择的转换类型 1'b0: 输出 1'b1: 输入	2'b0
EPNum	[14:11]	R_W	端点序号 指明作为数据源或转换器的设备服务点的端点序号	4'b0
MPS	[10:0]	R_W	最大包尺寸 指明相关端点的最大包尺寸	11'b0

## 2.主机通道-N 分裂寄存器（HCSPLTn）

通道序号：0≤n≤15

寄存器	地址	读/写	描述	复位值
HCSPLTn	0x7C00_0504 +n*20h	读/写	主机通道 n 分裂寄存器	0x00000000

HCSPLTn	位	读/写	描述	初始状态
SpltEna	[31]	R_W	分裂使能 通过应用设置此区域指明此通道可以运行分裂处理能力	1'b0
Reserved	[30:17]		保留	14'h0
CompSplt	[16]	R_W	完成分裂 通过应用设置此区域来请求 OTG 主机	1'b0

			运行一个完成的分裂交易	
XactPos	[15:14]	R_W	交易位置 此区域用来确定每个输出交易是发送全部负载单元还是第一个，中间位置的或者是最后一个负载单元。  2'b11: 所有的  2'b10: 开始的  2'b00: 中间的  2'b01: 末尾的	2'h0
HubAddr	[13:7]	R_W	Hub 地址 此区域为交易转换器的 HUB 设备地址	7'h0
PrtAddr	[6:0]	R_W	端口地址 此区域为接收交易转换器的端口序号	7'h0

### 3.主机通道-N 中断寄存器（HCINTn）

通道序号：0≤n≤15

此寄存器指明 USB 和 AHB 相关事件方面通道的状态。应用中当核心只能中断寄存器内的主机通道中断位设置时需要读取此寄存器。在应用之前可以读取此寄存器，但必须是第一次读取主机所有通道中断寄存器，以便得到正确的通道序号。应用中需要清除寄存器内适当的位来清除 HAINTE 和 GINTSTS 寄存器内相应的位。

寄存器	地址	读/写	描述	复位值
HCINTn	0x7C00_0508 +n*20h	读/写	主机通道 n 中断寄存器	0x00000000

HCINTn	位	读/写	描述	初始状态
Reserved	[31:11]		保留	21'h0
DataTgIErr	[10]	R_SS_WC	数据切换误差	1'b0

FrnOvrn	[9]	R_SS_WC	帧溢出	1'b0
BblErr	[8]	R_SS_WC	babble 错误	1'b0
XactErr	[7]	R_SS_WC	交易错误	1'b0
NYET	[6]	R_SS_WC	NYET 应答中断	1'b0
ACK	[5]	R_SS_WC	ACK 应答中断	1'b0
NAK	[4]	R_SS_WC	NAK 应答中断	1'b0
STALL	[3]	R_SS_WC	STALL 应答中断	1'b0
AHBErr	[2]	R_SS_WC	AHB 错误 只在内部 DMA 模式下产生，而且在 AHB 读/写操作期间有一个 AHB 错误	1'b0
ChHltd	[1]	R_SS_WC	通道停止 指明由于 USB 交易错误或应答中请求无效等原因异常完成的转化，	1'b0
XferCompl	[0]	R_SS_WC	转换完成 无任何错误的情况下完成转换。	1'b0

#### 4.主机通道-N 中断屏蔽寄存器（HCINTMSKn）

通道序号：0≤n≤15

寄存器影响先前部分中描述的通道状态的屏蔽功能。

屏蔽中断：1' b0

不屏蔽中断：1' b1

寄存器	地址	读/写	描述	复位值
HCINTMSKn	0x7C00_050C +n*20h	读/写	主机通道 n 中断屏蔽寄存器	0x00000000

HCINTMSKn	位	读/写	描述	初始状态
Reserved	[31:11]		保留	21'h0
DataTgIErrMsk	[10]	R_SS_WC	数据切换误差屏蔽	1'b0
FrnOvrnMsk	[9]	R_SS_WC	帧溢出屏蔽	1'b0

BblErrMsk	[8]	R_SS_WC	babble 错误屏蔽	1'b0
XactErrMsk	[7]	R_SS_WC	交易错误屏蔽	1'b0
NYETMsk	[6]	R_SS_WC	NYET 应答中断屏蔽	1'b0
ACKMsk	[5]	R_SS_WC	ACK 应答中断屏蔽屏蔽	1'b0
NAKMsk	[4]	R_SS_WC	NAK 应答中断屏蔽屏蔽	1'b0
STALLMsk	[3]	R_SS_WC	STALL 应答中断屏蔽	1'b0
AHBErrMsk	[2]	R_SS_WC	AHB 错误屏蔽	1'b0
ChHltdMsk	[1]	R_SS_WC	通道停止屏蔽	1'b0
XferComplMsk	[0]	R_SS_WC	转换完成 屏蔽	1'b0

### 5.主机通道-N 转换尺寸寄存器（HCTSIZn）

通道序号：0≤n≤15

寄存器	地址	读/写	描述	复位值
HCTSIZn	0x7C00_0510 +n*20h	读/写	主机通道 n 转换尺寸寄存器	0x00000000

HCTSIZn	位	读/写	描述	初始状态
DoPng	[31]	R_W	DO Ping 将此位设置为 1 指示主机座 PING 协议	1'h0
Pid	[30:29]	R_W	PID 应用用 PID 的类型运行此位，用于初始化交易。主机将保持此领域，为了转换复位。  2'b00: DATA0 2'b01:DATA1 2'b10:DATA2 2'b11:MDATA	2'b0
PktCnt	[28:19]	R_W	包数 主机在每次成功的传送或接收一个输出/输入包后减少计数。 一旦此计数为零，应用被中断。	10'b0F

XferSize	[18:0]	R_W	转换尺寸 对输出而言，此区域位主机将要发送的数据字节数量。 对输入而言，此区域是缓冲区尺寸。	19'b0
----------	--------	-----	--	-------

### 6.主机通道-N DMA 地址寄存器（HCDMA<sub>n</sub>）

通道序号：0 ≤ n ≤ 15

寄存器	地址	读/写	描述	复位值
HCDMA <sub>n</sub>	0x7C00_0514 +n*20h	读/写	主机通道 nDMA 地址寄存器	0x00000000

HCDMA <sub>n</sub>	位	读/写	描述	初始状态
DMAAddr	[31:0]	R_W	DMA 地址 此区域掌握外部存储器的开始地址，需要取得外部存储器的断点数据或者存储外部存储器的数据。每发生一次 AHB 转换，此就寄存器将会增加一次。	32'h0

## 26.5.3. 设备模式寄存器

设备模式寄存器只有在设备模式下可见，而且在主机模式下不能访问设备模式寄存器。有些设备模式寄存器将会影响所有的端点，其他的设备模式寄存器只会影响一些特殊的端点。设备模式寄存器主要分为两类：

- （1）设备全局寄存器
- （2）设备特定逻辑端点寄存器

### 26.5.3.1. 设备全局寄存器

#### 1.设备配置寄存器（DCFG）

在打开电源之后或者在一定的控制命令或列举之后，设备配置寄存器配置核心。在寄存器初始化运行之后不要改变寄存器。

寄存器	地址	读/写	描述	复位值
DCFG	0x7C00_0800	读/写	设备配置寄存器	0x00200000

DCFG	位	读/写	描述	初始状态
Reserved	[31:23]		保留	9'h0
EPMisCnt	[22:18]	R_W	<p>输入端点不匹配计数</p> <p>应用中通过计数运行此区域，来确定核心什么时候产生端点不匹配中断。核心向内部计数器转载计数值，并增加内部计数器的值。计数器数值在不停的移动变化。计数器的宽度有令牌队列的深度决定。</p>	5'h8
Reserved	[17:13]		保留	5'h0
PerFrInt	[12:11]	R_W	<p>周期帧间隔</p> <p>指明帧内的时间，主要应用在周期帧中断末尾。可以用来确定帧的所有同步运输是否完成。</p> <p>2'b00: 帧间隔的 80%</p> <p>2'b01: 85%</p> <p>2'b10:90%</p> <p>2'b11:95%</p>	2'h0
DevAddr	[10:4]	R_W	<p>设备地址</p> <p>应用中必须在每个设置地址控制命令之后运行此区域</p>	7'h0
Reserved	[3]		保留	1'b0
NZStsOUTHShk	[2]	R_W	<p>非零长度状态输出交换</p> <p>应用中可以用此区域选择核心发出的接收一个非零长度数据包的交换，此交换是在控制转换的状态阶段的输出交易时间内。</p>	1'b0
DevSpd	[1:0]	R_W	<p>设备速度</p> <p>指明应用中需要核心列举的速度，或者应用中可以支持的最大速度。但是实际总线速度只有在线性调频脉冲序列完成之后才能决定，实际总线速度基于核心连接的 USB 主机的速度。</p> <p>2'b00: 高速（USB 2.0 PHY 时钟是 30MHz 或 60MHz）</p> <p>2'b01: 全速（USB 2.0 PHY 时钟是 30MHz 或 60MHz）</p>	2'b0



			2'b10:低速（USB1.1 收发器时钟是 6MHz），如果选择 6MHz 的 LS 模式，必须做一个软件复位。	
			2'b11:全速（USB1.1 收发器时钟是 48MHz）	

## 2.设备控制寄存器（DCTL）

寄存器	地址	读/写	描述	复位值
DCTL	0x7C00_0804	读/写	设备配置寄存器	0x00000000

DCTL	位	读/写	描述	初始状态
Reserved	[31:12]		保留	20'h0
PWROnPrgDone	[11]	R_W	电源开启运行 应用中用此位指明在省电模式唤醒后寄存器运行完成。	1'b0
CGOUTNak	[10]	WO	清除全局输出 NAK 此区域的一个写操作将清除全局输出 NAK	1'b0
SGOUTNak	[9]	WO	设置全局输出 NAK 此区域写操作将设置全局输出 NAK 应用中用此位在所有的输出端点上发送一个 NAK 交易。 应用中必须设置此位，在确定核心中断寄存器内的全局输出 NAK 有效位被清除之后。	1'b0
CGNPInNAK	[8]	WO	清除全局非周期输入 NAK 此区域的一个写操作将清除全局非周期输入 NAK	1'b0
SGNPInNAK	[7]	WO	设置全局非周期输入 NAK 此区域写操作将设置全局非周期输入 NAK 应用中用此位在所有的非周期输入端点上发送一个 NAK 交易。当在非周期端点上检测到超时条件是，核心也可以设置此位。应用中必须设置此位，在确定核心中断寄存器内的全局输入 NAK 有效位被清除之后。	1'b0
TstCtl	[6:4]	R_W	测试控制 3'b000：测试模式无效	3'b0

			3'b001: 测试 J 模式 3'b010: 测试 K 模式 3'b011: 测试 SE0_NAK 模式 3'b100: 测试包模式 3'b101: Test_Force_Enable 其他: 保留	
GOUTNakSts	[3]	RO	全局输出 NAK 状态 1'b0:在 FIFO 状态以及设置 NAK 和 ATALL 位的情况下发送一个交易 1'b1:没有数据写入 RxFIFO。在所有包上发送 NAK 交易,除了 SETUP 处理意外。	1'b0
GNPINNakSts	[2]	RO	全局非周期输入 NAK 状态 1'b0:在传输 FIFO 内数据可用的基础上发送一个交易 1'b1:在所有非周期输入端点上发送 NAK 交易,不考虑传输 FIFO 内数据的可用能力。	1'b0
SftDiscon	[1]	R_W	软中断 应用中用此位向 OTG 核心发送信号,进行一个软中断。只要此位被设置,主机将不会看到设备已经连接上,同时设备将不会接受到 USB 上的信号。核心一直处于中断状态,知道应用中清除此位为止。 1'b0: 常规操作。在软中断后清除此位,核心在 UTMI+to 2'b00 上驱动 opmode 信号,可以向 USB 主机产生一个设备连接事件。当设备重新连接上后,USB 主机开始驱动列举。 1'b01: 核心在 UTMI+to 2'b01 上驱动 opmode 信号,向 USB 主机产生一个设备中断事件。	1'b0
RmtWkUpSig	[0]	R_W	远程唤醒信号 当应用中设置此位后,核心初始化远程信号来唤醒 USB 主机。 应用中需要设置此位来指示核心推出暂停状态。如在 USB 2.0 说明书中定义的一样,应用中需要在设置此位以后清除此位	1'b0

			1-15ms	
--	--	--	--------	--

下面的列表列出了不同条件下的最小期限， USB 主机的软中断位必须被设置，用于检查设备中断。  
为了容纳时钟抖动，建议应用中在明确的最小期限内增加一些额外延迟。

运行速度	设备状态	最小期限
高速	暂停	1ms+2.5μs
高速	闲置	3ms+2.5μs
高速	运行处理	125μs
全速/低速	暂停	1ms+2.5μs
全速/低速	闲置	2.5μs
全速/低速	运行处理	2.5μs

### 3. 设备状态寄存器（DSTS）

寄存器	地址	读/写	描述	复位值
DSTS	0x7C00_0808	读/写	设备状态寄存器	0x00000002

DCTL	位	读/写	描述	初始状态
Reserved	[31:22]		保留	10'h0
SOFFN	[21:8]	RO	当核心在高速下运行时，接收到的 SOF 帧或微帧数量；此区域包含微帧的数量。当核心运行在全速或低速时，此区域包含帧数量。	14'h0
Reserved	[7:4]		保留	4'h0
ErrticErr	[3]	RO	Erratic 错误 核心设置此位来报告在 UTMI+上发现的任何 Erratic 错误。由于 Erratic 错误，OTG 核心进入暂停状态，而且将向和姓中断寄存器内的早期暂停位的应用产生中断。如果早期暂停由于 erratic 错位被声明，应用可以只运行一个软中断恢复。	1'b0
EnumSpd	[2:1]	RO	列举速度	2'b01

			<p>指明 OTG 核心在通过 chirp 序列的高速检测后达到的数据。</p> <p>2'b00: 高速 (PHY 时钟是 30MHz 或 60MHz)</p> <p>2'b01: 全速 (PHY 时钟是 30MHz 或 60MHz)</p> <p>2'b10: 低速 (PHY 时钟 6MHz), 如果选择 6MHz 的 LS 模式, 必须做一个软件复位。</p> <p>2'b11: 全速 (PHY 时钟是 48MHz)</p>	
SuspSts	[0]	RO	<p>暂停状态</p> <p>在设备模式下, 只要 USB 检测到暂停条件, 此位就被设置。当较长时间内 line_state 信号都没被激活时, 核心进入暂停状态。当 line_state 信号被激活时或当应用中项设备控制寄存器内的远程唤醒信号位写入数据时, 核心跳出暂停状态。</p>	1'b0

#### 4.设备输入端点通用中断屏蔽寄存器 (DIEPMSK)

此寄存器同设备输入端点中断寄存器一起工作于所有的端点上, 用于每个输入端点产生中断。

DIEPINTn 寄存器内输入端点中断的特殊状态可以通过向此寄存器内的相应位写入数据进行屏蔽。默认状态位是被屏蔽的。

屏蔽中断: 1'b0

不屏蔽中断: 1'b1

寄存器	地址	读/写	描述	复位值
DIEPMSK	0x7C00_0810	读/写	设备输入端点通用中断屏蔽	0x00000000

DIEPMSK	位	读/写	描述	初始状态
Reserved	[31:7]		保留	25'h0
INEPNakEffMsk	[6]	R_W	输入端点 NAK 有效屏蔽	1'b0
INTknEPMisMsk	[5]	R_W	接收的输入令牌 EP 不匹配屏蔽	1'b0
INTknTXFEnpMsk	[4]	R_W	接收的输入令牌 TxFIFO 空闲屏蔽	1'b0
TimeOUTMsk	[3]	R_W	超时条件屏蔽	1'b0

AHBErrMsk	[2]	R_W	AHB 错误屏蔽	1'b0
EPDisbldMsk	[1]	R_W	端点无效中断屏蔽	1'b0
XferCompIMsk	[0]	R_W	转换完成中断屏蔽	1'b0

### 5.设备输出端点通用中断屏蔽寄存器（DOEPMASK）

此寄存器同设备输出端点中断寄存器一起工作于所有的端点上，用于每个输出端点产生中断。

DIEPINTn 寄存器内的输出端点中断的特殊状态可以通过向此寄存器内的相应的位写入数据进行屏蔽。默认状态位是被屏蔽的。

屏蔽中断：1'b0

不屏蔽中断：1'b1

寄存器	地址	读/写	描述	复位值
DOEPMASK	0x7C00_0814	读/写	设备输出端点通用中断屏蔽	0x00000000

DOEPMASK	位	读/写	描述	初始状态
Reserved	[31:7]		保留	27'h0
Back2BackSETup	[6]	R_W	接收的 Back to Back SETUP 包屏蔽，只应用于控制输出端点	1'b0
Reserved	[5]	R_W	保留	1'b0
OUTTknEPdisMsk	[4]	R_W	当端点无效时接收的输出令牌，只应用于控制输出端点	1'b0
SetUPMsk	[3]	R_W	SETUP 阶段操作屏蔽，只应用于控制输出端点	1'b0
AHBErrMsk	[2]	R_W	AHB 错误	1'b0
EPDisbldMsk	[1]	R_W	端点无效中断屏蔽	1'b0
XferCompIMsk	[0]	R_W	转换完成中断屏蔽	1'b0

### 6.设备所有端点中断寄存器（DAINT）

当端点发生一个明显事件时，设备所有端点中断寄存器使用核心中断寄存器内的设备输出端点中断位或设备输入端点中断位中断应用。每个端点都有一个中断位，输出端点和输入端点的最大值可达 16 位。对一个双向端点而言，将会使用其相应的输入输出中断位。当设置或清除相应设备端点-n 中断寄存器内的

适当位时可以设置或清除此寄存器内的位。

寄存器	地址	读/写	描述	复位值
DAINT	0x7C00_0818	读/写	设备所有端点中断寄存器	0x00000000

DAINT	位	读/写	描述	初始状态
OutEPInt	[31:16]	RO	输出端点中断位 每个位都有一个输出中断：输出端点 0 的中断位是第 16 位，输出端点 15 的中断位是第 31 位。	16'h0
InEPInt	[15:0]	RO	输入端点中断位 每个位都有一个输入中断：输入端点 0 的中断位是第 0 位，输入端点 15 的中断位是第 15 位。	16'h0

**7.设备所有端点中断屏蔽寄存器（DAINTMSK）**

设备端点中断屏蔽寄存器同设备端点中断寄存器一起工作，当设备端点发生事件时，中断应用。与中断相对应的设备所有端点中断寄存器将被设置。

屏蔽中断：1'b0

不屏蔽中断：1'b1

寄存器	地址	读/写	描述	复位值
DAINTMAK	0x7C00_0818	读/写	设备所有端点中断屏蔽寄存器	0x00000000

DAINTMAK	位	读/写	描述	初始状态
OutEPIntMsk	[31:16]	RO	输出端点中断屏蔽位 每个位都有一个输出中断屏蔽：输出端点 0 的中断屏蔽位是第 16 位，输出端点 15 的中断屏蔽位是第 31 位。	16'h0
InEPIntMsk	[15:0]	RO	输入端点中断屏蔽位 每个位都有一个输入中断屏蔽：输入端点 0 的中断屏蔽位是第 0 位，输入端点 15 的中断屏蔽位是第 15 位。	16'h0

### 8.设备输入令牌序列学习队列读寄存器 1（DTKNQR1）

此队列宽度为 4 位，用于储存端点序号。此寄存器内的一个读取将返回输入令牌序列学习队列中的前五个端点。当队列已满时，队列内将被推入新的令牌，旧的令牌被丢弃。

寄存器	地址	读/写	描述	复位值
DTKNQR1	0x7C00_0820	读/写	设备输入令牌序列学习队列读寄存器 1	0x00000000

DTKNQR1	位	读/写	描述	初始状态
EPTkn	[31:8]	RO	端点令牌 每个令牌位代表令牌的断点序号 位[31:28]:令牌 5 的断点序号 位[27:24]: 令牌 4 的端点序号 …… 位[15:12]:令牌 1 的断点序号 位[11:8]: 令牌 0 的断点序号	24'h0
WrapBit	[7]	RO	位 当写指针重叠时设置此位。当学习队列被清除时，此位被清除。	1'b0
Reserved	[6:5]	RO	保留	2'h0
INTKnWPtr	[4:0]	RO	输入令牌队列写指针	5'h0

### 9.设备输入令牌序列学习队列读寄存器 2（DTKNQR2）

此寄存器内的一个读取将返回输入令牌序列学习队列中的下 8 个端点。

寄存器	地址	读/写	描述	复位值
DTKNQR2	0x7C00_0824	读/写	设备输入令牌序列学习队列读寄存器 2	0x00000000

DTKNQR2	位	读/写	描述	初始状态
EPTkn	[31:0]	RO	端点令牌 每个令牌四个位代表令牌的断点序号 位[31:28]:令牌 13 的断点序号	32'h0

			位[27:24]: 令牌 12 的端点序号 ..... 位[7:4]:令牌 7 的断点序号 位[3:0]: 令牌 6 的断点序号	
--	--	--	---	--

**10.设备输入令牌序列学习队列读寄存器 3（DTKNQR3）**

此寄存器内的一个读取将返回输入令牌序列学习队列中的下 8 个端点。

寄存器	地址	读/写	描述	复位值
DTKNQR3	0x7C00_0830	读/写	设备输入令牌序列学习队列读寄存器 3	0x00000000

DTKNQR3	位	读/写	描述	初始状态
EPTkn	[31:0]	RO	端点令牌 每个令牌四个位代表令牌的断点序号 位[31:28]:令牌 21 的断点序号 位[27:24]: 令牌 20 的端点序号 ..... 位[7:4]:令牌 15 的断点序号 位[3:0]: 令牌 14 的断点序号	32'h0

**11.设备输入令牌序列学习队列读寄存器 4（DTKNQR4）**

此寄存器内的一个读取将返回输入令牌序列学习队列中的下 8 个端点。

寄存器	地址	读/写	描述	复位值
DTKNQR4	0x7C00_0834	读/写	设备输入令牌序列学习队列读寄存器 4	0x00000000

DTKNQR4	位	读/写	描述	初始状态
EPTkn	[31:0]	RO	端点令牌 每个令牌四个位代表令牌的断点序号 位[31:28]:令牌 29 的断点序号	32'h0



			位[27:24]: 令牌 28 的端点序号 ..... 位[7:4]:令牌 23 的断点序号 位[3:0]: 令牌 22 的断点序号	
--	--	--	---	--

### 12.设备 VBUS 放电时间寄存器（DVBUSDIS）

此寄存器定义了 SRP 期间 Vbus 脉冲之后的 Vbus 放电时间

寄存器	地址	读/写	描述	复位值
DVBUSDIS	0x7C00_0828	读/写	设备 VBUS 放电时间寄存器 2	0x000017D7

DVBUSDIS	位	读/写	描述	初始状态
Reserved	[31:16]		保留	16'h0
DVBUSDis	[15:0]	R_W	设备 Vbus 放电时间 指定 SRP 期间在 Vbus 脉冲之后的 Vbus 放电时间， 此值等于： PHY 时钟内的 Vbus 放电时间/1024	16'h17D7

### 13.设备 VBUS 脉冲时间寄存器（DVBUSPULSE）

此寄存器定义了 SRP 期间 Vbus 脉冲时间

寄存器	地址	读/写	描述	复位值
DVBUSPULSE	0x7C00_082C	读/写	设备 VBUS 脉冲时间寄存器 2	0x000005B8

DVBUSPULSE	位	读/写	描述	初始状态
Reserved	[31:12]		保留	16'h0
DVBUSPulse	[11:0]	R_W	设备 Vbus 脉冲时间 指定 SRP 期间 Vbus 脉冲时间，此值等于： PHY 时钟内的 Vbus 脉冲时间/1024	16'h5B8

26.5.3.2.设备特定逻辑端点寄存器

逻辑端点是单向的：可以用于输入或输出。为了代表双向端点，需要两个逻辑端点，一个用于输入方向，一个用于输出方向。这个部分描述的寄存器和寄存器区域会涉及到输入端点和输出端点，或者两者都会，或者为标准的特定类型端点。

1.设备控制输入端点 0 控制寄存器（DIEPCTL0）

这个部分描述控制输入端点 0 控制寄存器。

寄存器	地址	读/写	描述	复位值
DIEPCTL0	0x7C00_0900	读/写	设备控制输入端点 0 控制寄存器	0x00008000

DIEPCTL0	位	读/写	描述	初始状态
EPEna	[31]	R_WS_SC	端点使能  指明数据已经准备好在端点传送。在这个端点上，在设置端点禁止，和转换完成之前，核心清除此位。	1'b0
EPDis	[30]	R_WS_SC	端点禁止  应用中设置此位在端点上停止传送数据,甚至在通道转换完成之间便停止。在触发通道禁止之前需要等待通道禁止信号。核心在设置断点禁止中断之前设置此位。	1'b0
Reserved	[29:28]		保留	2'b0
SetNAK	[27]	WO	设置 NAK  此位的写操作将设置断点的 NAK。  应用中可以用此位控制端点上的 NAK 交易的处理。核心同时可以在端点接收到 SETUP 包以后设置此位。	1'b0
CANK	[26]	WO	清除 NAK  此位的写操作可以清除端点的 NAK 位。	1'b0
TxFNum	[25:22]	RO	TxFIFO 序号	4'h0

			此值通常被设置为 0, 指明非周期传送 FIFO 中的控制输入端口 0 数据经常被写入。	
Stall	[21]	RO	<b>STALL</b> 信号交换  应用中可以只设置此位, 当此端点接收到 <b>SETUP</b> 令牌时, 核心清除此位。如果设置此位的同时也设置了 <b>NAK</b> 位。全局非周期输入 <b>NAK</b> 位, 或全局输出 <b>NAK</b> 位, <b>STALL</b> 位的优先级最高。	1'b0
Reserved	[20]		保留	1'b0
EPTYPE	[19:18]	RO	端点类型  硬编码为 00 的控制	2'h0
NAKsts	[17]	RO	<b>NAK</b> 状态  指明下面情况  1'b0:核心传输基于 FIFO 状态非 <b>NAK</b> 信号交换。 1'b1:核心在断电传输 <b>NAK</b> 信号交换。  当此位设置后, 核心可以停止传输数据, 即使数据在 <b>TxFIFO</b> 中有效。不管此位的设置, 核心通常可以用 <b>ACK</b> 信号交换回应 <b>SETUP</b> 数据包。	1'b0
Reserved	[16]		保留	1'b0
USBActEP	[15]	RO	<b>USB</b> 活动端点  此位通常设置为 1, 指明控制端口 0 通常活动在所有的配置和接口中。	1'b1
NextEp	[14:11]	R_W	下一个端点  只应用于非周期输入端点  指明在取回当前端点数据后, 取回端点序号。核心可以访问此区域, 甚至在端点使能位没有设置的情况下。Slave 模式操作下此区域无效。	4'b0
Reserved	[10:2]		保留	9'h0
MPS	[1:0]	R_W	最大包尺寸  应用在输入或输出端点。	2'h0

			应用中，当前逻辑端点必须用最大的包尺寸运行	
			2'b00:64 字节	
			2'b01:32 字节	
			2'b10:16 字节	
			2'b11:8 字节	

2.设备控制输出端点 0 控制寄存器（DOEPCTL0）

这个部分描述控制输出端点 0 控制寄存器。

寄存器	地址	读/写	描述	复位值
DOEPCTL0	0x7C00_0B00	读/写	设备控制输出端点 0 控制寄存器	0x00008000

DOEPCTL0	位	读/写	描述	初始状态
EPEna	[31]	R_WS_SC	端点使能  指明应用中已经分配好寄存器开始从 USB 接收数据。核心在设置端点的 SETUP Phase Done 中断、端点禁止中断 和转换完成中断之前清除设置。  注：在 DMA 模式下，为了核心向存储器转换 SETUP 数据包，此位必须被设置。	1'b0
EPDis	[30]	RO	端点 disable  应用中不可以是控制输出端点 0 无效。	1'b0
Reserved	[29:28]		保留	2'b0
SetNAK	[27]	WO	设置 NAK  此位的写操作将设置端点的 NAK。  应用中可以用此位控制端点上的 NAK 交易的处理。核心同时可以在端点接收到 SETUP 包以后设置此位。	1'b0
CANK	[26]	WO	清除 NAK  此位的写操作可以清除端点的 NAK 位。	1'b0
Reserved	[25:22]		保留	4'h0
Stall	[21]	R_WS_SC	STALL 信号交换	1'b0

			应用中可以只设置此位，当此端点接收到 <b>SETUP</b> 令牌时，核心清除此位。如果设置此位的同时也设置了 <b>NAK</b> 位或全局输出 <b>NAK</b> 位， <b>STALL</b> 位的优先级最高。不考虑此位的设置，核心通常可以通过 <b>ACK</b> 信号交换回应 <b>SETUP</b> 数据包。	
Snp	[20]	R_W	窥探模式  此位配置端点为窥探模式。在窥探模式下，在向应用存储器传输包之前，核心不能坚持输出包的正确性。	1'b0
EPTYPE	[19:18]	RO	端点类型  硬编码为 00 的控制	2'h0
NAKsts	[17]	RO	<b>NAK</b> 状态  指明下面情况  1'b0:核心传输基于 FIFO 状态非 <b>NAK</b> 信号交换。  1'b1:核心在断点传输 <b>NAK</b> 信号交换。  当此位设置后，核心可以停止传输数据，即使数据在 <b>TxFIFO</b> 中有效。不管此位的设置，核心通常可以用 <b>ACK</b> 信号交换回应 <b>SETUP</b> 数据包。	1'b0
Reserved	[16]		保留	1'b0
USBActEP	[15]	RO	<b>USB</b> 活动端点  此位通常设置为 1，指明控制端口 0 通常活动在所有的配置和接口中。	1'b1
Reserved	[14:2]		保留	13b0
MPS	[1:0]	RO	最大包尺寸  应用在输入或输出端点。  应用中，当前逻辑端点必须用最大的包尺寸运行  2'b00:64 字节  2'b01:32 字节  2'b10:16 字节  2'b11:8 字节	2'h0

3.设备端点-N 控制寄存器（DIEPCTLn/DOEPCTLN）

端点序号：1≤n≤15

应用中用此寄存器控制除了端点 0 意外的每个逻辑端点的行为。

寄存器	地址	读/写	描述	复位值
DIEPCTLn/DOEPCTLn	0x7C00_0900+n*20h /0x7C00_0B00+n*20h	读/写	设备端点-n 控制寄存器	0x00000000

DIEPCTLn/DOEPCTLn	位	读/写	描述	初始状态
EPEna	[31]	R_WS_SC	端点使能  应用于输入端点和输出端点。  对输入端点而言，此位指明数据已经准备好在端点传送。对输出端点而言，此位指明应用中已经分配好寄存器开始从 USB 接收数据。核心在设置端点的 SETUP Phase Done 中断、端点禁止中断 和转换完成中断之前清除设置。  注：对于 DMA 模式内的控制输出端点而言，此位必须设置能够在存储器内钻换 SETUP 数据包。	1'b0
EPDis	[30]	RO	端点禁止  应用于输入端点和输出端点。  应用中设置此位停止端点上的传输和接收，及时在端点转换未完成的情况下也会停止。在设置端点禁止中断之前，核心清除此位。如过此端点的端点使能已经被设置，应用中必须设置此位。	1'b0
SetD1PID	[29]	WO	设置 DATA1 PID  应用于中断/块输入和输出端点。  向此区域写入数据，设置寄存器内的端点数据 PID 区域为 DATA1	1'b0
SetOddFr			设置奇数帧  应用于微帧输入和输出端点	

			<p>向此区域写入数据设置偶数/计数帧区域为奇数帧。</p>	
SetD0PID	[28]	WO	<p>设置 DATA0PID</p> <p>应用于中断/块输入和输出端点。</p> <p>向此区域写入数据，设置寄存器内的端点数据 PID 区域为 DATA0</p>	1'b0
SetEvenFr			<p>设置偶数帧</p> <p>应用于微帧输入和输出端点</p> <p>向此区域写入数据设置偶数/计数帧区域为奇数帧。</p>	
SNAK	[27]	WO	<p>设置 NAK</p> <p>应用于输入和输出端点</p> <p>此位的写操作将设置端点的 NAK。</p> <p>应用中可以用此位控制端点上的 NAK 交易的处理。核心同时可以在端点接收到 SETUP 包，或转换完成的中断以后设置此位。</p>	1'b0
CANK	[26]	WO	<p>清除 NAK</p> <p>应用于输入和输出端点。</p> <p>此位的写操作可以清除端点的 NAK 位。</p>	1'b0
TxFNum	[25:22]	R_W	<p>TxFIFO 序号</p> <p>只应用于输入端点。</p> <p>非周期端点必须设置此位为 0.周期端点必须映此位到相应的 TxFIFO 序号上</p>	4'h0
Stall	[21]	R_W	<p>STALL 信号交换</p> <p>应用于非控制，非同步输入和输出端点。应用中设置此位停止此端口的 USB 主机的所有令牌。如果一个 NAK 位、全局非周期输入 NAK 位或全局输出 NAK 位于此位同时被设置，STALL 位有优先权。只有应用可以清除此位，核心不能清除此位。</p>	1'b0

		R_WS_SC	<p>应用于控制端点</p> <p>当此端点接收到一个 SETUP 令牌时，只有在应用中设置此位，核心清除此位。如果一个 NAK 位、全局非周期输入 NAK 位或全局输出 NAK 位于此位同时被设置，STALL 位有优先权。不考虑此位的设置，核心通常可以用 ACK 信号交换回应 SETUP 数据包。</p>	
Snp	[20]	R_W	<p>窥探模式</p> <p>之应用于输出端点</p> <p>此位配置端点为窥探模式。在窥探模式下，在向应用存储器传输包之前，核心不能坚持输出包的正确性。</p>	1'b0
EPTYPE	[19:18]	RO	<p>端点类型</p> <p>应用于输入和输出端点。</p> <p>这是有逻辑端点支持的转换类型</p> <p>2'b00:控制</p> <p>2'b01: 同步</p> <p>2'b10: 块</p> <p>2'b11: 中断</p>	2'h0
NAKsts	[17]	RO	<p>NAK 状态</p> <p>应用于输入和输出端点。</p> <p>指明下面情况</p> <p>1'b0:核心传输基于 FIFO 状态的非 NAK 信号交换。</p> <p>1'b1:核心在断电传输 NAK 信号交换。</p> <p>当应用或核心设置此位时：</p> <p>（1）核心停止在输出端点接收任何数据，即使 RxFIFO 内有空间容纳新的数据包。</p> <p>（2）对非同步输入端口而言，核心停止在输入端口的任何数据的传输，即使 TxFIFO 内的数据有效。</p> <p>（3）对同步输入端点而言：核心发送出零长度的数据包，即使 TxFIFO 内的数据有效。</p>	1'b0



			不考虑此位的设置是,核心通常用 ACK 数据交换回应 SETUP 数据包。	
DPID	[16]	RO	<p>端点数据 PID</p> <p>应用于中断/块输入或输出端点。</p> <p>包括端点接收到或传输的包的 PID。端点激活后,应用中需要在端点上运行第一个接收到的或传输的包的 PID。用寄存器内的 SetD1PID 和 SetD0PID 区域运行 DATA0 或 DATA1 PID.</p> <p>1'b0: DATA0</p> <p>1'b1: DATA1</p>	1'b0
EO_FrNum			<p>偶数/奇数帧</p> <p>应用于同步输入和输出端点。</p> <p>指明此端点的核心传输/接收到得同步数据的帧序号。应用中,需要运行打算用寄存器呢 SetEvenFr 和 SetOddFr 区域传输/接收同步数据的偶数/奇数帧的序号。</p> <p>1'b0: 偶数帧</p> <p>1'b1: 计数真</p>	
USBActEP	[15]	R_W_SC	<p>USB 活动端点</p> <p>应用于输入和输出端点。</p> <p>指明当前配置和接口内的断点是否是激活的。在发现 USB 复位以后,核心清除所有端点的此位。在接收到设置配置和设置接口命令后,应用必须正确的运行端点寄存器并设置此位。</p>	1'b0
NextEp	[14:11]	R_W	<p>下一个端点</p> <p>应用于非周期输入端点。</p> <p>指明在取回当前端点数据后取回端点序号。核心可以访问此区域,即使端点使能位处于低电平。在 Slave 模式下此位无效。</p>	4'h0
MPS	[10:0]	R_W	<p>最大包尺寸</p> <p>应用在输入或输出端点。</p>	11'h0

			应用中，当前逻辑端点必须用最大的包尺寸运行.	
--	--	--	------------------------	--

#### 4.设备端点-N 中断寄存器（DIEPINTn/DOEPINTn）

端点序号：1≤n≤15

这个寄存器指明了于 USB 和 AHB 相关时间的端点的状态。应用中，当设置核心中断寄存器的输出端点中断位或输入端点中断位以后，必须读取此寄存器。在应用读取此寄存器之前，必须首先读取设备所有端点中断（DAINT）寄存器，取得设备端点-n 中断寄存器的正确端点序号。应用中需要清除此寄存器内适当的位来清除 DAINTE 和 GINTSTS 寄存器内相应的位。

寄存器	地址	读/写	描述	复位值
DIEPINTn/DOEPINTn	0x7C00_0908+n*20h /0x7C00_0B08+n*20h	读/写	设备端点-n 中断寄存器	0x00000080

DIEPINTn/DOEPINTn	位	读/写	描述	初始状态
EPEna	[31:7]		保留	25'h0
INEPNakEff	[6]	RO	输入端点 NAK 有效 应用于周期输入端点。 指明只输入端点 NAK 位卑核心内的应用设置。当应用通过向 DIEPCTLn.CNAK 内写入数据来清除输入端点 NAK 时，此位被清除。中断显示核心有相似的 NAK 位设置。	1'b0
Back2BackSETup		R_W	接收的 Back to Back SETUP 包，只应用于控制输出端点。 此位指明对于这一特定端点，核心接收到多于三个的 Back to Back SETUP 包。	
INTKnEPMis	[5]	R_SS_WC	接收的 EP 不匹配输入令牌 应用于周期输入端点。 指明属于端点的非周期 TxFIFO 的顶层数据，除了输入令牌受到的那个。中断在端点上声明输入令牌已经接收到。 对输出端点而言，此位保留	1'b0

INTKnTXFEmp	[4]	R_SS_WC	<p>当 TxFIFO 为空时接收的输入令牌</p> <p>应用于非周期输入端点。</p> <p>指明当相关的 TxFIFO 为空时，接收输入令牌。中断在端点上声明输入令牌已经接收到。</p>	1'b0
INTKnTXFEmp			<p>当端点禁止应用控制输出端点是，输出令牌被接收。</p> <p>指明当端点还未使能时，输出令牌被接收到。中断在端点上声明输出令牌已经接收到。</p>	
TimeOUT	[3]	R_SS_WC	<p>超时条件</p> <p>应用于非同步输入端点</p> <p>指明核心在 USB 上检测到此端点最后输入令牌的超时条件</p>	1'b0
SetUp			<p>设置阶段</p> <p>之应用与控制地输出端点。</p> <p>指明控制端点的设置阶段已经完成，而且当期按控制转换器内有接收到多于的 Back to Back SETUP 包。在此中断上，应用可以对接收到的设置数据包进行解码。</p>	
AHBErr	[2]	R_SS_WC	<p>AHB 错误</p> <p>应用于输入和输出端点</p> <p>只有在内部 DMA 模式下而且同时在 AHB 读/写期间有 AHB 错误时才产生。</p> <p>应用个可以读取相应的端点 DMA 地址寄存器以取得错误地址。</p>	1'b0
EPDisbld	[1]	R_SS_WC	<p>端点 Disabled 中断</p> <p>应用于输入和输出端点</p> <p>此位表示每个应用请求端点无效。</p>	1'b0
XferCompl	[0]	R_SS_WC	<p>转换完成 中断</p> <p>指明此端点运行的转换在 AHB 和 USB 上都以完成</p>	1'b0

## 5.设备端点 0 转换尺寸寄存器（DIEPTSIZ0）

实际应用中，在端点 0 使能以前必须修改此寄存器。一旦端点 0 使用设备控制端点 0 控制寄存器的端

点使能位使能，核心修改此寄存器。当核心已经清除端点使能位时，应用只能读取此寄存器。

寄存器	地址	读/写	描述	复位值
DIEPISIZ0	0x7C00_0910	读/写	设备输入端点 0 转换尺寸寄存器	0x00000000

DIEPISIZ0	位	读/写	描述	初始状态
Reserved	[31:21]		保留	11'h0
PktCnt	[20:19]	R_W	包计数 指明 USB 包的总的数量,USB 包的数量构成端口点 0 数据的换尺寸数量。 每从 TxFIFO 读取一个包后此区域的值将减少一次。	2'b0
Reserved	[18:7]		保留	12'h0
XferSize	[6:0]	R_W	转换尺寸 指明端点 o 的转换尺寸的字节。只有在转换尺寸数量的数据用完以后，核心将中断应用。转换尺寸可以设置为端点最大的包尺寸，在包的结尾将中断转换尺寸。 当外部存储器内的包写入 TxFIFO 时，核心将减少此区域。	7'h0

6.设备输出端点 0 转换尺寸寄存器（DOEPTISIZ0）

实际应用中，在端点 0 使能以前必须修改此寄存器。一旦端点 0 用设备控制端点 0 控制寄存器的断点使能位使能，核心修改此寄存器。当核心已经清除端点使能位时，应用只能读取此寄存器。

寄存器	地址	读/写	描述	复位值
DOEPTISIZ0	0x7C00_0B10	读/写	设备输出端点 0 转换尺寸寄存器	0x00000000

DOEPTISIZ0	位	读/写	描述	初始状态
Reserved	[31]		保留	1'h0
SUPCnt	[30:29]	R_W	设置包计数 此区域指定了端点可以接受的 back-to-back 设置数据包的数量 2'b01: 1 个包	2'h0

			2'b10: 两个包 2'b11: 3 个包	
Reserved	[28: 20]		保留	9'h0
PktCnt	[19]	R_W	包计数 向 R 向 FIFO 写入一个包以后，此区域值减到 0.	1'b0
Reserved	[18:7]		保留	12'h0
XferSize	[6:0]	R_W	转换尺寸 指明端点 0 的转换尺寸的字节。只有在转换尺寸数量的数据用完以后，核心将中断应用。转换尺寸可以设置为端点最大的包尺寸，在包的结尾将中断转换尺寸。 当外部存储器内的包写入 TxFIFO 时，核心将减少此区域。	7'h0

### 7.设备端点-N 转换尺寸寄存器（DIEPTISIZn/DOEPTISIZn）

端点序号：1≤n≤15

实际应用中，在端点 0 使能以前必须修改此寄存器。一旦端点 0 用设备控制端点 0 控制寄存器的断点使能位使能，核心修改此寄存器。当核心已经清除端点使能位时，应用只能读取此寄存器。此寄存器值用于除了端点 0 意外的端点

寄存器	地址	读/写	描述	复位值
DIEPISIZn/DOEPTISIZn	0x7C00_0910+n*20h /0x7C00_0B10+n*20h	读/写	设备端点-n 转换尺寸寄存器	0x00000000

DIEPISIZn/DOEPTISIZn	位	读/写	描述	初始状态
Reserved	[31]		保留	1'b0

MC	[30:29]	R_W	<p>多重计数</p> <p>只用在输入端点</p> <p>对于周期输入端点而言，此区域指定了 USB 上微帧可以转换的包的数量。核心用此区域计算同步输入端点的数据 PID。</p> <p>2'b01: 1 个包</p> <p>2'b10: 两个包</p> <p>2'b11: 3 个包</p>	2'b0
RxDPID		RO	<p>对于非周期输入而言，此区域只有在内部 DMA 模式下有效。</p> <p>它指明了核心可以为输入端点取得的包的数量。</p>	
		RO	<p>接收的数据 PID</p> <p>应用在同步输出端点</p> <p>这是端点在最后的包中接受的数据 PID</p> <p>2'b00: DATA0</p> <p>2'b01: DATA1</p> <p>2'b10: DATA2</p> <p>2'b11: MDATA</p>	
SUPCnt		R_W	<p>设置包计数</p> <p>应用于控制输出端点。此区域定义了端点可以接收的 back-to-back 设置数据包的量</p> <p>2'b01: 1 个包</p> <p>2'b10: 两个包</p> <p>2'b11: 3 个包</p>	
PktCnt	[28:19]	R_W	<p>包计数</p> <p>指明 USB 包的总的数量，USB 包的数量构成端口点 0 数据的换尺寸数量。</p> <p>对于输入端点而言：每从 TxFIFO 读取一个包后此区域的值将减少一次。</p> <p>对于输出端点而言：每向 TxFIFO 写入一个包后此区域的值将减少一次。</p>	1'b0

XferSize	[18:0]	R_W	<p>转换尺寸</p> <p>此区域包括当前端点的转换尺寸的字节。</p> <p>只有在转换尺寸数量的数据用完以后，核心将中断应用。转换尺寸可以设置为端点最大的包尺寸，在包的结尾将中断转换尺寸。</p> <p>对于输入端点而言：当外部存储器内的包写入 TxFIFO 时，核心将减少此区域。</p> <p>对于输出端点而言：每次从 RxFIFO 内读取包和向内部存储器写入包时，核心减少此区域。</p>	7'h0
----------	--------	-----	--	------

**8.设备端点-N DMA 地址（DIEPDMA<sub>n</sub>/DOEPDMA<sub>n</sub>）**

端点序号：1≤n≤15

DMA 的开始地址必须为 DWORD.

寄存器	地址	读/写	描述	复位值
DIEPDMA <sub>n</sub> /DOEPDMA <sub>n</sub>	0x7C00_0914+n*20h /0x7C00_0B14+n*20h	读/写	设备端点-n DMA 寄存器	0x00000000

DIEPDMA <sub>n</sub> /DOEPDMA <sub>n</sub>	位	读/写	描述	初始状态
DMAAddr	[31:0]	R_W	<p>DMA 地址</p> <p>控制外部存储器用于储存和获取端点数据的开始地址。这个寄存器每进行 AHB 处理后将会增加。</p> <p>注：对控制端点而言，这个地址储存了控制输出数据包和设置处置数据包。如果多重设置包被接受，存储期内的设置数据包将会覆写。</p>	32'b0

26.5.3.3. 电源和时钟门控制寄存器

电源和时钟门控制寄存器（PCGCCTL）

应用中可以用此寄存器控制 OTG 时钟门

寄存器	地址	读/写	描述	复位值
PCGCCTL	0x7C00_0E00	读/写	电源和时钟门控制寄存器	0x00000000

PCGCCTL	位	读/写	描述	初始状态
Reserved	[31:1]		保留	31'h0
StopPclk	[0]	R_W	停止 Pclk  应用中，当 USB 暂停时，或 session 无效，或设备未连接时，设置此位来停止 PHY 时钟。当 USB 恢复或新的 session 开始后清除此位。	1'b0



## 27 高速 MMC 控制器

这节主要介绍 S3C6410 RISC 微处理器上支持的多媒体卡控制器和相关寄存器。

HSMMC(高速 MMC)和 SDMMC 是一个组合编码/解码器主机，主要用于 SD 卡和多媒体卡。该主机是兼容 SD 协会(SDA)主机标准规范的，可以在系统上带有 SD 卡和 MMC 卡的接口。具有这样功能的主机，其性能是非常强大的。能获得 50MHz 的时钟频率，同时访问 8 位数据引脚。

### 27.1 高速 MMC 控制器特性

该高速MMC控制器支持：

- (1) 兼容 SD 标准主机规格（版本 1.0）。
- (2) 兼容 SD 存储卡规格（版本 2.0）/HSMMC 规格（版本 4.0）。
- (3) 兼容 SDIO 卡规格（版本 1.0）。
- (4) 512 字节的 FIFO 数据传输。
- (5) 48 位的指令寄存器。
- (6) 136 位的响应寄存器。
- (7) CPU 接口和 DMA 数据传输模式。
- (8) 支持 1 位/4 位/8 位模式转换。
- (9) 支持自动 CMD12。
- (10) 支持暂停/恢复。
- (11) 支持读等待操作。
- (12) 支持 CE-ATA 模式。

高速 MMC 控制器模块的结构框图，如图 27-1 所示。

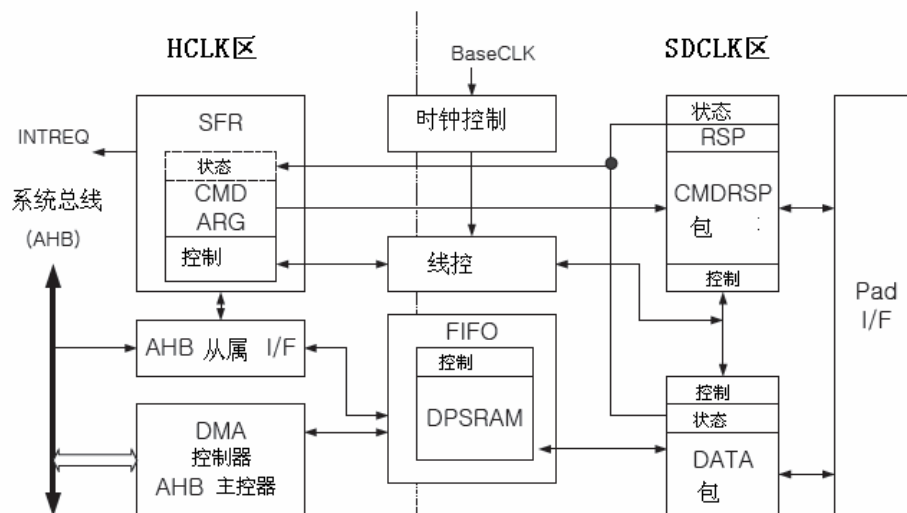


图 27-1 MMC 控制器结构框图

## 27.2 流程

下面将基本时序流程图分成几个小部分加以介绍。流程图需要“等待中断”，意思是主机驱动器等到指定的中断发生。如果发生中断，则进行流程图中的步骤；如果没有检测到中断产生，则不进行流程图的步骤。

### 1. SD卡检测流程

检测SD卡的流程图如图27-2所示。

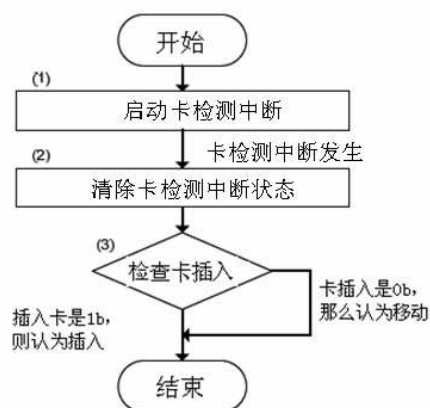


图 27-2 SD 卡检测流程

步骤如下：

(1) 卡能检测中断，将下面的位置1：

在正常的中断状态使能寄存器中，卡插入状态使能。

在正常的中断信号使能寄存器中，卡插入信号使能。

在正常的中断状态使能寄存器中，卡移除状态使能。

在正常的中断信号使能寄存器中，卡移除信号使能。

(2) 当主机驱动器检测卡插入还是移除时，它清除中断状态。如果卡插入状态产生，在正常中断状态寄存器中写‘1’则卡插入；如果卡移除中断产生，在正常中断状态寄存器中写‘1’则卡移除。

(3) 在当前状态寄存器检查卡插入。如果卡插入的情况为‘1’，那么主机驱动器提供能量和时钟到SD卡；如果卡插入的情况为‘0’，主机驱动器则立即关闭。

## 2. SD卡时钟供应流程

对于提供SD时钟到SD卡的流程，如图27-3 所示。

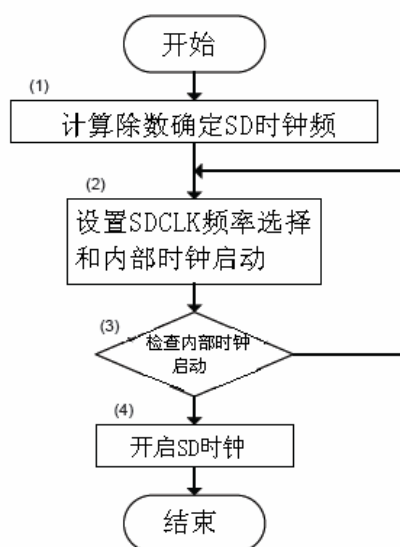


图 27-3 SD 时钟提供流程

过程如下：

- 发出一个SD指令。
- 从4位模式的SD卡中，检测到中断。

(1) 通过读在容限寄存器中的基础时钟频率，为SD卡时钟计算除数来确定SD卡时钟频率。如果SD时

钟的基础时钟频率是000000b，主机系统将通过其它方法为主机驱动器提供这些信息。

- (2) 在时钟控制寄存器中，设置内部时钟启动和SDCLK频率选择，使结果与（1）相符。
- (3) 在时钟控制寄存器中查看内部时钟状态，重复这些步骤直到时钟状态为‘1’。
- (4) 在时钟控制寄存器中设置SD时钟启动为‘1’，主机控制器开始提供SD卡时钟。

3. SD时钟停止流程

停止SD时钟流程图，如图27-4所示。

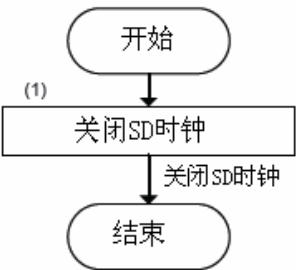


图 27-4 SD 时钟关闭流程

当SD总线处理进行，也就是，在当前状态寄存器中或者指令禁止（DAT）或者指令禁止（CMD）被置为1时，主机驱动器并不停止SD时钟。

在时钟控制寄存器中设置SD时钟启动为‘0’，然后，主机控制器停止提供给SD时钟。

4. SD时钟频率改变流程

改变SD时钟频率的流程，如图27-5所示。

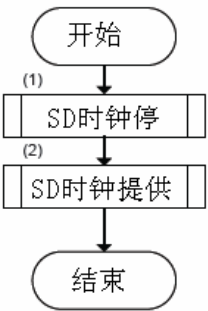


图 27-5 SD 时钟改变流程

当SD时钟静止时，步骤省略。

## 5. SD 总线能量控制流程

控制SD总线电源的流程，如图27-6所示。

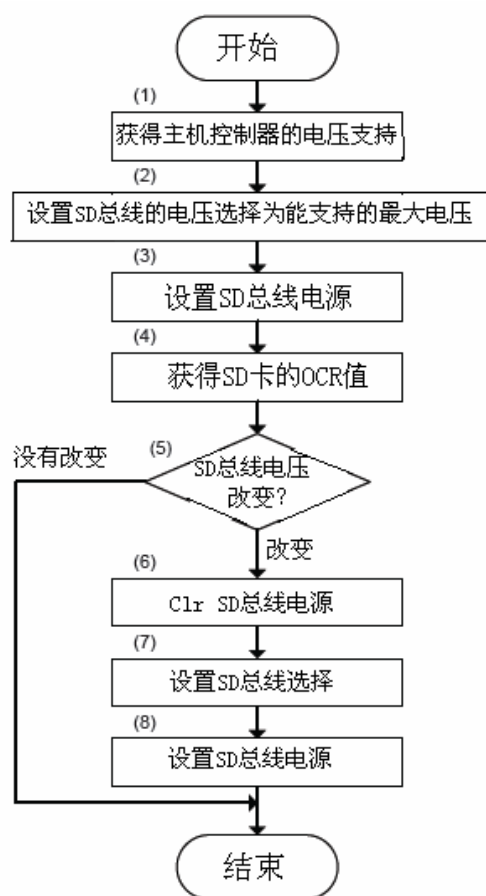


图 27-6 SD 总线控制流程

步骤如下：

- (1) 读容限寄存器，获得主机控制器的电压支持。
- (2) 在能量控制寄存器中设置SD总线电压选择主机控制器支持的最大电压。
- (3) 在能量控制寄存器中设置SD总线电源为1。
- (4) 获得SD卡内部功能的OCR值。
- (5) 判断是否必须改变SD总线电压。如果必须改变SD总线电压，则到步骤（6）；否则到‘结束’。

(6) 在能量控制寄存器中设置SD总线的能量为0来清除这位。卡要求电压从0上升到正确值，主机驱动器将在被通过SD总线设置选择改变电压前清除SD总线能量。

(7) 在能量控制寄存器中设置SD总线电压选择。

(8) 在能量控制寄存器中设置SD总线能量为1。

注：步骤（2）和步骤（3）能同时进行。同样，步骤（7）和步骤（8）也能同时进行。

## 6. 改变总线宽度流程

SD卡总线中改变位模式流程，如图27-7所示。

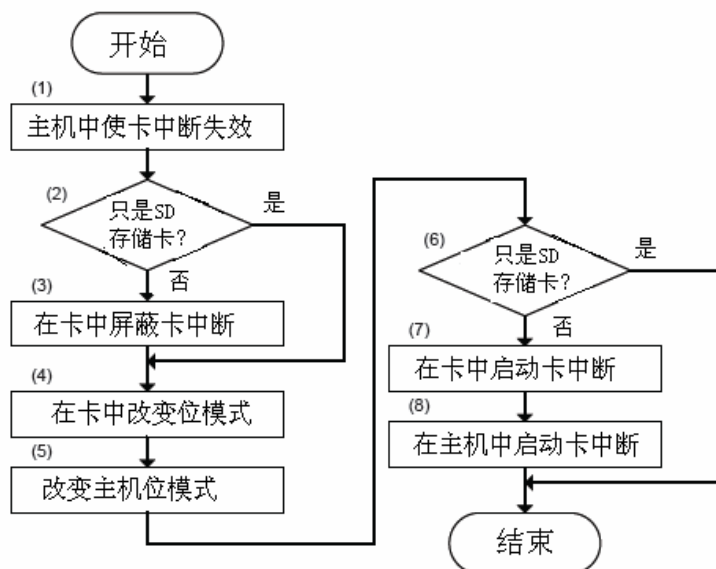


图 27-7 改变总线宽度的流程

步骤如下：

(1) 在正常中断状态启动寄存器中设置卡中断状态启动为0，用于屏蔽在改变总线宽度时可能发生的不正常的中断。

(2) 如果是SD存储卡，那么到步骤（4）；如果是其它的卡则进行步骤（3）。

(3) 在SDIO或者SD组合解码/编码卡中通过CMD52设置CCCR的“IENM”为0。

(4) 改变位模式为SD卡。通过ACMD6改变SD存储卡的总线宽度和通过设置总线接口控制器的总线宽度来改变SDIO卡总线宽度。

(5) 如果想改成4位模式，在主机控制寄存器中设置数据传输宽度为1。其它情况（1位模式），设置为0。

(6) 如果是SD存储卡，那么到“结束”；如果是其它卡，到步骤（7）。

(7) 在SDIO卡或者SD解码/编码卡中，通过CMD52设置CCCR的“IENM”为1。

(8) 在正常中断状态启动寄存器中设置卡中断状态启动为1。

## 7. 为数据线设置超时

为了对数据检测超时错误，在SD卡的处理之前，如图27-8所示。

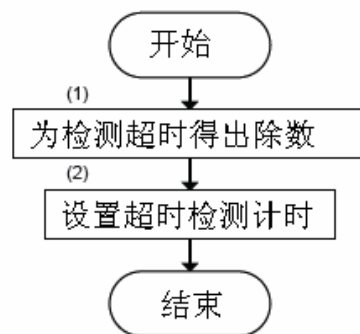


图 27-8 超时设置流程

主机驱动器将执行下面的步骤：

(1) 读容限寄存器中的超时时钟频率和超时时钟单元，计算除数以检测超时错误。如果时钟频率是000000b, 主机系统将通过其它方式向主机驱动器提供信息。

(2) 在时间控制寄存器中设置数据超时计数器的值，使其和步骤（1）中的值相符。

## 8. SD处理产生

下面介绍产生的流程和SD处理控制的种类。SD处理种类分为以下三中情况：

(1) 不使用线的处理。

(2) 对于繁忙信号使用数据线的处理。

(3) 使用数据线用于传输数据的处理。

在上述情况中第一种和第二种情况被分类为“数据传输没有使用数据线的传输控制”，第三种情况被分类为“数据传输使用数据线的传输控制”。

## 9. SD指令发出流程

如图27-9所示，显示了指令发出的流程图。

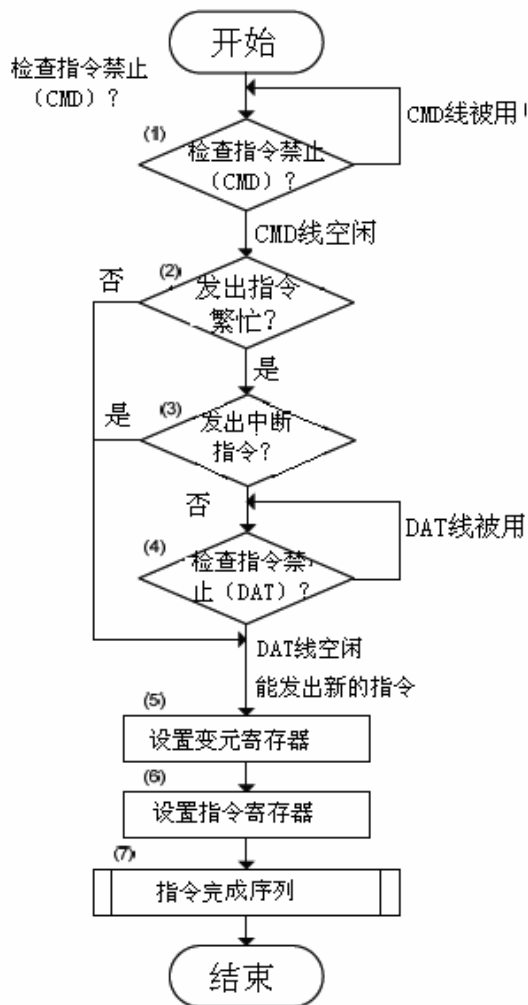


图 27-9 指令发出流程

其步骤如下：

- (1) 重复查看当前状态寄存器中指令禁止 (CMD)，直到指令禁止 (CMD) 是0。也就是，指令禁止 (CMD) 是1时，主机驱动器不会发出SD指令。
- (2) 如果主机驱动器发出SD指令伴有繁忙信号，则进行步骤(3)。如果没有繁忙信号则进行步骤(5)。
- (3) 如果主机驱动器是一个中断指令，进行步骤(5)。如果没有中断信号，进行步骤(4)。
- (4) 检查当前状态寄存器指令禁止 (DAT)。重复操作，直到指令禁止 (DAT) 是0。



(5) 在变源寄存器设置相应的值来发出指令。

(6) 在指令寄存器设置相应的值来发出指令。

注：在指令寄存器中写上面的字节来是SD指令发出。

(7) 执行指令完成顺序。

## 10. 指令完成流程

完成SD卡指令，如图27-10所示。

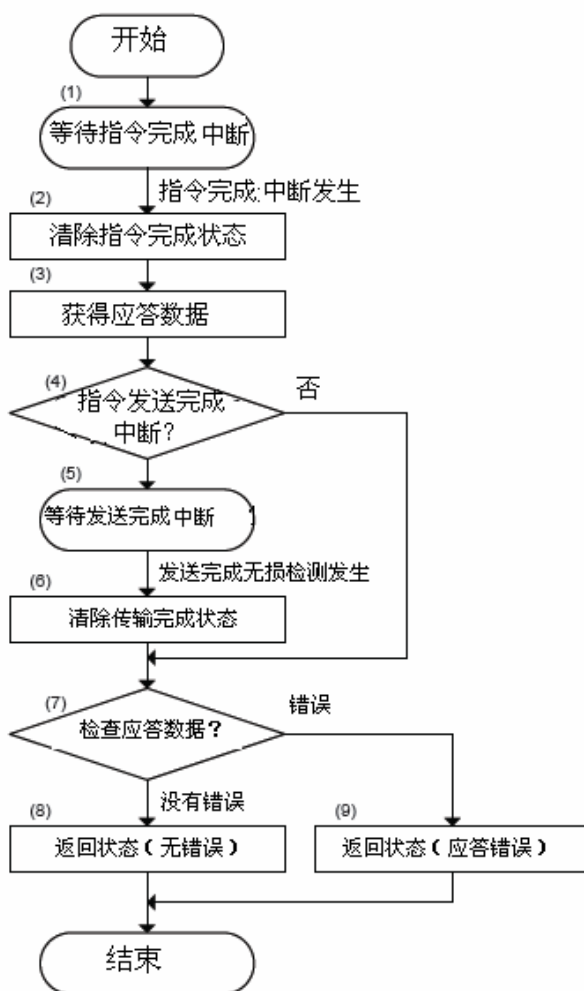


图 27-10 指令完成流程

流程中可能会有错误（指令索引/结束位/超时错误）发生。

- (1) 等待指令完成中断。如果完成中断发生，执行步骤(2)。
- (2) 在正常中断状态寄存器写1到指令完成来清除这位。
- (3) 读应答寄存器，并且根据发出的指令得到信息。
- (4) 判断指令是否使用传输完成中断。如果是，则继续执行步骤(5)；否则执行步骤(7)。
- (5) 等待传输完成中断，如果传输完成中断发生，执行步骤(6)。
- (6) 在正常中断状态寄存器中写1到传输完成来清除位。
- (7) 检查应答数据错误。如果没有错误，执行步骤(8)；否则，执行步骤(9)。
- (8) 返回“无错误”状态。
- (9) 返回“应答内容错误”状态。

注：

等待传输完成中断，主机驱动器将只发出没有使用繁忙信号的指令。

主机驱动器通过监控传输完成来判断自动 CMD12 完成。

当使用存储器多时钟模块读指令(CMD18)读最后一块无保护区域，即使流程正确的情况下，超出范围的错误也可能发生。主机驱动器必须屏蔽它。这种错误可能发生在自动 CMD12 应答或者下一条指令的应答。

## 11. 用DAT线传输数据的处理控制

是否使用DMA（可选），有两种执行方法。不使用DMA的流程如图27-11所示，使用DMA的流程如图27-12所示。

另外，根据指定区段的数量将SD传输流程分类。分成的三类如下：

### (1) 单区段传输

在传输前，向主机控制器指定传输的区段数量。这类中指定的传输数量总是1。

### (2) 多区段传输

在传输前，向主机控制器指定传输的数量。指定的传输数量是一个或者多个。

### (3) 无限制区段传输

在传输前，不向主机控制器指定区段传输的数量。在异常中断处理执行前，传输一直进行。在SD存储器卡中通过CMD12或者在SDIO卡中通过CMD52，处理这个异常中断。

不使用 DMA 控制流程

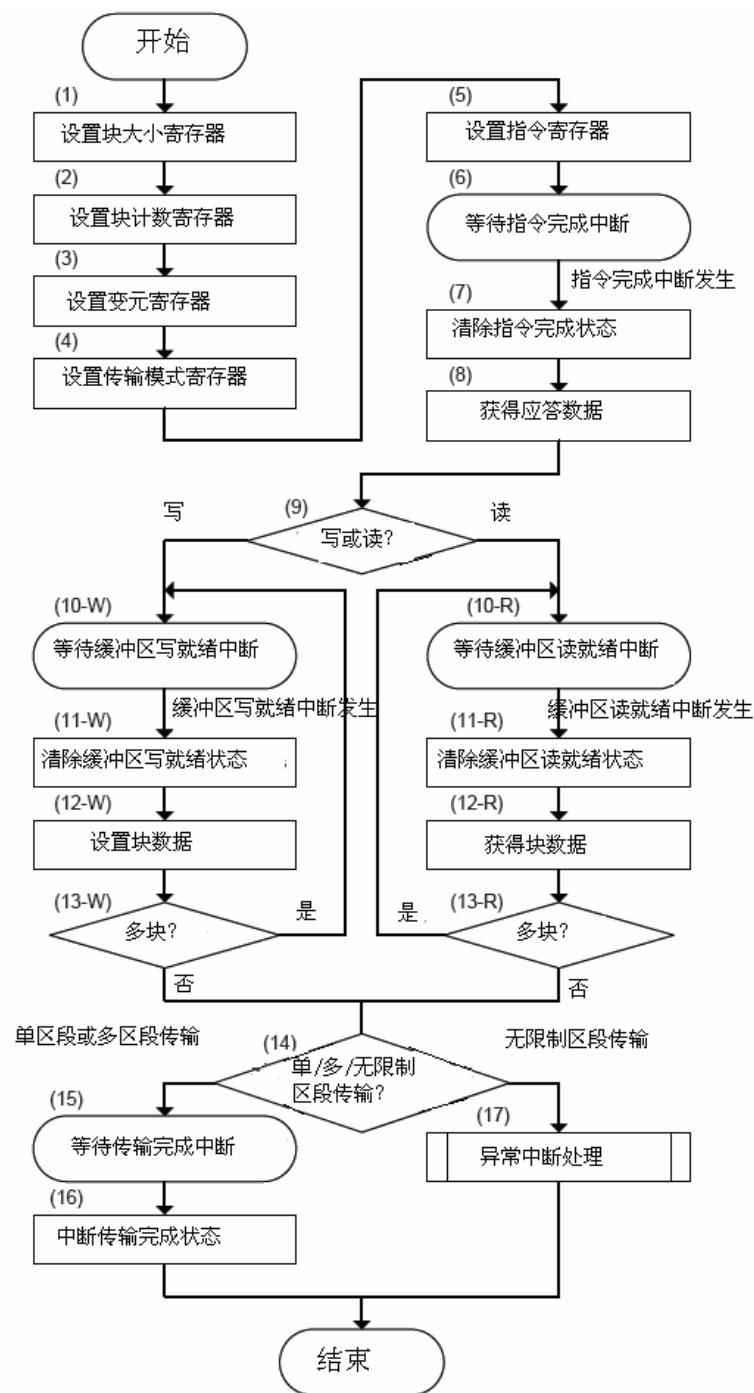


图 27-11 用 DAT 线进行数据传输的处理控制流程(不使用 DMA)

(1) 设置符合一个区段的执行数据字节长度值到块大小寄存器。

(2) 设置符合执行的数据区段计数的值到块计数寄存器。

(3) 设置符合发出指令的值到变元寄存器。

(4) 设置值到多个/单个区段选择和区段计数启动。同时，设置符合发出指令的值到数据传输流向，启动自动CMD12和DMA。

(5) 设置符合发出指令的值到指令寄存器。注：当写指令寄存器高位的字节，SD指令发出。

(6) 等待指令完成中断。

(7) 在正常状态寄存器将指令完成置1以清除这位。

(8) 读应答寄存器和获得与发出指令一致的必要信息。

(9) 如果是向卡中写入则执行步骤(10-W)；如果是从卡中读取则执行步骤(10-R)。

(10-W) 等待缓冲区的写就绪中断。

(11-W) 在正常中断状态寄存器中将缓冲区写就绪置1以清除这位。

(12-W) 写区段数据(根据步骤(1)指定的字节数)到缓冲区数据端口寄存器。

(13-W) 重复操作直到所有区段发送完毕，执行步骤(14)。

(10-R) 等待缓冲区读就绪中断。

(11-R) 在正常中断状态寄存器中将缓冲区读就绪置1以清除这位。

(12-R) 从缓冲区数据端口寄存器读数据(根据步骤(1)指定的字节数)。

(13-R) 重复操作直到所有的区段被接收，进入步骤(14)。

(14) 如果是单区段或者多区段传输，进入步骤(15)，如果是无限区段传输，进入步骤(17)。

(15) 等待发送完成中断。

(16) 在正常中断状态寄存器中将发送完成置1，以清除这位。

(17) 为异常中断处理执行这个流程。

注：步骤(1)和步骤(2)能同时执行，步骤(4)和步骤(5)能同时执行。

使用 DMA 控制流程

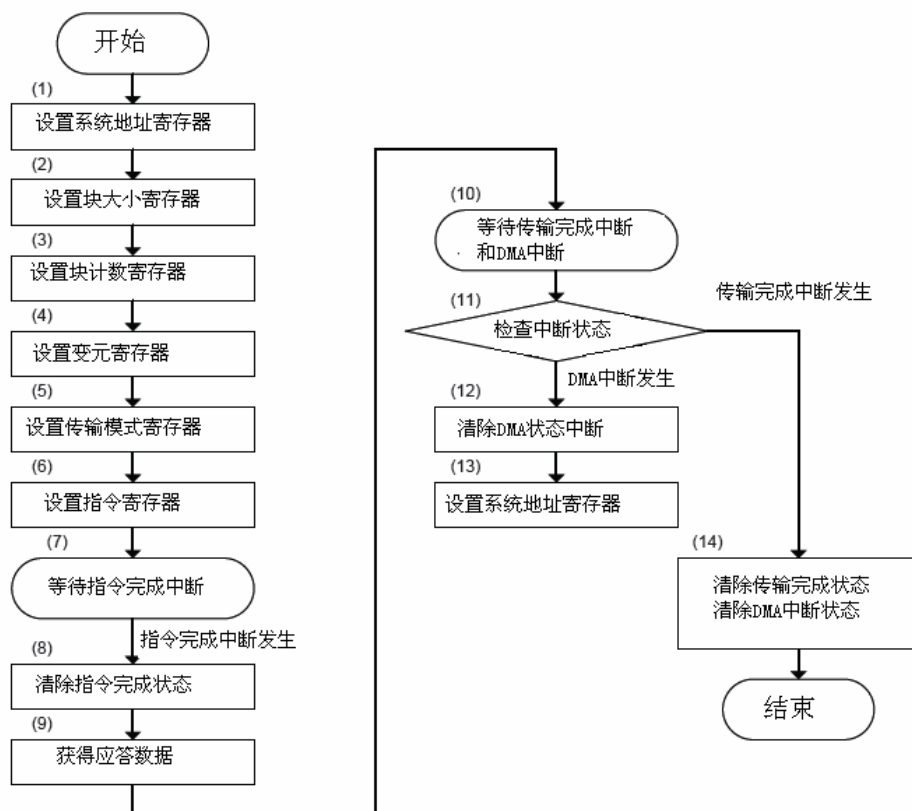


图 27-12 用 DAT 线进行数据传输的处理控制流程(使用 DMA)

(1) 在系统地址寄存器中为DMA设置系统地址。

(2) 在块大小寄存器中设置符合一个区段被执行数据的字节长度的值。

(3) 在块计数寄存器中设置符合被执行数据区段计数的值。

(4) 在变元寄存器中设置符合发出指令的值。

(5) 为单/多区段选择和区段计数启动设置值。同时，用于数据传输流向设置符合发出指令的值，启动自动CMD12和DMA。

(6) 在指令寄存器中设置符合发出指令的值。注：向指令寄存器高位字节写入时，SD指令被发送和DMA开始。

(7) 等待指令完成中断。

(8) 在正常中断状态寄存器中将指令完成置1来清除这位。

(9) 读应答寄存器，获得与发出指令一致的必要信息。

(10) 等待传输完成中断和DMA中断。

(11) 如果传输完成被置1，那么进入步骤(14)；如果DMA中断被置1，进入步骤(12)。较DMA中断，传输完成中断优先。

(12) 在正常中断状态寄存器将DMA中断置1，来清除这位。

(13) 设置下一个数据位置的系统地址，到系统地址寄存器，并且进入步骤(10)。

(14) 在正常中断状态寄存器中将传输完成置1，来清除这位。

注：步骤(2)和步骤(3)能同时执行。步骤(5)和步骤(6)能同时执行。

## 12. 异常中断处理

对于SD存储卡通过发出CMD12，对于SDIO卡通过发出CMD52来进行异常处理。有两种情况需要，主机驱动器需要做中断处理。第一种情况是主机驱动器停止无限驱动器传输；第二种情况是主机驱动器停止传输时多区段传输执行。

有两种方式来发送一个异常处理命令，第一种是异步中断，第二种是同步中断。对于异步中断，主机驱动器能发出异常中断命令，除非在当前状态寄存器中将命令禁止(CMD)设置为‘1’。对于同步异常中断，在块间隔控制寄存器通过使用停止块间隔请求停止数据传输后，主机驱动器将发出一个异常中断命令。

## 27.3 DMA 处理

DMA允许外设在没有CPU干涉的情况下来读和写存储器。只有一个SD指令处理能通过DMA执行。支持DMA的主机驱动器，支持单区段和多区段传输。

系统地址寄存器指向第一个数据地址，然后从这个地址连续地访问数据。主机控制寄存器将为在DMA传输期间发送non-DAT线指令保持可存取。不管系统总线使用的传输方式是哪一种，DMA传输结果是一样的。DMA不支持无限传输。

通过在块间隔控制寄存器中使用控制位，能启动和停止DMA传输。当停止块间隔被设置，DMA传输将暂停；当继续请求被设置或者发出一个恢复指令，DMA将继续执行传输。如果SD总线错误发生，SD总线传输和DMA传输将被停止。在软件复位寄存器为DAT线设置软件复位中断DMA传输。

## 27.4 SDI 特殊寄存器

本小节主要介绍一下 SDI 的特殊寄存器

### 27.4.1. 配置寄存器类型

配置寄存器域指定下面描述的属性。

寄存器属性	描述
RO	只读寄存器：寄存器位是只读的，并且不能通过软件或者其它设置操作来进行修改。向该位写入将被忽略
ROC	只读状态：这个位被初始化为 0，向该位写入将被忽略
RW 或者 R/W	可读写寄存器：寄存器位可读写，并且能进行其它任何设置或者通过软件清除到所要的状态
RW1C	只读状态，写 1 到清除状态：当读时寄存器位指出状态。一个设置位指出一个状态，通过写 1 可以清除。写 0 到 RW1C 位没有影响
RWAC	读写自动清除寄存器：通过设置这位主机驱动器要求一个主机控制器操作。当操作完成，主机驱动器将自动清除该位。写 0 到 RWAC 位没有影响
HWInit	硬件初始化：通过固件或者硬件装置初始化寄存器位。初始化后读取位，并且写入该位被忽略
Rsvd 或 Reserved	保留。这些位初始化为 0，写入这些位被忽略

### 27.4.2. 系统地址寄存器

该寄存器包含用于 DMA 传输的物理存储地址。

寄存器	地址	读/写	描述	复位值
SYSAD0	0x7C200000	读/写	系统地址寄存器(0通道)	0x0
SYSAD1	0x7C300000	读/写	系统地址寄存器(1通道)	0x0
SYSAD2	0x7C400000	读/写	系统地址寄存器(2通道)	0x0

名称	位	描述	初始状态
SYSAD	[30:0]	DMA 系统地址 这个寄存器包含系统存储地址用于DMA传输。当主机控制器停止	0x00

		<p>DMA传输，寄存器将指向下一个邻近数据位置的系统地址。如果没有处理进行(也就是，处理停止后)，那么它能被访问。在传输过程中，读操作可能返回一个无效值。</p> <p>在开始DMA传输前，主机控制器将初始化这个寄存器。DMA停止后，能从这个寄存器中读取邻近的数据位置的系统地址。</p> <p>DMA传输等待在每个在块大小寄存器中被主机DMA缓冲区边界指定的边界。主机控制寄存器产生DMA中断来请求主机更新寄存器。主机驱动器设置下一个数据位置的系统地址到这个寄存器。</p> <p>当这个寄存器最高的字节(003h)被写入，主机控制器重启DMA传输。当重新启动DMA通过重新启动指令或者通过在块间隔控制寄存器设置继续请求，主机控制器将在被存储在系统地址寄存器中的下一个邻近的地址开始。</p>	
--	--	--	--

### 27.4.3. 块大小寄存器

该寄存器用来设置一个数据块的字节数。

寄存器	地址	读/写	描述	复位值
BLKSIZE0	0x7C200004	读/写	主机DMA缓冲区边界和传输块大小寄存器(0通道)	0x0
BLKSIZE1	0x7C300004	读/写	主机DMA缓冲区边界和传输块大小寄存器(1通道)	0x0
BLKSIZE2	0x7C400004	读/写	主机DMA缓冲区边界和传输块大小寄存器(2通道)	0x0

名称	位	描述	初始状态
	[15]	保留。	0
	[14:12]	<p>主机DMA缓冲区边界。</p> <p>在虚拟内存系统大的邻近的存储空间不可用。为了进行更长的DMA传输，系统地址寄存器将在DMA传输期间对每个系统存储边界更新。这些位指定在系统内存中邻近的缓冲区的大小。DMA传输将等待被这些区域指定的每一个边界，并且主机控制器产生DMA中断来要求主机驱动器来更新系统地址寄存器。</p> <p>在这些寄存器设置为0的情况下(缓冲区大小为4K字节)，字节地址的低12位指向邻近的缓冲区的数据，高20位的指向系统存储器中缓冲区。当主机控制器检查完地址位11到12位，DMA传输停止。</p> <p>当在容限寄存器里将DMA支持设置为1时，这些位将被支持，并且当在传输模式寄存器里将DMA有效设置为1时，这项功能被启动。</p> <p>000b = 4KB(检测 A11 执行)</p> <p>001b = 8 KB (检测 A12 执行)</p> <p>010b = 16 KB (检测 A13 执行)</p>	0



		011b = 32 KB (检测 A14 执行) 100b = 64 KB (检测 A15 执行) 101b = 128 KB (检测 A16 执行) 110b = 256 KB (检测 A17 执行) 111b = 512 KB (检测 A18 执行)	
	[11:0]	传输块大小。 这个寄存器为CMD17、CMD18、CMD24、CMD25和 CMD53指定数据传输的传输区段大小。值的范围能设置为1到缓冲区的最大值。存储器中，它设置为512个字节。没有处理在执行，它只能被访问（也就是处理关闭后）。在传输期间读操作将返回无效值，写操作将被忽略。 0200h = 512字节 01FFh = 511字节 ... .. 0004h = 4字节 0003h = 3字节 0002h = 2字节 0001h = 1字节 0000h = 无数据传输	0

#### 27.4.4. 块计数寄存器

该寄存器用来设置数据块的数量。

寄存器	地址	读/写	描述	复位值
BLKCNT0	0x7C200006	读/写	为当前传输的块计数( 0通道)	0x0
BLKCNT1	0x7C300006	读/写	为当前传输的块计数( 1通道)	0x0
BLKCNT2	0x7C400006	读/写	为当前传输的块计数( 2通道)	0x0

名称	位	描述	初始状态
	[15:0]	为当前传输进行块计数。 当在传输模式寄存器中将块计数启动设置为1，并且是多区段传输时，这个寄存器启动。主机驱动器将设置为这个寄存器设置一个1到最大区段计数中的一个值。每一块传输后，主机控制器减少这个块计数，当计数减少到0时停止。设置块计数为0表示没有数据块正在传输。 这个寄存器必须在没有传输执行（也就是，传输停止）时访问这个寄存器。在数据传输期间，在这个寄存器读操作将返回一个有效值，忽略写操作。当将传输内容作为一个延缓指令的结果，通过读取这个寄存器的传输的块的数量。当发出一个	0

	<p>指令恢复先前的传输内容，主机驱动器将恢复先前保存的块数。</p> <p>FFFFh = 65535 块</p> <p>... ..</p> <p>0002h = 2块</p> <p>0001h = 1 块</p> <p>0000h = 停止块计数</p>	
--	--	--

### 27.4.5. 变元寄存器

该寄存器包含SD指令变元。

寄存器	地址	读/写	描述	复位值
ARGUMENT0	0x7C200008	读/写	指令变元寄存器(0通道)	0x0
ARGUMENT1	0x7C300008	读/写	指令变元寄存器(1通道)	0x0
ARGUMENT2	0x7C400008	读/写	指令变元寄存器(2通道)	0x0

名称	位	描述	初始状态
ARG	[31:0]	指令变元。在SD存储卡物理层规范中，SD指令变元被指定作为指令格式的位39到8	0

### 27.4.6. 传输模式寄存器

该寄存器用于控制数据传输的操作。在发出用于传输数据的指令前，或者是在发出一个恢复指令前，主机控制驱动器将设置这个寄存器。当数据传输暂停和使用恢复指令恢复它之前，主机驱动器将保存这个寄存器的值。为了防止数据丢失，在数据传输期间，主机控制器将为寄存器执行写保护。当指令禁止(DAT)在当前状态寄存器中设置为1时，写入这个寄存器将被忽略。

寄存器	地址	读/写	描述	复位值
TRNMOD0	0x7C20000C	读/写	传输模式设置寄存器(0通道)	0x0
TRNMOD1	0x7C30000C	读/写	传输模式设置寄存器(1通道)	0x0
TRNMOD2	0x7C40000C	读/写	传输模式设置寄存器(2通道)	0x0

名称	位	描述	初始状态
----	---	----	------

	[15:10]	保留。	0
	[9:8]	指令完成信号控制： ‘00’ = No CCS 操作 (正常操作，非 CE-ATA 模式) ‘01’ = 读或者写数据传输CCS启动 (仅CE-ATA 模式) ‘10’ = 没有数据传输CCS启动 (仅CE-ATA 模式) ‘11’ = 异常中断完成信号 (ACS) 产生 (仅CE-ATA 模式)	0
	[7:6]	保留。	0
	[5]	多/单区段选择。 这位激活多区段DAT线性数据传输。对于其它指令，这位设置为0。 如果这位为0不需要设置块计数寄存器 (参考表6-80，确定传输类型)。 1 = 多区段 0 = 单区段	0
	[4]	数据传输流向选择。 这位定义了DAT线数据传输的流向。主机驱动器将这位设置为1，将数据从SD卡传输到SD主机控制器，对于其它指令，它被设置为0。 1 = 读 (从卡到主机) 0 = 写 (从主机到卡)	0
	[3]	保留。	0
	[2]	自动CMD12启动。 存储器多区段传输要求CMD12来停止传输。当这位设置为1，最后一个区段传输完成时，主机控制器自动发出CMD12。主机驱动器不设置这位发出不要求CMD12来停止数据传输的指令。 1 = 有效 0 = 无效	0
	[1]	块计数启动。 该位用来启动块计数寄存器，仅使用于多区段传输。当该位置为0，块计数寄存器无效，这用于执行无限制传输 (参考表6-80)。 1 = 有效 0 = 无效	0
	[0]	DMA 启动。 该位启动DMA 功能。当在容限寄存器中“DMA支持”支持DMA作为指示时，DMA启动。 如果不支持DMA，这位是没有意义的，置为0。如果这位设置为1，当主机驱动器写到指令寄存器的高位字节 (00Fh)，将开始DMA操作。 1 = 有效 0 = 无效	0

寄存器设置怎样确定数据的类型，如表27-1所示。

表 27-1 确定传输类型

多/单区段块选择	块计数启动	块计数	功能
0	-	-	单区段传输
1	0	-	无限区段传输
1	1	非0	多区段传输
1	1	0	停止多区段传输

注：对于CE-ATA 存取，指令完成信号失效后，（自动）CMD12必须发送。

27.4.7. 命令寄存器

这个寄存器包含SD指令变元。在写入这个寄存器前，主机驱动器将在当前状态寄存器检查指令禁止 (DAT)位和指令禁止 (CMD)位。写入这个寄存器的高位字节引发SD指令产生。主机驱动器将负责写入这个寄存器，因为当指令禁止 (CMD)被设置，主机控制器不提供写保护。

寄存器	地址	读/写	描述	复位值
CMDREG0	0x7C20000E	读/写	命令寄存器(通道0)。	0x0
CMDREG1	0x7C30000E	读/写	命令寄存器(通道1)。	0x0
CMDREG2	0x7C40000E	读/写	命令寄存器(通道2)。	0x0

名称	位	描述	初始状态
	[15:14]	保留。	
	[13:8]	指令索引。 这些位设置为SD存储卡物理层规范中指令格式的第40到45位和SDIO卡规范中指定的指令数 (CMD0-63, ACMD0-63)。	
	[7:6]	指令类型。 有三种特殊指令类型：暂停、恢复和中断。对于其它指令这位设置为00b。 • 暂停指令 如果暂停指令成功，主机控制器将假设SD总线被释放，它能够用DAT线发出下一个指令。主机控制器将不进行读等待处理，并且停止检测写处理的繁忙状况。在4位模式时，将开始中断周期。如果暂停指令失败，主机控制器将保持它的当前状态，主机驱动器将通过在块间隔控制器中设置“继续请求”恢复传输。	

		<ul style="list-style-type: none"> <li>• 恢复指令 主机驱动器通过恢复数据传输通过恢复寄存器在范围000~00Dh。主机控制器将在写传输前检测繁忙情况。</li> <li>• 异常中断指令 当执行读传输时，如果设置这个指令，主机控制器将停止读取缓冲区。当执行写传输的时候，如果设置这个指令，主机控制器将停止控制DAT线。发出异常中断指令后，主机驱动器必须发出一个软件复位。 11b =在CCCR 中，为写 “I/O 异常中断” ，中断CMD12, CMD52 10b = 在CCCR中，为写 “功能选择” ，恢复CMD52 01b = 在CCCR中，为写 “总线暂停”，暂停CMD52 00b = 正常其它指令</li> </ul>	
	[5]	<p>当前数据选择。</p> <p>该位设置为1，表示数据是当前的，将使用数据线传输。</p> <p>下面情况将设置为0:</p> <p>(1) 指令只用CMD 线（不包括CMD52）。</p> <p>(2) 指令没有数据传输，但在DAT[0]线上使用繁忙信号。</p> <p>(R1b或者R5b ，不包括CMD38)</p> <p>(3) 恢复指令</p> <p>1 = 当前数据</p> <p>0 = 无当前数据</p>	
	[4]	<p>指令索引检测启动。</p> <p>如果这位设置为1，主机控制器将在应答中检测索引域是否有相同的值作为指令索引。如果没有，它将作为一个索引错误被提出。如果这位设置为0，不检测索引域。</p> <p>1 = 有效</p> <p>0 = 无效</p>	
	[3]	<p>指令CRC检测启动。</p> <p>如果该位设置为1，主机控制器将检测应答中的CRC域。如果检测到错误，它将作为一个指令CRC错误被报告。如果该位设置为0，不检测CRC域。CRC域值检测到的位的数值依据应答的长度而改变。</p> <p>1 =有效</p> <p>0 = 无效</p>	
	[2]	保留。	
	[1: 0]	<p>应答类型选择:</p> <p>00 = 无应答</p> <p>01 = 应答长度 136</p> <p>10 = 应答长度48</p> <p>11 = 应答长度 48应答后检测繁忙情况</p>	

如表27-2所示，显示了参数名称和应答类型的关系。

表 27-2 参数名称和应答类型的关系

应答类型	索引检查启动	CRC检查启动	应答类型的名称
00	0	0	没有应答
01	0	1	R2
10	0	0	R3、R4
10	1	1	R1、R6、R5
11	1	1	R1B、R5b

注：在SDIO规范中，应答类型符号R5b没有定义。R5包括R5b在SDIO规范中。规范中定义R5b来指定接收应答后主机控制器检查繁忙情况。例如，通常CMD52被作为R5，但是I/O异常中断将被作为R5b。

注：写“总线暂停”后，CMD52来读BS，指令类型也必须为“暂停”。

27.4.8. 应答寄存器

该寄存器用来存储来自SD卡的应答。

寄存器	地址	读/写	描述	复位值
RSPREG0_0	0x7C200010	ROC	应答寄存器0(0通道)。	0x0
RSPREG1_0	0x7C200014	ROC	应答寄存器1(0通道)。	0x0
RSPREG2_0	0x7C200018	ROC	应答寄存器2(0通道)。	0x0
RSPREG3_0	0x7C20001C	ROC	应答寄存器3(0通道)。	0x0

寄存器	地址	读/写	描述	初始状态
RSPREG0_1	0x7C300010	ROC	应答寄存器0(1通道)。	0x0
RSPREG1_1	0x7C300014	ROC	应答寄存器1(1通道)。	0x0
RSPREG2_1	0x7C300018	ROC	应答寄存器2(1通道)。	0x0
RSPREG3_1	0x7C30001C	ROC	应答寄存器3(1通道)。	0x0

寄存器	地址	读/写	描述	初始状态
RSPREG0_2	0x7C400010	ROC	应答寄存器0(2通道)。	0x0

RSPREG1_2	0x7C400014	ROC	应答寄存器1(2通道)。	0x0
RSPREG2_2	0x7C400018	ROC	应答寄存器2(2通道)。	0x0
RSPREG3_2	0x7C40001C	ROC	应答寄存器3(2通道)。	0x0

名称	位	描述	初始状态
	[127:0]	指令应答。下表27-4为每一个应答描述了从SD总线到寄存器的指令映射。在这个表中，在表中 R[] 指出在SD总线上传输的应答数据的范围，REP[] 指出应答寄存器中位的范围。 128位应答位的顺序： {RSPREG3, RSPREG2, RSPREG1, RSPREG0}	

对于每一种应答类型，应答位的定义如表27-3所示。

表 27-3 每一种应答类型，应答位的定义

应答种类	应答含义	应答区域	应答寄存器
R1, R1b（正常应答）	卡状态	R [39:8]	REP [31:0]
R1b（自动 CMD12 应答）	对于自动CMD12卡状态	R [39:8]	REP [127:96]
R2（CID, CSD 寄存器）	CID或CSD reg. incl.	R [127:8]	REP [119:0]
R3（OCR 寄存器）	OCR寄存器用于存储	R [39:8]	REP [31:0]
R4（OCR寄存器）	OCR寄存器用于I/O等	R [39:8]	REP [31:0]
R5, R5b	SDIO应答	R [39:8]	REP [31:0]
R6（发表的 RCA 应答）	新发表的RCA[31: 16]等	R [39:8]	REP [31:0]

应答域指出了PHYSICAL LAYER SPECIFICATION1.01版本中定义的“应答”的位置。上表27-33显示大部分48(R[47:0])位的长度应答数据有32位(R[39:8])存储在应答寄存器的REP[31:0]。应答类型R1b（自动CMD12 应答）应答数据R[39:8]存储在应答寄存器的REP[127:96]。长136（R[135:0]）的应答的120位应答数据(R[127:8])存储在应答寄存器的REP[119:0]。

为了有效地读应答状态，主机控制器只在应答寄存器中存储部分的应答数据。这能使主机驱动器在32位总线系统的读周期有效地读取32位应答数据。对于部分应答，索引域和CRC被主机控制器（由指令寄存器的指令索引检测启动和指令CRC检测启动位指定）检测，并且如果检测到错误则产生一个错误中断。CRC检测的范围由应答的长度决定。如果应答长度是48，主机控制器将检测R[47:1]，如果应答长度是136，主机控制器将检测R[119:1]。

由于主机控制器通过CMD\_wo\_DAT指令可以让多区段数据DAT线同步传输执行，在应答寄存器的高位(Rep[127:96])，主机控制器存储自动CMD12应答。CMD\_wo\_DAT应答存储在Rep[31:0]。这使主机控制器避免自动CMD12应答写入过多，反之亦然。

当主机控制器修改应答寄存器中的部分位时，像表27-4所示的那样，它将保护未修改的位。

27.4.9. 缓冲区数据端口寄存器

32位数据端口寄存器来访问内部缓冲区。

寄存器	地址	读/写	描述	复位值
BDATA0	0x7C200020	读/写	缓冲区数据 寄存器(0通道)。	0x0
BDATA1	0x7C300020	读/写	缓冲区数据 寄存器(1通道)。	0x0
BDATA2	0x7C400020	读/写	缓冲区数据 寄存器(2通道)。	0x0

名称	位	描述	初始状态
		缓冲区数据。 通过32位数据端口寄存器能访问主机控制器缓冲区。	0

27.4.10. 当前状态寄存器

该寄存器包含SD指令变元。

寄存器	地址	读/写	描述	复位值
PRNSTS0	0x7C200024	RO/ROC	当前状态寄存器(通道0)。	0x000A0000
PRNSTS1	0x7C300024	RO/ROC	当前状态寄存器(通道1)。	0x000A0000
PRNSTS2	0x7C400024	RO/ROC	当前状态寄存器(通道2)。	0x000A0000

名称	位	描述	初始状态
	[31:25]	保留。	0
	[24]	CMD线信号电平(RO)。 这个状态用于检测DMD线电平是否从错误中恢复，并且调试。 注：CMD端口被映射到SD0_CMD引脚。	0



	[23:20]	<p>DAT[3:0] 线信号电平 (R0)。</p> <p>这个状态用于检测DAT线电平是否错误中恢复，并调试。对于从DAT[0]检测很有用。</p> <p>D23 : DAT[3] D22 : DAT[2] D21 : DAT[1] D20 : DAT[0]</p> <p>注： DAT 端口被映射到SD0_DAT pin</p>	线状态
	[19]	<p>写保护开关引脚电平 (R0)。</p> <p>这个写保护开关由存储器和组合编码/解码卡支撑。该位反应SDWP#引脚。</p> <p>1 = 写激活 (SDWP#=1) 0 = 写保护 (SDWP#=0)</p> <p>注：SDWP# 被映射到SD0_nWP 引脚， S3C6410 中，SD#_nWP 端口被固定到高位。</p>	1
	[18]	<p>卡检测引脚电平 (R0)。</p> <p>该位反映SDCD#引脚的倒置值，不执行反跳。当卡状态稳定设置为1时，该位可能有效，但是由于传导延迟并不能保证是这样。由于它必须被软件反跳，使用这位限制测试。</p> <p>1 = 当前有卡 (SDCD#=0) 0 = 当前没有卡 (SDCD#=1)</p> <p>注：SDCD# 端口被映射到SD0_nCD 引脚， S3C6410 中，SD2_nCD (2通道) 端口设置到低位。</p>	线状态
	[17]	<p>卡状态稳定 (R0)。</p> <p>该位用于测试。如果它是0，卡检测引脚电平不稳定；如果该位设置为1，意味着卡测试引脚电平稳定。无卡的检测状态能通过该位，设置为1，卡插入设置为0。在软件复位寄存器中“所有软件复位”不影响该位。</p> <p>1 = 无卡或插入 0 = 复位或反跳</p>	1 (复位后)
	[16]	<p>卡插入 (R0)。</p> <p>该位指示是否卡已经被插入。主机控制器将反跳这个信号一致于主机驱动器不需要来等它稳定。在正常中断状态寄存器，从0变为1产生一个卡插入中断，从1变为0产生一个卡移除中断。软件复位寄存器中“所有软件复位”不影响这该。如果移走卡，当它正在上电和时钟正在震荡，主机控制器将在电源控制寄存器中清除SD总线电源，并且在时钟控制寄存器SD时钟启动。</p> <p>当该位从1变到0，主机控制器将立即停止控制CMD和DAT[3:0] (三态)。而且，主机驱动器必须通过软件复位寄存器的“所有软件复位”清除主机控制器。卡检测是有效的。</p> <p>1 = 卡插入 0 = 复位或者反跳或者没有卡</p>	0

	[15:14]	保留。	0
DIFF4W	[13]	<p>FIFO 指示器差分4字 (ROC)。</p> <p>当地址指示器AHB和SD相差大于或者等于4字，这个状态位设置为高位。其它的自动清除。</p> <p>写(发送) 模式 :当这位是高位，大于或者等于4字能被CPU写。</p> <p>读(接收) 模式 : 当这位是高位，大于或者等于4字能被CPU读。</p>	0
DIFF1W	[12]	<p>FIFO 指示器差分 1字 (ROC)。</p> <p>当地址指示器在AHB和SD的差分大于或者等于1字，这个状态位被设置为高位。其它的自动清除。</p> <p>写(发送) 模式 :当这位是高位，大于或者等于1字能被CPU写。</p> <p>读 (接收) 模式: 当这位是高位，大于或者等于1字能被CPU读。</p>	0
	[11]	<p>缓冲区读有效 (ROC)。</p> <p>这个状态用于非DMA读传输。为了有效传输数据，主机控制器使用多个缓冲区。这个只读标志表示在主机方面缓冲区状态有效数据存在。如果这位是1，在缓冲区能读的数据存在。当所有的区段的数据被从缓冲区读取，这位从1变为0。当缓冲区中区段数据就绪，这位从0变为1，并且产生缓冲区读就绪中断。</p> <p>1 = 读有效</p> <p>0 = 读无效</p>	0
	[10]	<p>缓冲区写有效 (ROC)。</p> <p>这个状态用于非DMA写传输。为了使传输数据有效主机控制器能使用多个缓冲区。只读标志表示是否空间对于写数据可用。如果这位是1，数据能写入缓冲区。当所有的区段数据写入缓冲区这位从1变为0。当区段数据的顶部能写入缓冲区，这位从0变为1，并且产生缓冲区写就绪中断。</p> <p>1 = 写有效</p> <p>0 =写无效</p>	0
	[9]	<p>读传输有效 (ROC)。</p> <p>这个状态用于读取传输的测试完成。</p> <p>对于下面的条件这个位设置为1:</p> <p>(1) 读命令位结束后。</p> <p>(2) 写1到块间隔寄存器“继续请求”来重新启动读传输。</p> <p>对于下面的情况，这位清除到0:</p> <p>(1) 当最后的块长度指定的数据块传输到系统。</p> <p>(2) 所有有效数据块传输到系统，并且发送的非当前块传输被作为“停止在块间隔请求” 被设置为1的结果。这位置0时，传输完成中断产生。</p> <p>1 = 传输数据</p> <p>0 = 没有有效数据</p>	0
	[8]	<p>写传输有效 (ROC)。</p> <p>这个状态表示写传输有效。如果这位为0，意味着主机控制器中没</p>	0

		<p>有有效的写数据存在。</p> <p>下面的情况下这个位被设置：</p> <p>(1) 写指令的最后位后。</p> <p>(2) 当将块间隔控制器“继续请求”置1来重新启动写传输。</p> <p>下面的情况下清除这位：</p> <p>(1) 获得最后区段的CRC状态被传输计数（单和多）。</p> <p>(2) 在获得传输将要被“停止在块间隔请求”停止任何区段的CRC状态后。</p> <p>在写传输期间，当这位变为0时，“块间隔事件”中断产生。这个状态对于主机驱动器确定在写繁忙期间什么时候发出指令合适很有用。</p> <p>1 = 传输数据</p> <p>0 = 无有效数据</p>	
	[7:3]	保留。	0
	[2]	<p>DAT 线有效(ROC)。</p> <p>这位用于指出SD总线上的DAT线是否在使用。</p> <p>(a) 在读传输的情况下</p> <p>这个状态显示在SD总线上是否一个读传输在执行。在正常中断状态寄存器中，数据块之间产生一个“块间隔事件”中断，这个值从1变为0。</p> <p>这个位在下面的情况下被设置：</p> <p>(1) 读指令最后的位后</p> <p>(2) 在块间隔控制寄存器中，当写1到“继续请求”来恢复读传输。</p> <p>这个位在下面的情况下清除：</p> <p>(1) 当最后数据块的最后的位从SD总线发送到主机控制器。</p> <p>(2) 当开始等待读传输在块间隔被“停止在块间隔请求”初始化停止。</p> <p>通过在下一个中断周期开始，主机控制器将等待在下一个块间隔。如果读等待信号已经驱动（数据缓冲区不能收到数据），主机控制器能通过继续驱动读等待信号等待当前块间隔。支持读暂停/恢复功能是必要的。</p> <p>(b) 写传输条件下</p> <p>这个状态指示在SD总线上执行写传输。将这个值从1变为0，在正常中断状态寄存器产生一个传输完成中断。</p> <p>下面的情况下这个位被设置：</p> <p>(1) 写指令的结束位后</p> <p>(2) 当写1到块间隔控制寄存器的“继续请求”来继续写传输。</p> <p>下面情况下这个位被清除：</p> <p>(1) 主机控制器将检测输出是否繁忙的最后一块，SD卡释放写繁忙。如果SD卡不驱动繁忙信号对于8个SD时钟，主机控制器将认为</p>	0

		卡驱动“不繁忙”。 (2) 对于写传输当SD卡释放写繁忙优先于等待作为一个在块空白请求停止结果。 1 = DAT 线活动 0 = DAT 线不活动	
	[1]	指令禁止 (DAT) (ROC)。 (ROC) 如果DAT线活动或读传输有效被设置为1, 这个状态位产生。如果这位是0, 它指示主机控制器能发出下一个SD指令。繁忙信号指令属于指令禁止 (DAT) (不包括 R1b, R5b 字节)。从1变为0, 在正常中断状态寄存器产生一个传输完成中断。 注: 这位从1变到0后, 对于暂停传输, SD主机驱动器能保存范围为000~00Dh 的寄存器。 1 = 用DAT 线不能发出指令 0 = 能用 DAT 线发出指令	0
	[0]	指令禁止 (CMD) (ROC)。 如果这位是0, 它表示CMD线没有使用, 并且主机控制器能发出一个SD指令用CMD线。当指令寄存器 (00Fh)写入后, 这个位立即设置。当指令应答被接收, 这个位清除。甚至指令禁止 (DAT) 设置为1, 如果这位是0仅仅CMD线指令能发出。从1变为0, 在正常中断状态寄存器产生一个指令完成中断。如果由于指令冲突错误 (参考指令CRC错误), 主机控制器不能发出指令, 或者由于指令不能被“自动CMD12错误”发出, 该位将保持1, 并且指令完成不设置。发出自动CMD12状态不读取该位。 1 = 不能发出指令 0 = 不能只用 CMD 线发出指令	0

注: 在当前寄存器, 缓冲区写启动必须对DMA传输无效, 自从它产生缓冲区写就绪中断。

如图27-13所示, 显示了处理“Debouncing”硬件的状态定义。

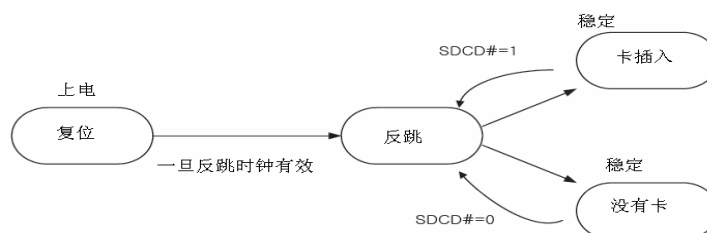


图27-13卡检测状态

指令禁止和指令禁止时间的选择框图, 如图27-14所示。

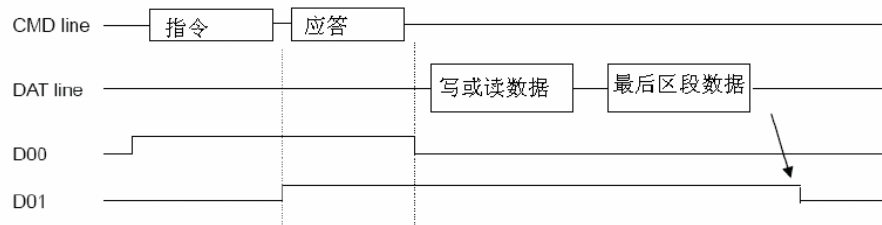


图27-14指令禁止（DAT）和指令禁止（CMD）时间选择

用于忙状态下的应答，指令禁止（DAT）时间选择，如图27-15所示。

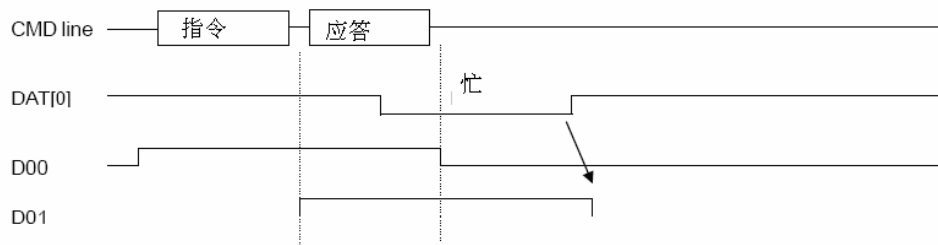


图27-15 指令禁止（DAT）时间选择

用于没有应答指令的情况下的指令禁止（CMD）时间选择，如图27-16所示。

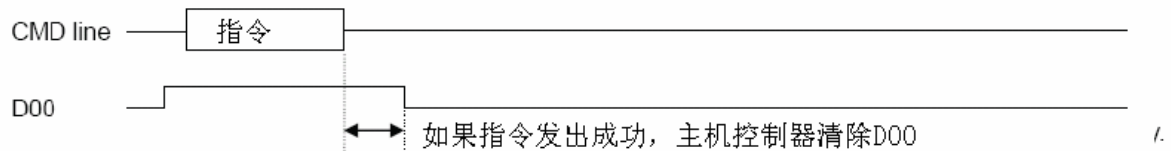


图27-16指令禁止（CMD）时间选择

## 27.4.11. 主机控制寄存器

该寄存器包含SD指令变元。

寄存器	地址	读/写	描述	复位值
HOSTCTL0	0x7C200028	读/写	当前状态寄存器(0通道)。	0x0
HOSTCTL1	0x7C300028	读/写	当前状态寄存器(1通道)。	0x0
HOSTCTL2	0x7C400028	读/写	当前状态寄存器(2通道)。	0x0

名称	位	描述	初始状态
CDSigSel	[7]	卡检测信号选择。 该位为卡测试选择源。 ‘1’ = 卡检测电平被选择 T（测试目的） ‘0’ = SDCD# 被选择（正常使用）	0
CDTestLvl	[6]	卡检测电平。 当卡检测信号设置为1时，这位激活。并且指示卡是否插入。 ‘1’ = 卡插入 ‘0’ = 没有卡	0
Wide8	[5]	扩展数据传输宽度（用于 MMC 8位卡）。 ‘1’ = 8 位操作 ‘0’ = 这个为宽度被位1指定（数据传输宽度）	0
	[4: 3]	保留。	0
	[2]	高速启动。 该位是可选择的。设置该位前，主机驱动器将在容限寄存器检查“高速支持”。如果该位设置为0（默认），主机控制器在SD时钟下降沿（25MHz）输出CMD行和DAT行。如果该位设置为1，主机控制器在SD时钟上升沿（到50MHz）输出数据DMD行和DAT行。 ‘1’ = 高速模式 ‘0’ = 正常速度模式	0
	[1]	数据传输宽度。 该位选择主机控制器的数据宽度。主机控制器将设置它匹配SD卡的数据宽度。 ‘1’ = 4位模式 ‘0’ = 1位模式	0
	[0]	LED 控制。 该位用于警告用户当SD卡正在被访问时不要移开卡。如果软件将发出一个多SD指令，该位在整个传输过程中能被设置。它不是对每一个传输必须改变的。 ‘1’ = LED 打开 ‘0’ = LED 关闭 注：LED端口被映射到 SDO_LED 引脚。	0

注：卡检测引脚电平并不简单的反映SDCD#引脚，但是从SDCD、DAT[3]或者 CDTestlvl选择依靠关于 CDSigSel 和 SDCDSel的值。

### 27.4.12. 电源控制寄存器

该寄存器包含SD指令变元。

寄存器	地址	读/写	描述	复位值
PWRCON0	0x7C200029	读/写	当前状态寄存器 (0通道)。	0x0
PWRCON1	0x7C300029	读/写	当前状态寄存器 (1通道)。	0x0
PWRCON2	0x7C400029	读/写	当前状态寄存器 (2通道)。	0x0

名称	位	描述	初始状态
	[7:4]	保留。	
	[3:1]	SD总线电压选择。 通过设置这些位，主机驱动器为SD卡选择电压水平。在设置这个寄存器前，主机驱动器将检查容限寄存器的“电压支持”位。如果选择了一个不支持的电压，主机系统将不支持SD总线电压。 ‘111b’ = 3.3V（典型值） ‘110b’ = 3.0V（典型值） ‘101b’ = 1.8V（典型值） ‘100b’ - ‘000b’ = 保留	0
	[0]	SD总线电源。 设置该位前，SD主机驱动器将设置“SD总线电压选择”。如果主机控制器检测为无卡状态，该位将被清除。如果该位被清除，主机控制器将立即停止操纵CMD和DAT[3:0]（三态），并且驱动SDCLK到低电平。 ‘1’ = 电压打开 ‘0’ = 电源关闭	0

### 7.4.13. 块间隔控制寄存器

该寄存器包含SD指令变元。

寄存器	地址	读/写	描述	复位值
BLKGAP0	0x7C20002A	读/写	块间隔控制寄存器 (0 通道)。	0x0
BLKGAP1	0x7C30002A	读/写	块间隔控制寄存器 (1 通道)。	0x0
BLKGAP2	0x7C40002A	读/写	块间隔控制寄存器 (2 通道)。	0x0

名称	位	描述	初始状态
	[7:4]	保留。	0
	[3]	<p>块间隔中断。</p> <p>该位是仅适用于 4 位模式的 SDIO 卡，并且在中断周期选择一个采样点。为多区段传输，设置 1 启动中断检测在块间隔。在多区段传输期间设置 0 使中断检测失效。在多区段传输期间，当 SD 卡不能检测一个中断信号，这位必须设置为 0。当主机驱动器检测一个 SD 卡插入，它将依据 SDIO 卡的 CCCR 来设置该位。</p> <p>‘1’ = 有效</p> <p>‘0’ = 无效</p>	0
	[2]	<p>读等待控制。</p> <p>对于 SDIO 卡，读等待功能是可选的。如果这个卡支持读等待，设置该位来利用读等待协议停止使用 DAT[2] 线读取数据。否则，主机控制器必须停止 SD 时钟来暂停读取数据，这将限制指令产生。当主机驱动器检测 SD 卡插入，它将根据 SDIO 卡的 CCCR 来设置该位。如果这卡不支持读等待，该位不能设置为 1，否则 DAT 线冲突将发生。如果该位设置为 0，则不支持挂起/恢复。(RW)</p> <p>‘1’ = 启动读等待控制</p> <p>‘0’ = 使读等待控制失效</p>	0
	[1]	<p>持续请求。</p> <p>给位用于恢复传输，该传输是通过“在块空白请求停止”被停止的。为了取消在块空白请求停止，设置“停止在块间隔”为 0，并且设置 1 来恢复传输。</p> <p>主机控制器在下面条件下自动清除该位。</p> <p>(1) 如果一个读传输，作为一个读传输恢复，“DAT 线活动”从 0 变为 1。</p> <p>(2) 如果写传输，作为写传输恢复，写“传输活动”从 0 变为 1。</p> <p>因此对于主机驱动器来说设置该位为 0 不是必须的。如果“在块空白请求停止”设置为 1，写入该位将被忽略。(RWAC)</p> <p>‘1’ = 恢复</p> <p>‘0’ = 不影响</p>	0
	[0]	<p>停止在块间隔请求。</p> <p>该位用来为 DMA 和非 DMA 传输停止执行下一个块空白的传输，直到传输完成设置为 1，表示传输完成，主机驱动器将该位设置为 1。</p> <p>清除“在块空白请求停止”和“继续请求”不会引起传输恢复。读等待用于停止在块间隔的读传输。对于写传输，主机控制器将支持在块空白请求停止，但是对于读传输，它要求 SD 卡支持读等待。因此主机驱动器在读传输期间不设置该位，除非 SD 卡支持读等待和设置“读等待控制”为 1。在写传输的情况下，主机驱动器写数据到缓冲区数据端口寄存器，在所有区段数据写完后，主机驱动器设置该位。如果该位设置为 1，主机驱动器</p>	0



		不写数据到缓冲区数据端口寄存器。 该位影响当前状态寄存器的读发送有效、写发送有效、DAT 线有效和指令禁止 (DAT)。 ‘1’ = 停止 ‘0’ = 传输	
--	--	---	--

在块间隔停止后，有三个条件来恢复传输。哪一个条件适合，依据是否主机控制器发出一个暂停指令和是否 SD 卡接受这个暂停指令。

条件如下：

- (1) 如果主机驱动器不发出暂停指令，“继续请求”不能用于恢复传输。
- (2) 如果主机驱动器发出一个暂停指令并且 SD 卡接受它，恢复指令用于恢复传输。
- (3) 如果主机驱动器发出暂停指令，SD 卡不接受它，“继续请求”用于恢复传输。

在任何时候“在块空白请求停止”停止数据传输，在尝试恢复传输前，主机驱动器将等待传输完成(在正常中断状态寄存器)。当数据传输通过“继续请求”恢复，主机驱动器清除在“在块空白请求停止”之前或者同时进行。

注：设置在块空白请求停止后，它不能清除，除非块间隔事件或者传输完成中断发生。否则，该模块挂起。

### 27.4.14. 唤醒控制寄存器

这个寄存器是强制性的，但是唤醒功能依靠主机控制器系统硬件和软件。主机驱动器维持总线上的电压，通过在电源控制器中设置 SD 总线电源为 1，唤醒时间由卡中断得到。

寄存器	地址	读/写	描述	复位值
WAKCON0	0x7C20002B	读/写	唤醒控制寄存器(0 通道)。	0x0
WAKCON1	0x7C30002B	读/写	唤醒控制寄存器(1 通道)。	0x0
WAKCON2	0x7C40002B	读/写	唤醒控制寄存器(2 通道)。	0x0

名称	位	描述	初始状态
	[7:3]	保留。	0
	[2]	唤醒事件启动在 SD 卡移除时。 该位通过在正常状态寄存器中“卡移除”声明来启动唤醒事件。CIS 中	0

		FN_WUS（唤醒支持）不影响该位。（RW） ‘1’ = 有效 ‘0’ = 无效	
	[1]	唤醒事件启动在 SD 卡插入时。 该位通过正常状态寄存器“卡插入”声明激活唤醒事件。CIS 中 FN_WUS（唤醒支持）不影响该位。（RW） ‘1’ = 有效 ‘0’ = 无效	0
	[0]	唤醒事件启动在卡中断时。 该位通过正常中断寄存器“卡中断”声明激活唤醒事件。如果 N_WUS（唤醒支持）设置为 1，该位也可以设置为 1。．（RW） ‘1’ = 有效 ‘0’ = 无效	0

## 27.4.15. 时钟控制寄存器

在主机控制器初始化时，主机驱动器依据容限寄存器设置 SDCLK 频率选择。

寄存器	地址	读/写	描述	复位值
CLKCON0	0x7C20002C	读/写	指令寄存器(0 通道)。	0x0
CLKCON1	0x7C30002C	读/写	指令寄存器(1 通道)。	0x0
CLKCON2	0x7C40002C	读/写	指令寄存器(2 通道)。	0x0

名称	位	描述	初始状态
	15:8]	SDCLK 频率选择。 该寄存器用于选择 SDCLK 引脚的频率。对于 SD 时钟，在容限寄存器中，这个频率不是直接编程，而是寄存器保存基础时钟频率的除数。只有下面的设置允许： 80h 基础时钟被 256 除 40h 基础时钟被 128 除 20h 基础时钟被 64 除 10h 基础时钟被 32 除 08h 基础时钟被 16 除 04h 基础时钟被 8 除 02h 基础时钟被 4 除 01h 基础时钟被 2 除 00h 基础时钟(10~63MHz) 设置 00h 指定 SD 时钟的最高频率。设置多位，最高位用做除数。但是	0

		<p>多位必须不设置。这两个默认除数值，能通过容限寄存器中“用于 SD 卡的基础时钟频率”计算。</p> <p>(1) 25MHz 除数值</p> <p>(2) 400KHz 除数值根据 SD 物理规范 1.01 版本和 SDIO 卡规范 1.0 版本，最大 SD 时钟频率是 25 MHz，并且绝不超过这个限制。通过下面的公式设置频率：</p> <p>时钟频率 = (基础时钟) / 除数</p> <p>因此，选择最小可能的除数，这这个除数使时钟频率少于或者等于目标频率。</p> <p>例如：如果容限寄存器中“基础时钟频率对于 SD 卡”值 33MHz，目标频率是 25MHz，选择 01h 除数值将产生 16.5MHz，它是小于或者等于目标值最近的频率。同样，对于接近一个时钟值 400kHz，除数值 40h 产生 258kHz 最佳的时钟值。</p>	
	[7:4]	保留。	0
	[3]	<p>外部时钟状态。</p> <p>写到“SD 时钟启动”在这个寄存器为 1 后，当 SD 时钟输出是稳定的，该位设置为 1。SD 主机驱动器等待发出指令来开始直到该位设置为 1。(ROC)</p> <p>‘1’ = 就绪</p> <p>‘0’ = 没有就绪</p>	0
	[2]	<p>SD 时钟启动。</p> <p>当写该位为 0，主机控制器停止 SDCLK。当该位为 0，“SDCLK 频率选择”可以被改变。接下来主机控制器将保持相同的时钟频率直到 SDCLK 停止(停止在 SDCLK=0)。如果在当前状态寄存器清除卡插入，该位将被清除。(RW)</p> <p>‘1’ = 有效</p> <p>‘0’ = 无效</p>	0
	[1]	<p>内部时钟稳定。</p> <p>在该寄存器写“内部时钟稳定”到 1 后，当 SD 时钟稳定，该位设置为 1。SD 主机驱动器将等待设置“SD 时钟启动”直到该位设置为 1。</p> <p>注：当要求设置时间的时钟震荡器用 PLL 时，这个很有意义。(ROC)</p> <p>‘1’ = 就绪</p> <p>‘0’ = 没有就绪</p>	0
	[0]	<p>中断时钟启动。</p> <p>当主机驱动器没有使用主机控制器或者主机控制器等待一个唤醒中断，该位设置为 0。主机控制器必须停止它的内部时钟到非常低能量状态。并且，寄存器将能读和写。当该位设置为 1 时，时钟开始震荡。当时钟震荡平稳，在该寄存器中，主机控制器能被设置“内部时钟状态”为 1。该位不影响卡检测。(RW)</p> <p>‘1’ = 震荡</p> <p>‘0’ = 停止</p>	

### 27.4.16. 超时控制寄存器

主机控制器初始化时，主机驱动器能根据容限寄存器设置数据超时计数器值。

寄存器	地址	读/写	描述	复位值
CMDREG0	0x7C20002E	读/写	超时控制寄存器(0 通道)。	0x0
CMDREG1	0x7C30002E	读/写	超时控制寄存器(1 通道)。	0x0
CMDREG2	0x7C40002E	读/写	超时控制寄存器(2 通道)。	0x0

名称	位	描述	初始状态
	[7:4]	保留。	0
	[3:0]	数据超时计数器值。 这个值决定时间间隔，通过数据线超时设定被检测检测。对于指示超时产生的信息因素，参考错误中断寄存器的“数据超时错误”。通过划分基础时钟 TMCLK 值，产生超时时钟频率。当设置这个寄存器时，通过清除“数据超时错误状态启动”（在错误中断状态启动寄存器中）来预防引起不慎超时事件。 1111b 保留 1110b TMCLK x 227 1101b TMCLK x 226 ..... 0001b TMCLK x 214 0000b TMCLK x 213	0

### 27.4.17. 软件复位寄存器

写 1 到这个寄存器的每一位，则产生复位脉冲。复位完成后，主机控制器清除每一位。由于完成软件复位要花一些时间，SD 主机驱动器将确认这些位是 0。

寄存器	地址	读/写	描述	复位值
SWRST0	0x7C20002F	读/写	软件复位寄存器(0 通道)。	0x0
SWRST1	0x7C30002F	读/写	软件复位寄存器(1 通道)。	0x0
SWRST2	0x7C40002F	读/写	软件复位寄存器(2 通道)。	0x0

名称	位	描述	初始状态
	[7:3]	保留。	0
	[2]	软件复位对于 DAT 线。 只复位部分数据电路。也复位 DMA 电路。(RWAC) 下面的寄存器和位通过该位清除： 缓冲区数据端口寄存器：缓冲区被读取和初始化 当前状态寄存器：缓冲区读启动、缓冲区写启动、读传输有效、写传输有效、DAT 线有效、指令禁止 (DAT) 块间隔控制寄存器：继续请求、停止在块间隔请求 正常中断状态寄存器：缓冲区读就绪、缓冲区写就绪、DMA 中断、块间隔事件、传输完成。 ‘1’ = 复位 ‘0’ = 工作	0
	[1]	软件复位对于 CMD 线。 只有部分指令电路复位。(RWAC) 下面的寄存器和位通过该位清除： 当前状态寄存器 指令禁止 (CMD) 正常中断状态寄存器 指令完成 ‘1’ = 复位 ‘0’ = 工作	0
	[0]	软件复位对于所有的。 除了卡检测电路，复位影响整个主机控制器。类型 ROC, RW, RW1C, RWAC 寄存器位清除为 0。在初始化期间，主机驱动器设置这位为 1 来复位主机控制器。当容限寄存器有效并且主机驱动器能读它们，主机控制器复位到 0。另外，使用“软件复位对于所有”不影响容限寄存器的这个值。如果该位设置为 1, SD 卡将复位本身，并且必须被主机驱动器初始化。(RWAC) ‘1’ = 复位 ‘0’ = 工作	0

## 27.4.18. 正常中断状态寄存器

正常中断状态启动影响这个寄存器的读，但是正常中断信号启动就没有这个影响。当正常中断信号启动激活，并且至少一个状态位设置为 1，一个中断产生。除了卡中断和错误中断位外，对于所有的位写 1 到一个位来清除它；写 0 到这些位保持不变。通过写入一个信号寄存器，超过一个状态能被清除。当卡停止中断有效时，卡中断被清除，也就是，卡驱动器为中断提供服务的条件。

寄存器	地址	读/写	描述	复位值
NORINTSTS0	0x7C200030	ROC/RW1C	正常中断状态寄存器(0 通道)。	0x0
NORINTSTS1	0x7C300030	ROC/RW1C	正常中断状态寄存器(1 通道)。	0x0
NORINTSTS2	0x7C400030	ROC/RW1C	正常中断状态寄存器(2 通道)。	0x0

名称	位	描述	初始状态
	[15]	错误中断。 如果错误中断寄存器的任何位被设置，然后将设置这个位。。因此通过首先检查这个位,对于任何错误,主机驱动器能有效测试。这个位是只读。(ROC) ‘0’ = 无错误 ’ 1’ = 错误	0
StaFIA3	[14]	FIFO SD 地址指示器中断 3 状态 (RW1C)。 ‘0’ = 发生 ’ 1’ = 没有发生 当 SD 时钟 FIFO 地址达到 FIFO 中断地址寄存器 3 的值,该状态位有效。	0
StaFIA2	[13]	FIFO SD 地址指示器中断 2 状态 (RW1C)。 ‘0’ = 发生 ’ 1’ = 没有发生 当 SD 时钟 FIFO 地址达到 FIFO 中断地址寄存器 2 的值,该状态位有效。	0
StaFIA1	[12]	FIFO SD 地址指示器中断 1 状态 (RW1C)。 ‘0’ = 发生 ’ 1’ = 没有发生 当 SD 时钟 FIFO 地址达到 FIFO 中断地址寄存器 1 的值,该状态位有效。	0
StaFIA0	[11]	FIFO SD 地址指示器中断 0 状态 (RW1C)。 ‘0’ = 发生 ’ 1’ = 没有发生 当 SD 时钟 FIFO 地址达到 FIFO 中断地址寄存器 0 的值,该状态位有效。	0
StaRWaitInt	[10]	读等待中断状态 (RW1C)。 ‘0’ = 读等待中断发生’ 1’ = 读等待中断没有发生 注：当检查暂停指令的应答后，如果 BS = 0，自动释放读等待中断状态。	0
StaCCS	[9]	CCS 中断状态 (RW1C)。 指令完成信号中断状态位用于 CE-ATA 模式。 ‘0’ = CCS 中断发生	0

		’ 1’ = CCS 中断没有发生	
	[8]	<p>卡 中断。</p> <p>将这位写 1 不清除这位。它被清除通过复位 SD 卡中断因子。在 1 位模式中，主机控制器将检测卡中断，没有 SD 时钟来支持唤醒。在 4 位模式中，中断循环期间，卡中断信号被取样，因此在来自 SD 卡的中断信号和到主机系统的中断之间有许多样本延迟。当这个状态已经被设置，并且主机驱动器需要开始中断服务时，为了清除主机控制器中的卡中断状态锁和停止操纵中断信号到主机系统，正常中断状态启动寄存器中卡中断状态启动必须设置为 0。卡中断服务（它必须复位在 SD 卡的中断因子，并且中断信号可能无效）完成后，设置卡中断状态启动为 1 和再次开始取样中断信号。（ROC, RW1C）</p> <p>‘1’ =产生卡中断</p> <p>‘0’ = 无卡中断</p>	0
	[7]	<p>卡移除。</p> <p>如果当前状态寄存器的卡插入从 1 变到 0，该状态被设置。当主机驱动器写这位为 1 来清除这个状态，当前状态寄存器中的卡插入状态必须被确认。因为当主机驱动器清除该位时卡检测状态可能改变，并且可能不产生中断事件。（RW1C）</p> <p>‘1’ = 移除</p> <p>‘0’ = 卡状态稳定或抖动</p>	0
	[6]	<p>卡插入。</p> <p>如果当前状态寄存器中卡插入从 0 变为 1，该位被设置。当主机驱动器将该位置 1 来清除这个状态时，当前状态寄存器的卡插入状态必须被确认。因为当主机驱动器清楚该位时卡检测状态可能改变，并且可能不产生中断事件。（RW1C）</p> <p>‘1’ = 卡插入</p> <p>‘0’ = 卡状态稳定或抖动</p>	0
	[5]	<p>缓冲区读就绪。</p> <p>如果缓冲区读启动从 0 变到 1，该状态被设置。参考当前状态寄存器中的缓冲区读启动。（RW1C）</p> <p>‘1’ = 准备读缓冲区</p> <p>’ 0’ =没有准备读缓冲区</p>	0
	[4]	<p>缓冲区写就绪。</p> <p>如果缓冲区写启动从 0 变到 1，该状态被设置。参考当前状态寄存器中的缓冲区写启动。（RW1C）</p> <p>‘1’ = 准备写入缓冲区</p> <p>’ 0’ =没有准备写入缓冲区</p>	0
	[3]	<p>DMA 中断。</p> <p>如果传输期间主机控制器检测主机 DMA 缓冲区边界，则该状态被设置。参考块大小寄存器中主机 DMA 缓冲区边界。其它 DMA 中断</p>	0

		因素可能在将来添加。传输完成后，其不能产生中断。(RW1C) ‘1’ = DMA 中断产生 ‘0’ = 非 DMA 中断																	
	[2]	块间隔事件。 如果块间隔控制寄存器中“停止在块间隔请求”被设置，当读/写传输在块间隔被停止时，该位被设置。如果“停止在块间隔请求”没有被设置为 1，该位不设置为 1。(RW1C) (1) 在读传输的情况下 该位在 DAT 线活动状态的下降沿（当传输被停止在 SD 总线传输时间选择）被设置。为了使用这个功能，读等待必须停止。 (2) 写传输的情况下 该位在写传输活动状态的下降沿（在 SD 总线时间选择获得 CRC 状态后）被设置。 ‘1’ = 传输被停止在块间隔 ‘0’ = 无块间隔事件	0																
	[1]	传输完成。 当读/写传输完成，该位被设置。 (1) 读传输情况下 该位在读传输活动状态下下降沿被设置。有两种产生中断的情况。第一种是数据传输完成。第二种是数据传输被停止在块间隔，和通过设置块间隔控制寄存器中的“停止在块间隔请求”完成数据传输。 (2) 写传输情况下 该位在 DAT 线活动状态的下降沿被设置。有两种产生中断的情况。第一种是最后的数据被写入 SD 卡，并且释放繁忙信号。第二种情况是数据传输被停止在块间隔，和通过设置块间隔控制寄存器中的“停止在块间隔请求”完成数据传输。 下表显示数据传输完成比数据超时错误有更高的优先权。如果两位都设置为 1，认为数据传输完成。 <table><tr><td>传输完成</td><td>数据超时错误</td><td>状态含义</td><td></td></tr><tr><td>0</td><td>0</td><td>因其它因素中断</td><td></td></tr><tr><td>0</td><td>1</td><td>数据传输过程中超时错误</td><td></td></tr><tr><td>1</td><td>-</td><td>传输完成</td><td></td></tr></table> ‘1’ = 数据传输完成 ‘0’ = 无数据传输完成	传输完成	数据超时错误	状态含义		0	0	因其它因素中断		0	1	数据传输过程中超时错误		1	-	传输完成		0
传输完成	数据超时错误	状态含义																	
0	0	因其它因素中断																	
0	1	数据传输过程中超时错误																	
1	-	传输完成																	
	[0]	指令完成。 当获得指令应答的最后一位（除了自动 CMD12）时，该位被设置。参考当前状态寄存器中指令禁止（CMD）。下表表明指令超时错误比指令完成有更高的优先权。如果两位都设置位 1，则认为应答没有正确接收。(RW1C)	0																



指令完成	指令超时错误	状态的含义	
0	0	被另一个因素中断	
不关心	1	64 SDCLK 周期内不接收应答	
1	0	应答被接收	
‘1’ =指令完成			
‘0’ = 无指令完成			

注：通过轮询和监视 INTREQ 端口，主机驱动器将检查是否中断真的被清除。如果 HCLK 比 SDCLK 更快，该位要花更多的时间被清除。

注：卡中断状态位保持先前的值直到下一个卡中断（一级中断），并且当置 1 的时候能被清除。

### 27.4.19. 错误中断状态寄存器

通过错误中断状态启动寄存器能启动这个寄存器中的信号定义，但是不能通过错误中断信号启动寄存器。当错误中断信号启动激活，并且至少一个状态设置为 1，中断产生。写 1 清除这位，写 0 保持这位不变。在一个寄存器写，超过一个状态能被清除。

寄存器	地址	读/写	描述	复位值
ERRINTSTS0	0x7C200032	ROC/RW1C	错误中断状态寄存器(0 通道)。	0x0
ERRINTSTS1	0x7C300032	ROC/RW1C	错误中断状态寄存器(1 通道)。	0x0
ERRINTSTS2	0x7C400032	ROC/RW1C	错误中断状态寄存器(2 通道)。	0x0

名称	位	描述	初始状态
	[15:9]	保留。	0
	[8]	自动 CMD12 错误。 当检测（自动 CMD12 错误状态寄存器中的一位）已经从 0 变为 1 时发生。该位设置为 1，不仅错误在自动 CMD12 发生，而且由于先前的指令错误，自动 CMD12 不被执行。 ‘1’ = 错误 ‘0’ = 无错误	0
	[7]	当前限制错误。 不执行此版本。始终为 0	0
	[6]	数据最后位错误。	0

		当检测 0 在最后位位置（使用 DAT 线）或者 CRC 状态的最后位位置时发生。 ‘1’ = 错误 ‘0’ = 无错误	
	[5]	数据 CRC 错误。 检测 CRC 错误时，用 DAT 线传输读取的数据，或者当检测写 CRC 状态有一个其它的值而不是“010”时，数据 CRC 错误发生 ‘1’ = 错误 ‘0’ = 无错误	0
	[4]	数据超时错误。 当检测到下面的一种时发生： （1） 对于 R1b, R5b 类型忙超时 （2） 写入 CRC 状态后忙超时 （3） 写入 CRC 状态超时 （4） 读数据超时 ‘1’ = 超时 ‘0’ = 无错误	0
	[3]	如果在指令响应中一个指令索引错误产生，指令索引错误产生。 ‘1’ = 错误 ‘0’ = 无错误	0
	[2]	指令末位错误。 当检测到指令的最后位时，错误产生。 ‘1’ = 最后位错误产生 ‘0’ = 无错误	0
	[1]	指令 CRC 错误。 在下面两种情况下指令 CRC 错误产生： （1）如果应答被返回和指令超时错误被设置为 0（表示没有超时），当在指令应答检测一个 CRC 错误时，该位设置为 1。 （2）当发出一个命令，通过监测 CMD 线主机控制器检测一个 CMD 线。如果主机控制器驱动 CMD 线到 1 级，但在下一个 SDCLK 缓冲中检测 CMD 线上的 0 级，那么主机控制器将中止命令（停止驱动 CMD 线）和设置此位为 1。命令超时错误也将被设置为 1 用于区别 CMD 线的冲突。 ‘1’ = CRC 错误产生 ‘0’ = 无错误	0
	[0]	指令超时错误。 如果没有应答在 64 SDCLK 周期内被从指令的最后位返回，则错误发生。如果主机控制器测试到一个 CMD 线冲突，在这种情况下，超时错误将被设置，如表 27-34 所示。由于指令将被主机控制器中断，没有等待 64 SDCLK 周期情况下，该位也将被设置。 ‘1’ = 超时 ‘0’ = 无错误	0

指令 CRC 错误和指令超时错误关系如表 27-4 所示。

表 27-4 指令 CRC 错误和指令超时错误关系

指令 CRC 错误	指令超时错误	错误种类
0	0	无错误
0	1	应答超时错误
1	0	应答 CRC 错误
1	1	CMD 线冲突

## 27.4.20. 正常中断状态启动寄存器

设置 1 来使中断状态启动。

寄存器	地址	读/写	描述	复位值
NORINTSTSEN0	0x7C200034	读/写	正常中断状态启动寄存器(0 通道)。	0x0
NORINTSTSEN1	0x7C300034	读/写	正常中断状态启动寄存器(1 通道)。	0x0
NORINTSTSEN2	0x7C400034	读/写	正常中断状态启动寄存器(2 通道)。	0x0

名称	位	描述	初始状态
	[15]	固定到 0。 通过错误中断状态启动寄存器，主机驱动器将控制错误中断。(R0)	0
EnStaFIA3	[14]	FIFO SD 地址指示字中断 3 状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
EnStaFIA2	[13]	FIFO SD 地址指示字中断 2 状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
EnStaFIA1	[12]	FIFO SD 地址指示字中断 1 状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
EnStaFIA0	[11]	FIFO SD 地址指示字中断 0 状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
EnStaRWait	[10]	读等待中断状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
EnStaCCS	[9]	CCS 中断状态有效。	0

		‘1’ = 有效 ‘0’ = 屏蔽	
	[8]	卡中断状态有效。 如果这个位被设置为 0，主机控制器清除中断请求到系统。这个位被设置为 1 时，这个位被清除和恢复，卡中断检测停止。必须重新这个位从卡被清除到阻止无意的中断，在所有中断请求后，维修卡中断前主机控制器必须再次清除卡中断状态启动。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[7]	卡移动状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[6]	卡插入状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[5]	缓冲区读就绪信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[4]	缓冲区写就绪信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[3]	DMA 中断状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[2]	块间隔事件状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[1]	传输完成状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[0]	指令完成状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0

### 27.4.21. 错误中断状态启动寄存器

设置 1 来激活错误中断状态。

寄存器	地址	读/写	描述	复位值
-----	----	-----	----	-----

ERRINTSTSEN0	0x7C200036	读/写	错误中断状态启动寄存器(0 通道)。	0x0
ERRINTSTSEN1	0x7C300036	读/写	错误中断状态启动寄存器(1 通道)。	0x0
ERRINTSTSEN2	0x7C400036	读/写	错误中断状态启动寄存器(2 通道)。	0x0

名称	位	描述	初始状态
	[15:9]	保留。	0
	[8]	自动 CMD12 错误状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[7]	当前限制错误状态有效。 这个功能在该版本不执行。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[6]	数据最后位错误状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[5]	数据 CRC 错误状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[4]	数据超时错误状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[3]	指令索引错误状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[2]	指令最后位错误状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[1]	指令 CRC 错误状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[0]	指令超时错误状态有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0

### 27.4.22. 正常中断信号启动寄存器

该寄存器用于选择主机系统哪一个中断状态作为中断。这些中断状态将共享相同的 1 位中断线。

寄存器	地址	读/写	描述	复位值
NORINTSIGEN0	0x7C200038	读/写	正常中断信号启动寄存器(0 通道)。	0x0
NORINTSIGEN1	0x7C300038	读/写	正常中断信号启动寄存器(1 通道)。	0x0
NORINTSIGEN2	0x7C400038	读/写	正常中断信号启动寄存器(2 通道)。	0x0

名称	位	描述	初始状态
	[15]	固定为 0。 主设备将利用错误中断信号启动寄存器控制错误中断。	0
EnSigFIA3	[14]	FIFO SD 地址指示器中断 3 信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
EnSigFIA2	[13]	FIFO SD 地址指示器中断 2 信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
EnSigFIA1	[12]	FIFO SD 地址指示器中断 1 信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
EnSigFIA0	[11]	FIFO SD 地址指示器中断 0 信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
EnSigRWait	[10]	读等待中断信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
EnSigCCS	[9]	CCS 中断信号有效。 指令完成信号中断位用于 CE-ATA 接口模式。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[8]	卡中断信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[7]	卡移除信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0

	[6]	卡插入信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[5]	缓冲区读就绪信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[4]	缓冲区写就绪信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[3]	DMA 中断信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[2]	块间隔事件信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[1]	传输完成信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[0]	命令完成信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0

27.4.23. 错误中断信号启动寄存器

该寄存器用于选择主机系统哪一个中断状态作为中断。所有的状态位共享相同的 1 位中断线。

寄存器	地址	读/写	描述	复位值
ERRINTSIGEN0	0x7C20003A	读/写	错误中断信号启动寄存器(0 通道)。	0x0
ERRINTSIGEN1	0x7C30003A	读/写	错误中断信号启动寄存器(1 通道)。	0x0
ERRINTSIGEN2	0x7C40003A	读/写	错误中断信号启动寄存器(2 通道)。	0x0

名称	位	描述	初始状态
	[15:9]	保留。	0
	[8]	自动 CMD12 错误信号启动。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[7]	当前限制错误信号有效。	0

		这个功能在这个版本中不能被执行。 ‘1’ = 有效 ‘0’ = 屏蔽	
	[6]	数据最后位错误信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[5]	数据 CRC 错误信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[4]	数据超时错误信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[3]	指令索引错误信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[2]	命令最后位错误信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[1]	指令 CRC 错误信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0
	[0]	指令超时错误信号有效。 ‘1’ = 有效 ‘0’ = 屏蔽	0

## 27.4.24. 自动 CMD12 错误状态寄存器

当自动 CMD12 错误状态被设置，主机驱动器检查这个寄存器来鉴别自动 CMD12 显示的是什么类型错误。这个寄存器只有当自动 CMD12 错误被设置时有效。

寄存器	地址	读/写	描述	复位值
ACMD12ERRSTS0	0x7C20003C	ROC	自动 CMD12 错误状态寄存器 (0 通道)。	0x0
ACMD12ERRSTS1	0x7C30003C	ROC	自动 CMD12 错误状态寄存器 (1 通道)。	0x0
ACMD12ERRSTS2	0x7C40003C	ROC	自动 CMD12 错误状态寄存器 (2 通道)。	0x0

名称	位	描述	初始状态
	[15: 8]	保留。	0



	[7]	指令不被“自动 CMD12 错误”发出 设置该位为 1 意味着 CMD_wo_DAT 不被执行,由于寄存器中自动 CMD12 错误 (D04~D01)。 ‘1’ = 没有发出 ‘0’ = 无错误	0
	[6:5]	保留。	0
	[4]	自动 CMD12 索引错误。 如果对指令的应答中指令索引发生错误,该错误发生。 ‘1’ = 错误 ‘0’ = 无错误	0
	[3]	自动 CMD12 末位错误。 当检测指令末位的应答是 0 时,该错误发生。 ‘1’ = 错误发生 ‘0’ = 无错误	0
	[2]	自动 CMD12 CRC 错误。 当在指令应答检测到 CRC 错误时,该错误发生。 ‘1’ = CRC 错误发生 ‘0’ = 无错误	0
	[1]	自动 CMD12 超时错误。 如果在 64 SDCLK 周期内没有应答返回。如果该位设置为 1,其它的错误状态位 (D04-D02) 无意义。 ‘1’ = 超时 ‘0’ = 无错误	0
	[0]	自动 CMD12 没有执行。 如果由于指令错误,存储器多区段数据传输没有开始,该位不设置,因为它不需要发出自动 CMD12。设置该位到 1,意思是由于一些错误主机控制器不能发出自动 CMD12 来停止存储器多时钟数据传输。如果该位设置为 1,其它的错误状态 (D04-D01) 位无意义。 ‘1’ = 不执行 ‘0’ = 执行	0

自动 CMD12 CRC 错误和 CMD12 超时错误的关系,如表 27-5 所示。

表 27-5 自动 CMD12 CRC 错误和 CMD12 超时错误的关系

自动 CMD12 CRC 错误	自动 CMD12 超时错误	错误种类
0	0	无错误
0	1	应答超时错误
1	0	应答 CRC 错误

1	1	CMD 线冲突
---	---	---------

### 27.4.25. 容限寄存器

对于主机控制器执行，该寄存器为主机驱动器提供了详细信息。上电初始化期间，主机控制器执行这些值作为固定的或者从闪存载入。

寄存器	地址	读/写	描述	复位值
CAPAREG0	0x7C200040	读/写	容限寄存器(0 通道)。	0x05E00080
CAPAREG1	0x7C300040	读/写	容限寄存器(1 通道)。	0x05E00080
CAPAREG2	0x7C400040	读/写	容限寄存器(2 通道)。	0x05E00080

名称	位	描述	初始状态
	[31:27]	保留。	1
	[26]	电压支持 1.8V (HWInit)。 ‘1’ =支持 1.8V ‘0’ =不支持 1.8V	0
	[25]	电压支持 3.0V (HWInit)。 ‘1’ =支持 3.0V ‘0’ =不支持 3.0V	1
	[24]	电压支持 3.3V (HWInit)。 ‘1’ =支持 3.3V ‘0’ =不支持 3.3V	1
	[23]	暂停/恢复支持 (HWInit)。 这个位指示是否主机控制器支持暂停/恢复功能。如果这个位置是 0，不支持延缓/恢复设置，并主机驱动器也不发出暂停/恢复指令。 ‘1’ = 支持 ‘0’ = 不支持	1
	[22]	DMA 支持(HWInit)。 这个位指示是否主机控制器是能够使用 DMA 直接传输数据在系统内存和主机控制器之间。	1

		‘1’ = 支持 DMA ‘0’ = 不支持 DMA	
	[21]	高速支持 (HWInit)。 这个位显示是否主机控制器和主系统支持高速模式，并且它们能支持从 25~50MHz 的 SD 时钟频率。 ‘1’ =支持高速 ‘0’ =不支持高速	1
	[20:18]	保留。	0
	[17:16]	最大块的长度(HWInit)。 这个值显示主机驱动器在主控制器里能读写到缓冲器最大块的大小。不需要等待周期的缓冲器传输块的大小。3 个块的大小被定义如下显示： ‘00’ = 512 字节      ‘01’ = 1024 字节 ‘10’ = 2048 字节    ‘11’ = 保留	0
	[15:14]	保留。	0
	[13:8]	基础时钟频率对于 SD 时钟 (HWInit)。 这个值显示适用于 SD 时钟的基础时钟频率（最大）。 单元值是 1MHz。如果真实的频率是 16.5MHz，较大的值被设置为 01 0001b (17MHz)，因为主机驱动器使用这个值计算时钟分配器的值（参考时钟控制寄存器中 SDCLK 频率选择）。并且不能超过 SD 时钟频率的上限。支持时钟范围是 10~63MHz。如果这些位都是 0，主系统不得不通过其它方法得到信息。  非 ‘0’ =1~63MHz 000000b =通过其它方法得到信息	0
	[7]	超时时钟单位(HWInit)。 这个位显示用来检测数据的超时错误的基础时钟频率的单位。 ‘0’ =kHz,    ‘1’ =MHz	1
	[6]	保留。	0
	[5:0]	超时时钟频率 (HWInit)。 这个位显示用于检测数据超时错误的基础时钟频率。超时时钟单位详细说明了这个域值的单位。 超时时钟单位 =0 [kHz] 单位：1~63KHz 超时时钟单位 =1 [MHz] 单位：1~63MHz 没有 0 = 1kHz 到 63kHz 或 1~63MHz 00 0000b =通过其它方法得到信息	0

## 27.4.26. 最大电流容限寄存器

这个寄存器指示最大的电流容限。如果电压支持在容限寄存器中被设置，那么这个最大电流容限值是

很有意义的。如果主机系统通过另一种方法提供这个信息，所有的最大电流容限寄存器将是 0。

寄存器	地址	读/写	描述	复位值
MAXCURR0	0x7C200048	HWInit	最大电流容限寄存器(0 通道)。	0x0
MAXCURR1	0x7C300048	HWInit	最大电流容限寄存器(1 通道)。	0x0
MAXCURR2	0x7C400048	HWInit	最大电流容限寄存器(2 通道)。	0x0

名称	位	描述	初始状态
	[31:24]	保留。	
	[23:16]	对于 1.8V (HWInit)最大电流。	0
	[15:8]	对于 3.0V (HWInit)最大电流。	0
	[7:0]	对于 3.3V (HWInit)最大电流。	0

如表 27-6 所示，描述了每种电平的电流支持。

表 27-6 最大电流值定义

寄存器值	电流值
0	通过其它方式获得信息
1	4mA
2	8mA
3	12mA
...	...
255	1020mA

27.4.27. 控制寄存器 2

该寄存器包含 SD 指令变元。

寄存器	地址	读/写	描述	复位值
CONTROL2_0	0x7C200080	读/写	控制寄存器 2 (0 通道)。	0x0
CONTROL2_1	0x7C300080	读/写	控制寄存器 2 (1 通道)。	0x0
CONTROL2_2	0x7C400080	读/写	控制寄存器 2 (2 通道)。	0x0

名称	位	描述	初始状态
Reserved	[31]	保留。	0
CmdCnfmask	[30]	指令冲突屏蔽启动。 这个位可以屏蔽启动命令冲突状态（“错误中断状态寄存器”的位[1: 0]）。 0 = 屏蔽无效     1 = 屏蔽有效	0
CDInvRXD	[29]	卡检测信号倒置对于 RX_DAT[3]。 0=禁止, 1=有效	0
SelCardOut	[28]	卡移除条件选择。 0= 卡移除条件是“无卡插入” 状态。 1= 卡移除状态是“卡拔出” 状态。	0
FltClkSel	[27: 24]	滤波器时钟 (iFLTCLK) 选择。 滤波器时钟周期 = $2^{(FltClkSel + 5)} \times iSDCLK$ 周期 0000 = $25 \times iSDCLK$ , 0001 = $26 \times iSDCLK \dots 1111 = 220 \times iSDCLK$	0
LvlDAT	[23: 16]	DAT 线电平。 BIT[23]=DAT[7], BIT[22]=DAT[6], BIT[21]=DAT[5], BIT[20]=DAT[4], BIT[19]=DAT[3], BIT[18]=DAT[2], BIT[17]=DAT[1], BIT[16]=DAT[0] (只读)	线状态
EnFBCLKT	[15]	反馈时钟有效对于发送数据/指令时钟。 ‘0’ =禁止, ‘1’ =有效	0
EnFBCLKR	[14]	反馈时钟有效对于接收数据/指令时钟。 ‘0’ =禁止, ‘1’ =有效	0
SDCDSe1	[13]	SD 卡检测信号选择。 卡检测引脚电平并不是简单地反映 SDCD#引脚, 但是从 SDCD, DAT[3], 或者 CDTestlvl 中选择依据 CDSigSel 和该域 (SDCDSe1) 的值。 ‘0’ =nSDCD 用于 SD 卡检测信号 ‘1’ =DAT[3] 用于 SD 卡检测信号	0
CardSync	[12]	SD 卡检测同步支持。 该区域用于有效输出 CMD 和 DAT, 当设置时, 参考 PWRCON 寄存器 SD 总线电源位。 ‘0’ =不同步, 没有交换输出有效信号(指令, 数据) ‘1’ =同步, 控制输出有效信号(指令, 数据)	0
TxBStartEn	[11]	CE-ATA I/F 模式。 发送数据启动状态前忙碌状态检测。 0=禁止, 1=有效	0
DFCnt	[10: 9]	反跳滤波器计数。 反跳滤波器计数设置寄存器用于卡检测信号输入 (SDCD#)。 00 = 不使用反跳滤波器     01 = $4 \times iSDCLK$ 10 = $16 \times iSDCLK$ 11 = $64 \times iSDCLK$	0
EnSCHold	[8]	SDCLK 操作有效。	0

		通过主机控制器，完成出入时钟操作状态。 0=禁止， 1=有效	
RwaitMode	[7]	读等待释放控制。 0 = 主机控制器释放读等待状态（自动） 1 = 主机设备释放读等待状态（手动）	0
DisBufRD	[6]	缓冲器读取禁止。 0 = 正常模式，用 0x20 寄存器使用者可以读缓冲区（FIFO）数据 1 =使用者不能用 0x20 寄存器读取到缓冲器（FIFO）的数据。在此情况下，只能通过存储器区域读取缓冲区存储器。（用于调试）	0
SelBaseClk	[5: 4]	基础时钟源选择。 00 或 01 = HCLK，10 = EPLL 输出时钟（来自系统） 11 = 外部时钟源（XTI 或 XEXTCLK）	00
PwrSync	[3]	SD OP 电源同步支持 SD 卡。 该域用于使输入 CMD 和 DAT 有效。当设置时，参考 PWRCON 寄存器的 SD 总线电源位。 ‘0’ = 不同步，没有转换输入有效信号（指令，数据） ‘1’ = 同步，控制输入有效信号（指令，数据）	0
ModePwrPin	[2]	保留。	0
EnSDCLKmsk	[1]	当卡插入被清除时，SDCLK 输出时钟屏蔽。当处于无卡状态时，设置该区域为高位来停止 SDCLK。 ‘0’ = 禁止， ‘1’ = 有效	0
HwInitFin	[0]	SD 主机控制器硬件初始化完成。 0 = 未完成     1 = 完成	0

注：如果卡不支持读取等待保证以保证接收数据不会被覆盖到内部 FIFO 存储器，始终确保设置 SDCLK 有效启用 (EnSCHold)

注：读传输过程中，当 SDCLK 控制启动被设置时，CMD\_wo\_DAT 发出被禁止。

### 27.4.28. 控制寄存器 3 寄存器

寄存器	地址	读/写	描述	复位值
CONTROL3_0	0x7C200084	读/写	FIFO 中断控制(控制寄存器 3) (0 通道)。	0x7F5F3F1F
CONTROL3_1	0x7C300084	读/写	FIFO 中断控制(控制寄存器 3) (1 通道)。	0x7F5F3F1F
CONTROL3_2	0x7C400084	读/写	FIFO 中断控制(控制寄存器 3) (2 通道)。	0x7F5F3F1F

名称	位	描述	初始状态
----	---	----	------

FCSe13	[31]	反馈时钟选择[3]。 参考注(1)	0x0
FIA3	[30: 24]	FIFO 中断地址寄存器 3。 FIFO（512 字节缓冲存储器，字地址单元）。 初始值（0x7F）产生在 512 字节（128 字）的位置。	0x7F
FCSe12	[23]	反馈时钟选择[2]。 参考注(1)	0x0
FIA2	[22: 16]	FIFO 中断地址寄存器 2。 FIFO（512 字节缓冲存储器，字地址单元）。 初始值（0x5F）产生在 384 字节（96 字）的位置。	0x5F
FCSe11	[15]	反馈时钟选择[1]。 参考注（2）	0x0
FIA1	[14: 8]	FIFO 中断地址寄存器 1。 FIFO（512 字节存储器，字地址单元）。 初始值（0x3F）产生在 256 字节（64 字）的位置。	0x3F
FCSe10	[7]	反馈时钟选择[0]。 参考注（2）	0x0
FIA0	[6: 0]	FIFO 中断地址寄存器 0。 FIFO（512 字节缓冲存储器，字地址单元）。 初始值（0x1F）产生在 128 字节（32 字）的位置。	0x1F

注（1）：FCSe1[3:2]：传输反馈时钟延迟控制

‘01’=延迟 1（较小延迟），‘11’=延迟 2，‘00’=延迟 3，‘10’=延迟 4（较多延迟）

注（2）：FCSe1[1:0]：接收反馈时钟延迟控制

‘00’=延迟 1（较少延迟），‘01’=延迟 2，‘10’=延迟 3，‘11’=延迟 4（较多延迟）

27.4.29. 主机控制器版本寄存器

该寄存器包含 SD 指令变元。

寄存器	地址	读/写	描述	复位值
HCVER0	0x7C2000FE	HWInit	主机控制版本寄存器（0 通道）。	0x1300
HCVER1	0x7C3000FE	HWInit	主机控制版本寄存器（1 通道）。	0x1300
HCVER2	0x7C4000FE	HWInit	主机控制版本寄存器（2 通道）。	0x1300

名称	位	描述	初始状态
----	---	----	------

	[15:8]	供应商的版本号。 为供应商的版本号所保留的状态。主设备不用这种状态。 0x3 : SDMMC3.0 主机控制器。	0x13
	[7:0]	规范版本号。 该状态指示规范版本。较高和较低 4 位表示版本。 ‘00’ = SD 主机规范版本 1.0 其它 = 保留	0x00



# 28 MIPI HIS 接口控制器

## 28.1 概述

MIPI HIS 接口是一种高速同步串行接口。

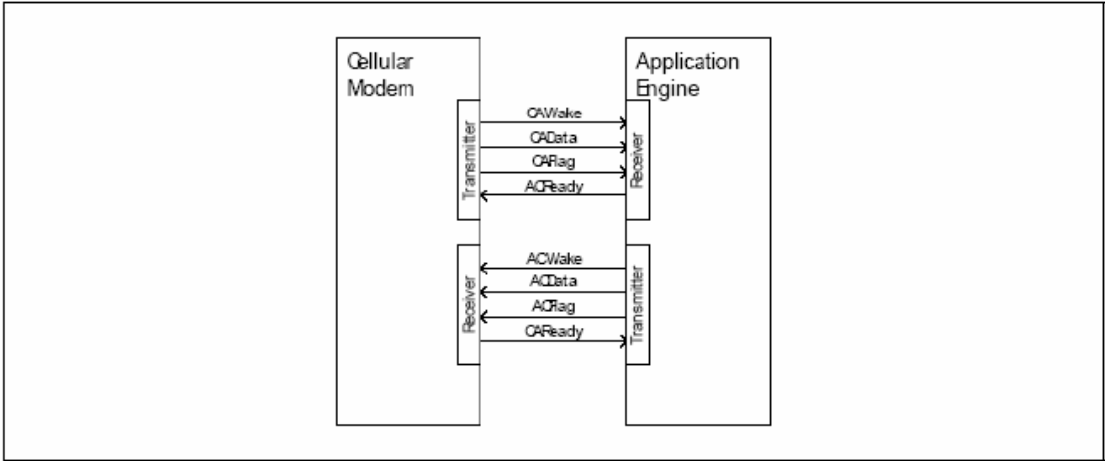


图 28-1 MIPI HIS 信号定义模块图

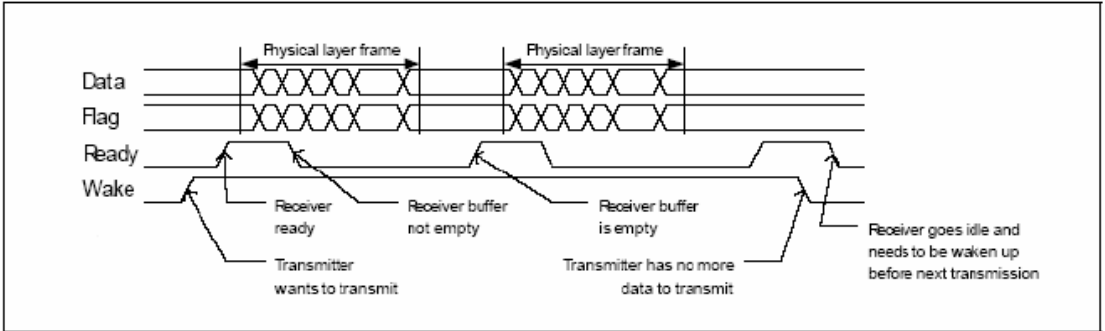


图 28-2 MIPI HIS 传输实例模块图

## 28.1.性能

MIPI HIS Rx/Tx 控制器主要性能如下：

MIPI HIS 接口是单向接口。

MIPI HIS RX 最大带宽是 100Mbps，MIPI HIS Tx 控制器使用 PCLK 进行数据传输。

### 1. TX 模块

状态寄存器

- FIFO 状态
- MIPI 状态

配置寄存器

- 选择运行模式（流模式或帧模式）
- 固定通道 ID 模式
- 通道序号
- 清除产生的错误
- TxHOLD 状态定时器使能
- TxIDLE 状态定时器使能
- TxREQ 状态定时器使能

中断源寄存器

- FIFO 空
- 打破帧转换完成
- TxHOLD 状态超时
- TxIDLE 状态超时
- TxREQ 状态超时

中断屏蔽寄存器

软件复位寄存器

通道 ID 寄存器

数据寄存器

- Tx FIFO 输入
- Tx FIFO 尺寸：

▼ 32 位宽度×32 位深度（128 字节）

## 2.RX 模块

状态寄存器

- FIFO 状态
- MIPI 状态

配置寄存器 0

- 选择运行模式（流模式或帧模式）
- 固定通道 ID 模式
- 通道序号
- 清除产生的错误
- RxACK 状态定时器使能
- Rx 状态 定时器

配置寄存器 1

- Rx FIFO 清除
- Rx FIFO 定时器使能

中断源寄存器

- Rx FIFO 满
- 数据接收完成
- Rx FIFO 超时
- 接收的打破帧
- 打破帧接收错误
- RxACK 状态超时
- 丢失的时钟输入
- 增加的时钟输入

软件复位寄存器

通道 ID 寄存器

数据寄存器

- Rx FIFO 输入

●Rx FIFO 尺寸:

▼32 位宽度×64 位深度（128 字节）

## 28.2.模块图

### 1. 顶级模块图

Rx 模块部分基础架构与 Tx 模块部分相似

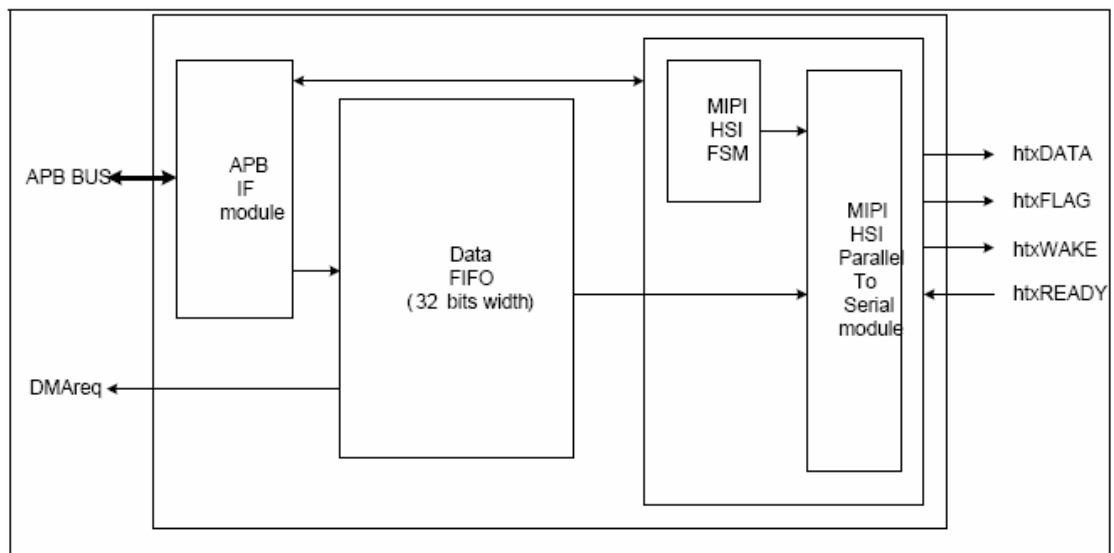


图 28-3 MIPI HIS 接口控制 Tx 模块顶层模块图

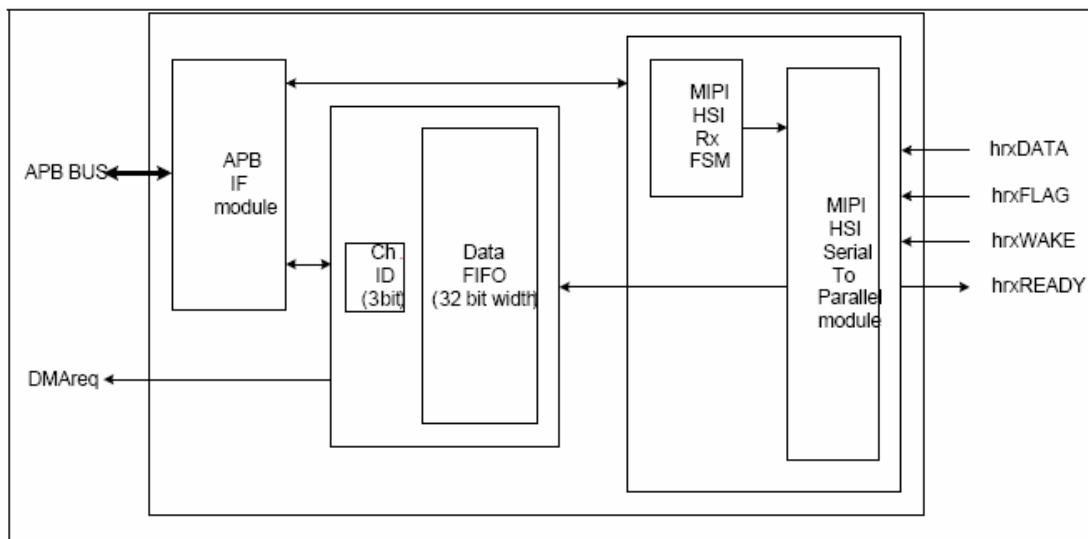


图 28-4 MIPI HIS 接口控制 Rx 模块顶层模块图

### T<sub>X</sub> 模块部分并行-串行块

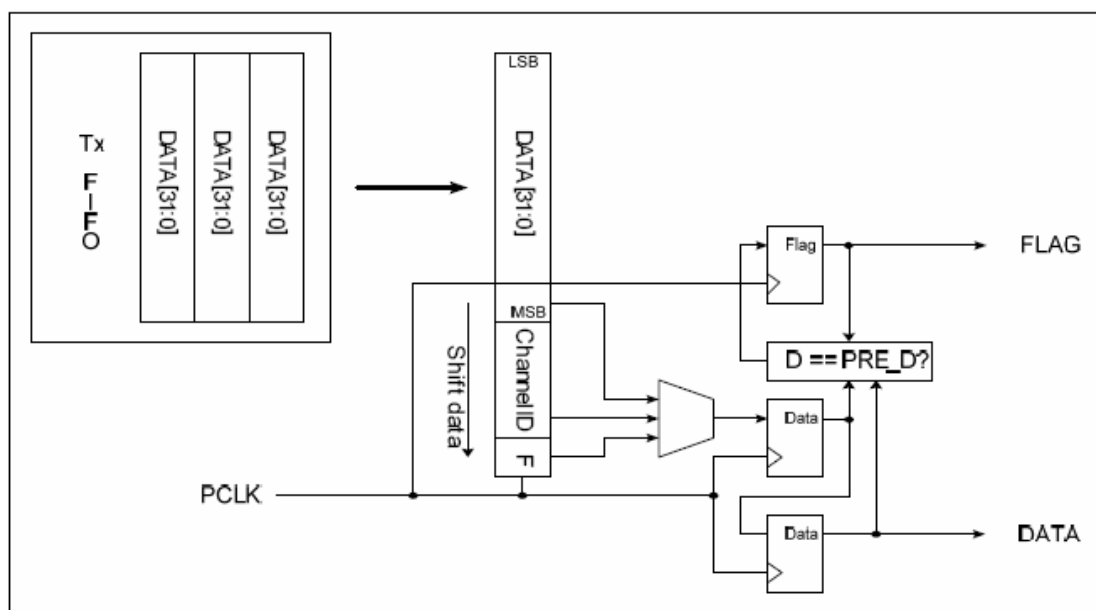


图 28-5 并行-串行块 (Tx 模块部分)

### R<sub>X</sub> 模块部分串行-并行块

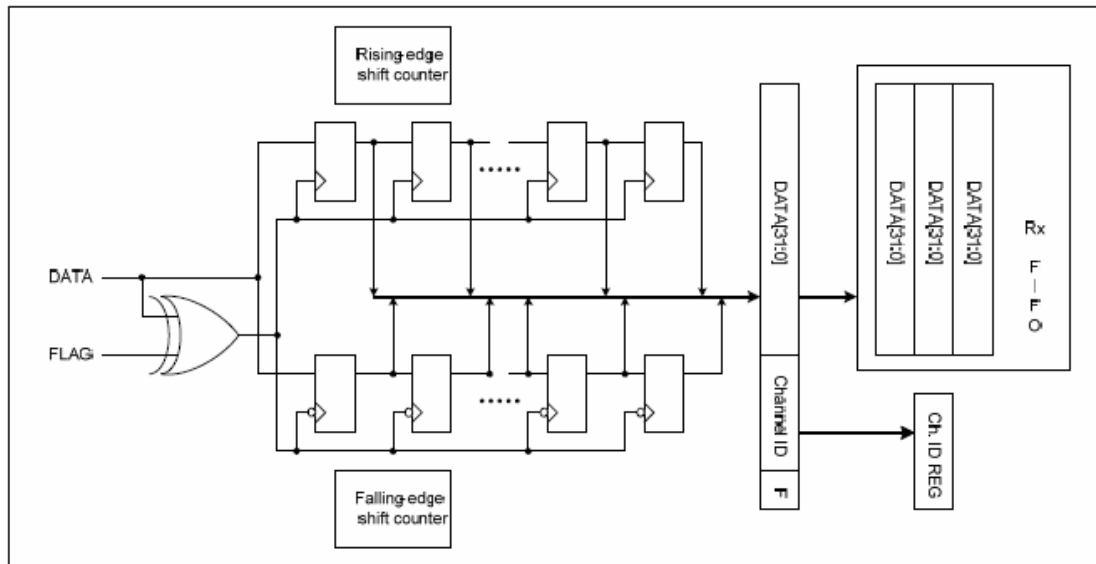


图 28-6 串行-并行块 (Rx 模块部分)

## 28.2 I/O 描述

## 1. TX 模块 I/O 列表

表 28-1 Tx I/O 描述

名称	#位	I/O	功能
MIPI HSI 接口信号 (Tx)			
TX_DATA	1	I	MIPI HSI 数据行
TX_FLAG	1	I	MIPI HSI 标志行
TX_WAKE	1	I	MIPI HSI 从其它部分 Tx 的唤醒行
TX_READY	1	0	MIPI HSI 向其他部分 Tx 的准备行

如果 DMA 请求使能位在中断和 DMA 请求屏蔽寄存器内使能，Tx 模块在 FIFO 为空闲状态时请求 DMA。

2.RX 模块 I/O 列表

表 28-2 Rx I/O 描述

名称	#位	I/O	功能
MIPI Hsi 接口信号 (Tx)			
RX_DATA	1	0	MIPI HSI 数据行
RX_FLAG	1	0	MIPI HSI 标志行
RX_WAKE	1	0	MIPI HSI 向其它部分 Rx 的唤醒行
RX_READY	1	I	MIPI HSI 从其他部分 Rx 的准备行

如果 DMA 请求使能位在中断和 DMA 请求屏蔽寄存器内使能，以及配置寄存器 0 内的 DMA 请求阈值位被设置为 (0x00~0x11)，Rx 模块在 Rx FIFO 数据数量大于阈值时，配置寄存器内的 Rx 模块将请求 DMA.

28.3 时序图

1.波形

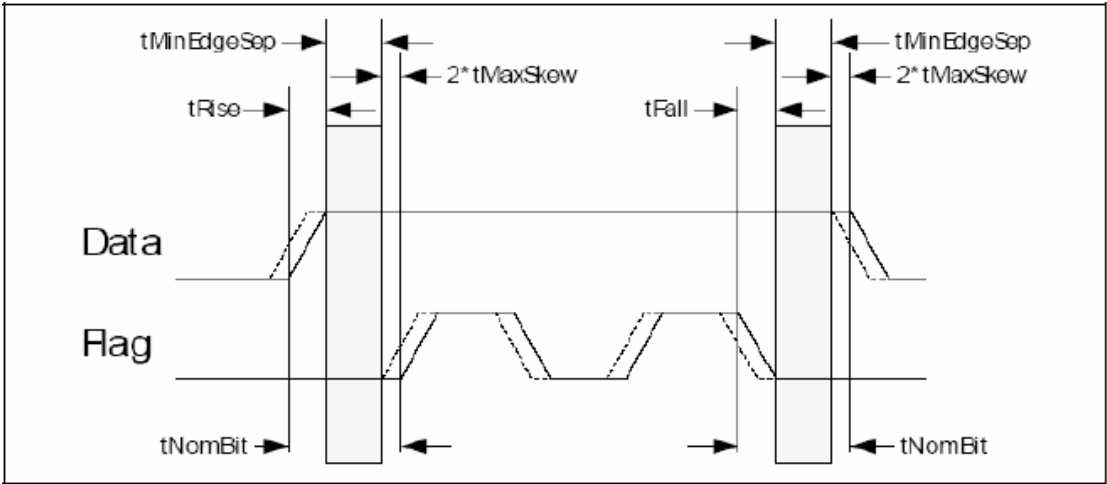


图 28-7 波形模块图

2. 信号时序

表 28-3 信号时序

参数	描述	1Mbit/s	100M bit/s
TNomBit	额定位时间	1000ns	10ns
TMinEdgeSep	DATA 和 FLAG 信号过度时间允许的最小间隔	500ns	5ns
TNomBit	允许的结合扭曲和抖动的最大时间。	249ns	1.5ns
tRise and tFall	最小允许信号上升和下降时间	2ns	2ns

3. 单个/突发通道 ID 模式

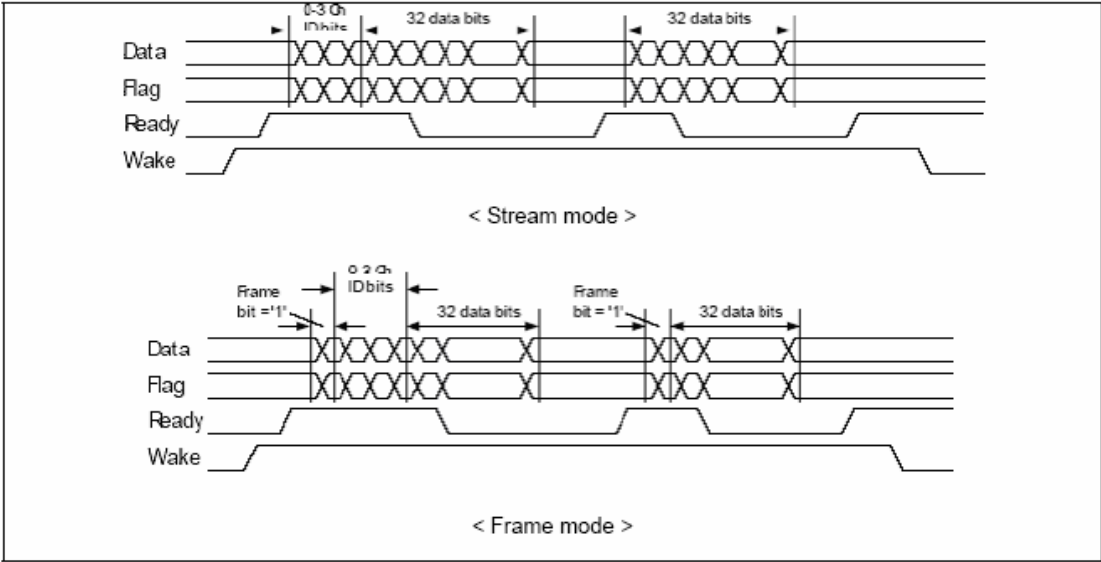


图 8-8 突发通道 ID 模式的例子模块图

在信号通道 ID 模式内，通道 ID 于每个数据的前端连在一起发送。在突发通道 ID 模式内，通道 ID 只于第一帧连在一起，并且在 IDLE 状态之后发送，只有 32 位数据可以发送到 IDLE 模式内。可变通道模式与固定通道 ID 模式相比带宽较宽，可以传送较大的数据，这主要是因为可以减少传输通道 ID 的数量。



#### 4. 流模式

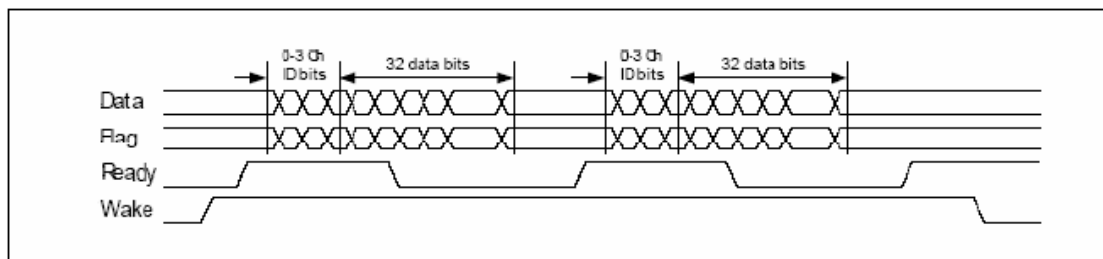


图 28-9 流模式例子模块图

#### 5. 帧模式

常规模式

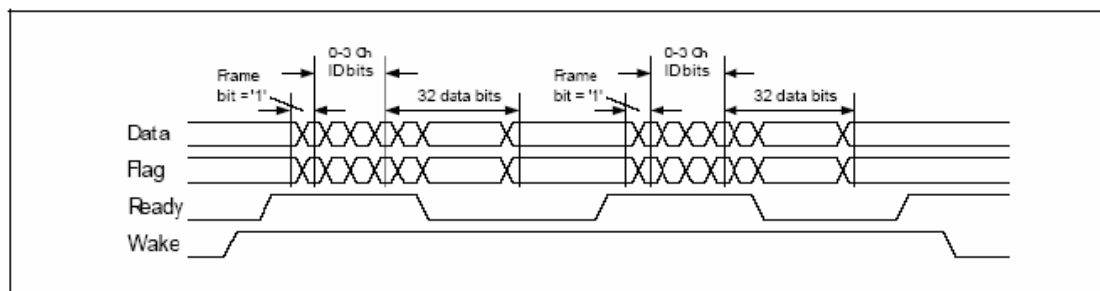
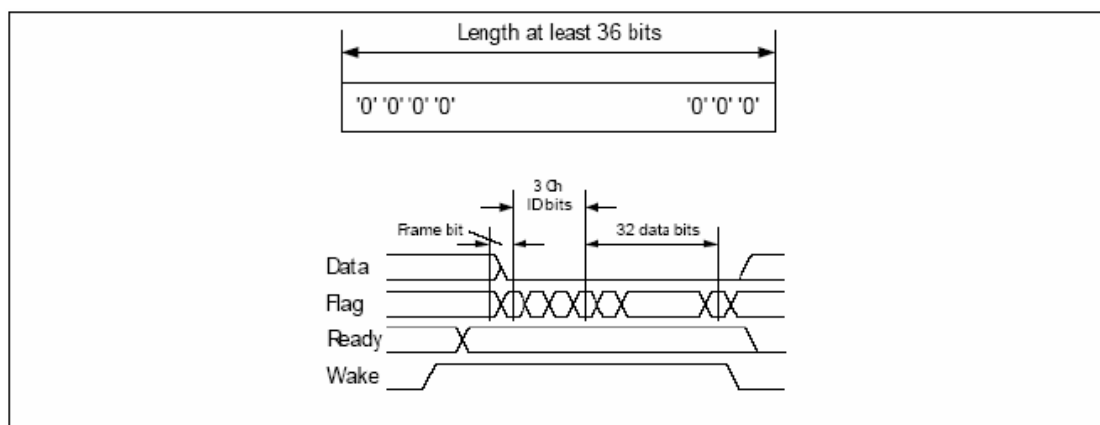


图 28-10 帧模式（常规模式）例子模块图

打破帧



28-11 打破帧模块图

标志信号不断的切换，直到转换结束为止。Tx 模块不镜像准备信号，而转换打破帧，这点与常规模式不同。因此，准备信号在上图中将不考虑。

# 28.4 功能描述

## 1. MIPI HIS Tx 控制器部分

有限状态机

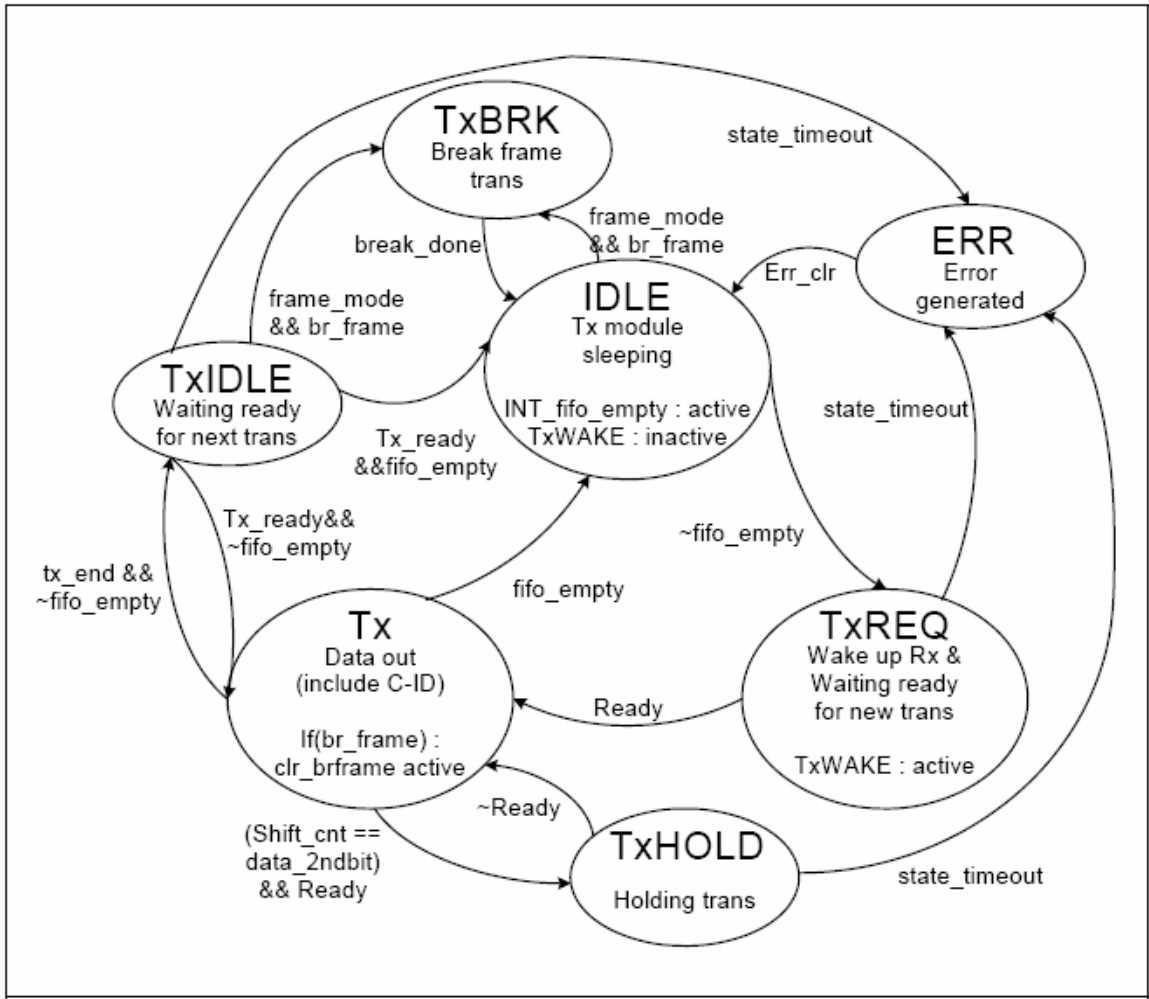


图 28-12 Tx 模块部分的 FSM 模块图

## 2. MIPI HIS Rx 控制器部分

原始状态机

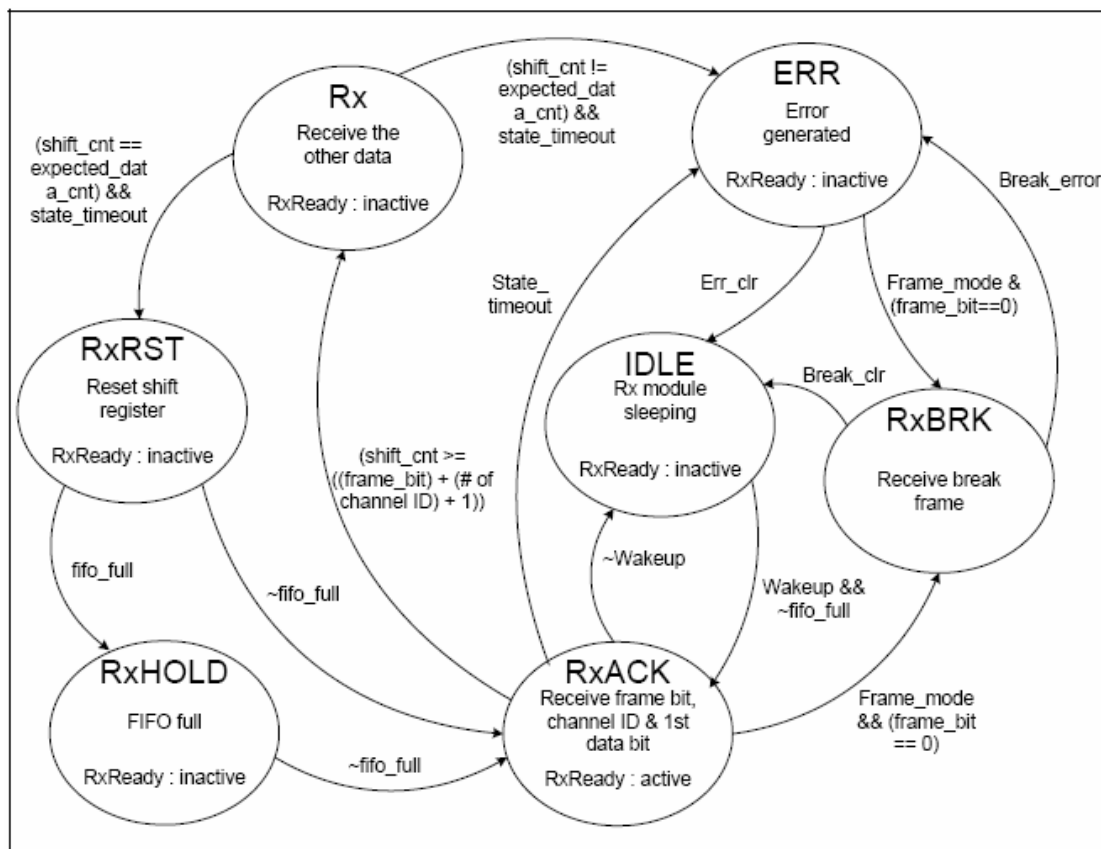


图 28-13 Rx 模块部分的 FSM

Rx 状态由定时器操作，可以设置为穿过 MIPI HIS 接口的时钟频率输入，操作模式以及通道 ID 的数量。例如，假设 MIPI 的时钟频率为 100MHz，通道 ID 数为 3，操作模式为帧模式，则 Rx 模块的操作时钟频率为 133MHz。如果一个数据帧是 10ns×36 周期，则需要 360ns 进行转换。设置 SFR 的值为 27-1，Rx 状态定时器可以在最佳状态下运行。FSM 的 RxRST 状态是用于准备下次操作的，可以通过复位转换寄存器应用于 MIPI HIS。

打破帧甚至可以在错误状态下被反馈。如果当时钟切换时的帧位为 0，这个状态变为 RxBREAK 状态。

# 28.5 特殊功能寄存器

寄存器映射

## 1.MIPI HIS Tx 控制器寄存器映射表

表 28-4 Tx 控制器寄存器映射表

寄存器	地址	描述	复位值
TX_STATUS_REG	0x7E006000	MIPI HIS Tx 控制器状态寄存器	0x00010000
TX_CONFIG_REG	0x7E006004	MIPI HIS Tx 控制器配置寄存器	0xFFFFFFFF02
Reserved	0x7E006008	保留的寄存器空间	0x00000000
TX_INTSRC_REG	0x7E00600C	MIPI HIS Tx 控制器中断源寄存器	0x00000000
TX_INTMSK_REG	0x7E006010	MIPI HIS Tx 控制器中断屏蔽寄存器	0x8000001F
TX_INTMSK_REG	0x7E006014	Tx 控制器软件复位	0x00000000
TX_CHID_REG	0x7E006018	MIPI HIS Tx 控制器通道 ID 寄存器	0x00000000
TX_DATA_REG	0x7E00601C	MIPI HIS Tx 控制器数据寄存器（FIFO 输入）	0x00000000

## 2. MIPI HIS Rx 控制器寄存器映射表

表 28-5 Rx 控制器寄存器映射表

寄存器	地址	描述	复位值
RX_STATUS_REG	0x7E007000	MIPI HIS Rx 控制器状态寄存器	0x00010000
RX_CONFIG0_REG	0x7E007004	MIPI HIS Rx 控制器配置寄存器	0x0FFFFFF02
RX_CONFIG1_REG	0x7E007008	MIPI HIS Rx 控制器配置寄存器	0x00FFFFFFF
RX_INTSRC_REG	0x7E00700C	MIPI HIS Rx 控制器中断源寄存器	0x00000000
RX_INTMSK_REG	0x7E007010	MIPI HIS Rx 控制器中断屏蔽寄存器	0x8000001F
RX_INTMSK_REG	0x7E007014	Rx 控制器软件复位	0x00000000
RX_CHID_REG	0x7E007018	MIPI HIS Rx 控制器通道 ID 寄存器	0x00000000
RX_DATA_REG	0x7E00701C	MIPI HIS Rx 控制器数据寄存器（FIFO 输出）	0x00000000

# 28.6 INDIVIDUAL 寄存器描述（TX 控制器）

## 1. TX\_STATUS\_REG

TX\_STATUS\_REG 是一个内部逻辑镜像窗口

表 28-6 TX\_STATUS\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
TX-STATUS_REG	0x7E00_6000	读	MIPI HIS Tx 控制器状态寄存器	0x00010000

位	名称	描述	读/写	复位值
[31]	Reserved	保留位	读	0x0
[30:28]	Next_state	下一个状态	读	0x0
[27]	Reserved	保留位	读	0x0
[26:24]	Current state	当前状态	读	0x0
[23:18]	Reserved	保留位	读	0x00
[17]	FIFO_full	FIFO 满 0: FIFO 不满 1: FIFO 满	读	0x0
[16]	FIFO_empty	FIFO 空 0: FIFO 不空 1: FIFO 空	读	0x1
[15:13]	Reserved	保留位	读	0x0
[12:8]	Tx_rd_point	TxFIFO 读指针	读	0x00
[7:5]	Reserved	保留位	读	0x0
[4:0]	Tx_wr_point	TxFIFO 写指针	读	0x00

状态寄存器值

- 000: IDLE
- 001:TxREQ
- 010:Tx
- 011: TxHOLD

100: TxIDLE

101: 保留状态

110: TxBRK

111: TxERR

2. TX\_CONFIG\_REG

CONFIG\_REG 用于设置 Tx 控制器的配置

表 28-7 TX\_CONFIG\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
TX-CONFIG_REG	0x7E00_6004	读/写	MIPI HIS Tx 控制器配置寄存器	0xFFFFFFFF02

位	名称	描述	读/写	复位值
[31:24]	TxHOLD time	TxHOLD 状态定时器设置值	读/写	0xFF
[23:16]	TxIDLE time	TxIDLE 状态定时器设置值	读/写	0xFF
[15:8]	TxREQ time	TxREQ 状态定时器设置值	读/写	0xFF
[7]	TxHOLD time_en	TxHOLD 状态定时器使能 0: 禁止     1: 使能	读/写	0x0
[6]	TxIDLE time_en	TxIDLE 状态定时器使能 0: 禁止     1: 使能	读/写	0x0
[5]	TxREQ time_en	TxREQ 状态定时器使能 0: 禁止     1: 使能	读/写	0x0
[4]	Err_clr	清除产生的错误 0: 保留     1: 清除	读/写	0x0
[3:2]	Width of CHID	通道 ID 的宽度	读/写	0x0
[1]	Burst_mode	固定通道 ID 模式 0: 突发通道 ID 模式 1: 单通道 ID 模式	读/写	0x1
[0]	Frame_mode	帧 模式 0: 流模式 1: 帧 模式	读/写	0x0

3. TX\_INTSRC\_REG

INTSRC\_REG 是中断源悬挂寄存器

表 28-8 TX\_INTSRC\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
TX_INTSRC_REG	0x7E00_600C	读/写	MIPI HIS Tx 控制器中断源寄存器	0x00000000

位	名称	描述	读/写	复位值
[31:5]	Reserved	保留位	读	0x0000001
[4]	TxH_timeout	TxHOLD 状态超时中断（设置 1 进行清除）	读/写	0x0
[3]	TxI_timeout	TxIDIE 状态超时中断（设置 1 进行清除）	读/写	0x0
[2]	TxR_timeout	TxREQ 状态超时中断（设置 1 进行清除）	读/写	0x0
[1]	Brframe_end	帧模式下的打破帧转换完成（设置 1 进行清除）	读/写	0x0
[0]	TxFIFO_empty	TxFIFO 空中断（设置 1 进行清除）	读/写	0x1

4. TX\_INTMSK\_REG

INTMSK\_REG 是中断屏蔽和 DMA 请求使能寄存器

表 28-9 TX\_INTMSK\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
TX_INTMSK_REG	0x7E00_6010	读/写	MIPI HIS Tx 控制器中断屏蔽寄存器	0x8000001F

位	名称	描述	读/写	复位值
[31]	DMA_req_en	DMA 请求信号使能 0：使能      1：禁止	读/写	0x1

[30:5]	Reserved	保留位	读	0x0000000
[4]	TxH_timeout_mak	TxHOLD 状态超时中断屏蔽 0: 不屏蔽 1: 屏蔽	读/写	0x1
[3]	TxI_timeout_mak	TxIDIE 状态超时中断屏蔽 0: 不屏蔽 1: 屏蔽	读/写	0x1
[2]	TxR_timeout_mak	TxREQ 状态超时中断屏蔽 0: 不屏蔽 1: 屏蔽	读/写	0x1
[1]	Brframe_end_mak	帧模式下的打破帧转换完成 中断屏蔽 0: 不屏蔽 1: 屏蔽	读/写	0x1
[0]	TxFIFO_empty_mak	TxFIFO 空中断屏蔽 0: 不屏蔽 1: 屏蔽	读/写	0x1

5. TX\_SWRST\_REG

SWRST\_REG 是软件复位

表 28-10 TX\_SWREST\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
TX_SWRST_REG	0x7E00_6014	读/写	Tx 控制器软件复位	0x00000000

位	名称	描述	读/写	复位值
[31:1]	Reserved	保留位	读/写	0x00000000
[0]	Sw_rst	软件复位 0: 设置 1: 复位	读/写	0x0

6. TX\_CHID\_REG

CHID\_REG 用于转换通道 ID

表 28-11 TX\_CHID\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
TX_CHID_REG	0x7E00_6018	读/写	MIPI HIS Tx 控制器通道 ID 寄存器	0x00000000



位	名称	描述	读/写	复位值
[31]	Break_frame	帧模式下的打破帧转换  在自动清除模式下，此位将被自动清除。在其他模式下，在 br_frame_clr 位设置 1 期间，TxDAT 发送 0。	读/写/C	0x00000000
[30]	Auto_clr	打破帧自动清除位  0：自动清除和 TxBRK 状态结束 1：自动清除不可用，和 TxBRK 状态继续	读/写	0x0
[29]	Br_frame_clr	停止打破帧继续转换	写	0x0
[28:3]	Reserved	保留位	读	0x00000000
[2:0]	CHID	通道 ID	读/写	0x0

注：为了发送数据，必须首先设置 TX\_CHID\_REG，然后向数据 FIFO 内推入其他转换数据。当通过 TxDATA 传送使，相同的通道 ID 将与数据相连接。如果通道 ID 与先前的不同，必须在向数据 FIFO 推入数据之前设置 TX\_CHID\_REG 的新的通道 ID。在帧模式下，当 1 被输入到 Break\_frame 位的时候，打破帧向 Rx 部分传输数据。在自动清除模式下，传输完成以后此位将自动清除。在这种情况下，内部状态进入 IDLE 状态。如果不在自动清除模式下，内部状态一直保持在 TxBRK 状态的情况下，TxDATA 继续转换 0，状态将冲 RxBRK 转变到 IDLE，同时 TxDATA 停止转换 0。

7. TX\_DATA\_REG

TX\_DATA\_REG 是 TxFIFO 输入

表 28-12 DATA\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
TX_DATA_REG	0x7E00_601C	读/写	MIPI HIS Tx 控制器数据寄存器	0x00000000

位	名称	描述	读/写	复位值
[31:0]	TxFIFO in	用于转换的 TxFIFO 数据输入	读/写	0x0

注：一旦数据下载到 TxFIFO 上，TxFIFO 内的数据将通过 MIPI HIS Tx 控制器被转换到其他部分的 RX，直到 TxFIFO 完全变空为止。

# 28.7 INDIVIDUAL 寄存器描述（RX 控制器）

## 1. RX\_STATUS\_REG

RX\_STATUS\_REG 是一个内部逻辑镜像窗口

表 28-13 RX\_STATUS\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
RX_STATUS_REG	0x7E00_7000	读	MIPI HIS Rx 控制器状态寄存器	0x00010000

位	名称	描述	读/写	复位值
[31]	Reserved	保留位	读	0x0
[30:28]	Next_state	下一个状态	读	0x0
[27]	Reserved	保留位	读	0x0
[26:24]	Curr_state	当前状态	读	0x0
[23:19]	Reserved	保留位	读	0x00
[18]	FIFO_timeout	RxFIFO 读超时 0: 及时 1: 超时	读	0x0
[17]	FIFO_full	FIFO 满 0: FIFO 不满 1: FIFO 满	读	0x0
[16]	FIFO_empty	FIFO 空 0: FIFO 不空 1: FIFO 空	读	0x1
[15:14]	Reserved	保留位	读	0x0
[13:8]	Rx_rd_point	RxFIFO 读指针	读	0x00
[7:6]	Reserved	保留位	读	0x0
[5:0]	Rx_wr_point	RxFIFO 写指针	读	0x00

状态寄存器值

000: IDLE	001:RxACK
010: Rx	011: RxHOLD
100: RxBREAK	101: 保留状态
110: RxRST	111: RxERR

2. RX\_CONFIG0\_REG

RX\_CONFIG0\_REG 用于设置 Rx 控制器的配置

表 28-14 RX\_CONFIG0\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
RX_CONFIG0_REG	0x7E00_7004	读/写	MIPI HIS Rx 控制器配置寄存器	0x0FFFFFF02

位	名称	描述	读/写	复位值
[31:30]	Reserved	保留位	读	0x0
[29:28]	DREQ_thres_val	DMA 请求阈值 当 FIFO 的有效数据为下列情况时，DMA 请求信号被激活 0x00: 满 0x01: 多于四个字 0x10: 多于 8 个字 0x11: 多于 16 个字	读/写	0x00
[27:16]	Rx_State time	Rx 状态定时器设置值	读/写	0xFFF
[15:8]	RxACK time	RxACK 状态定时器设置值	读/写	0xFF
[7]	Reserved	保留位	读/写	0x0
[6]	RxACK time_en	RxACK 状态定时器使能 0: 禁止     1: 使能	读/写	0x0
[5]	Break_clr	RxBREAK 状态清除位使能 0: 禁止     1: 使能	读/写	0x0
[4]	Err_clr	清除产生的错误	读/写	0x0

		0: 保留    1: 清除		
[3:2]	Width of CHID	通道 ID 的宽度	读/写	0x0
[1]	Burst_mode	固定通道 ID 模式 0: 突发通道 ID 模式 1: 单通道 ID 模式	读/写	0x1
[0]	Frame_mode	帧 模式 0: 流模式 1: 帧 模式	读/写	0x0

3. RX\_CONFIG1\_REG

RX\_CONFIG1\_REG 用于设置 Rx FIFO 的配置

表 28-15 RX\_CONFIG1\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
RX_CONFIG1_REG	0x7E00_7008	读/写	MIPI HIS Rx 控制器配置寄存器	0x00FFFFFF

位	名称	描述	读/写	复位值
[31]	RxFIFO_clr	打破帧接收定时器设置值	读/写	0x0
[30:28]	Reserved	保留位	读	0x0
[27]	RxFIFO_timer_en	RxFIFO 定时器使能	读/写	0x0
[26:24]	Reserved	保留位	读	0x0
[23:0]	RxFIFO_time	RxFIFO 定时器设置值	读/写	0xFFFFFFFF

4. RX\_INTSRC\_REG

INTSRC\_REG 是中断源悬挂寄存器

表 28-16 RX\_INTSRC\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
RX_INTSRC_REG	0x7E00_700C	读/写	MIPI HIS Rx 控制器中断源寄存器	0x00000000

位	名称	描述	读/写	复位值
[31:8]	Reserved	保留位	读	0x0000000
[7]	Break_done	帧模式下接收的打破帧（设置 1 进行清除）	读/写	0x0
[6]	Added_clock	增加的时钟输入（设置 1 进行清除）	读/写	0x0
[5]	Missed_clock	丢失的始终输入中断（设置 1 进行清除）	读/写	0x0
[4]	RxACK_timeout	RxACK 状态超时中断（设置 1 进行清除）	读/写	0x0
[3]	Brframe_err	接收的数据不是打破帧（设置 1 进行清除）	读/写	0x0
[2]	RxDONE	数据接收完成（设置 1 进行清除）	读/写	0x0
[1]	RxFIFO_timeout	RxFIFO 超时但是 RxFIFO 不为空（设置 1 进行清除）	读/写	0x0
[0]	RxFIFO_full	RxFIFO 满中断（设置 1 进行清除）	读/写	0x0

### 5. RX\_INTMSK\_REG

RX\_INTMSK\_REG 是中断屏蔽和 DMA 请求使能寄存器

表 28-17 RX\_INTMSK\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
RX_INTMSK_REG	0x7E00_7010	读/写	MIPI HIS Rx 控制器中断屏蔽寄存器	0x800001FF

位	名称	描述	读/写	复位值
[31]	DMA_req_en	DMA 请求信号使能 0：使能      1：禁止	读/写	0x1
[30:9]	Reserved	保留位	读	0x0000000

[8]	Wakeup_enn	MIPI 唤醒使能 0: 使能      1: 禁止	读/写	0x1
[7]	Break_done_mak	打破帧完成中断屏蔽 0: 不屏蔽    1: 屏蔽	读/写	0x1
[6]	Add_clock_mak	增加的时钟输入中断屏蔽 0: 不屏蔽    1: 屏蔽	读/写	0x1
[5]	Missed_clock_mak	丢失的时钟中断屏蔽 0: 不屏蔽    1: 屏蔽	读/写	0x1
[4]	RxACK_timeout_mak	RxACK 状态超时中断屏蔽 0: 不屏蔽    1: 屏蔽	读/写	0x1
[3]	Brframe_err_mak	打破帧错误中断屏蔽 0: 不屏蔽    1: 屏蔽	读/写	0x1
[2]	RxDONE_mak	RxDONE 中断屏蔽 0: 不屏蔽    1: 屏蔽	读/写	0x1
[1]	RxFIFO_timeout_mak	RxFIFO 超时中断屏蔽 0: 不屏蔽    1: 屏蔽	读/写	0x1
[0]	RxFIFO_full_mak	RxFIFO 满中断屏蔽 0: 不屏蔽    1: 屏蔽	读/写	0x1

## 6. RX\_SWRST\_REG

SWRST\_REG 是软件复位

表 28-18 RX\_SWREST\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
RX_SWRST_REG	0x7E00_7014	读/写	Rx 控制器软件复位	0x00000000

位	名称	描述	读/写	复位值
[31:1]	Reserved	保留位	读/写	0x00000000
[0]	Sw_rst	软件复位 0: 设置      1: 复位	读/写	0x0

7. RX\_CHID\_REG

CHID\_REG 用于转换通道 ID

表 28-19 RX\_CHID\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
RX_CHID_REG	0x7E00_7018	读/写	MIPI HIS Rx 控制器通道 ID 寄存器	0x00000000

位	名称	描述	读/写	复位值
[31:3]	Reserved	保留位	读	0x00000000
[2:0]	CURR_ID	当前通道 ID	读	0x0

注：因为 RxFIFO 内的数据有相同的通道 ID，在是 RxFIFO 为空以后改变通道 ID。

8. RX\_DATA\_REG

RX\_DATA\_REG 是 RxFIFO 输入

表 28-20 DATA\_REG 寄存器描述

寄存器	地址	读/写	描述	复位值
RX_DATA_REG	0x7E00_701C	读	MIPI HIS Rx 控制器数据寄存器 (FIFO 输出)	0x00000000

位	名称	描述	读/写	复位值
[31:0]	RxFIFO out	RxFIFO 数据输出	读	0x0

注：一旦数据下载到 RxFIFO 上，RxFIFO 内的数据将通过 MIPI HIS Rx 控制器被转换到其他部分的 RX，直到 RxFIFO 完全变空为止。

# 29 SPI 接口

串行外设接口（SPI）能进行串行数据传输。SPI 包括两个 8、16、32 位的移位寄存器，分别用于传输和接收。在 SPI 传输期间，数据同步传输（串行输出）和接收（串行输入）。

## 29.1 SPI 接口的特性

SPI支持下面特性：

- 全双工。
- 用于发送和接收的8/16/32位移位寄存器。
- 8位预分频逻辑。
- 三个时钟源。
- 8/16/32位总线接口。
- 两个独立的发送和接收FIFO。
- 主控制器模式和从属器模式。

## 29.2 信号描述

如表 29-1 所示，列出了 SPI 和外部设备之间的外部信号。当无效时，SPI 的所有端口可以作为通用 I/O 端口。

表 29-1 外部信号描述

名称	方向	描述
XspiCLK	输入输出	XspiCLK 是串行时钟，用于控制传输数据的时间。
XspiMISO	输入输出	在主控制器模式，该端口是输入端口。输入模式用于从从属器输出端口获得数据。数据通过从属器模式下的端口传输到主控制器。
XspiMOSI	输入输出	在主控制器模式，该端口是输出端口。该端口用于从主控制器输出端口传输数



		据。数据通过从属器模式下的端口被接收。
XspiCS	输入输出	从属器选择信号，当 XspiCS 是低电平时，所有数据发送/接收依次被执行。

## 29.3 SPI 操作

S3C6410 中，SPI在S3C6410和外部设备之间传输1位串行数据。S3C6410中的SPI支持CPU或DMA分别发送或接收FIFO，并且同时双向传输数据。SPI有两个通道，发送通道和接收通道。

CPU（或DMA）必须在SPI\_TX\_DATA寄存器写入数据以将数据写入到FIFO。寄存器的数据自动移动到发送FIFO。为了从接收FIFO读数据，CPU（或DMA）必须访问寄存器SPI\_RX\_DATA，并且数据被自动发送到寄存器SPI\_RX\_DATA。

### 1. 操作模式

HS\_SPI有两个模式，主控器模式和从属器模式。在主控器模式下，HS\_SPICLK产生，并被发送到外部设备。PSS选择从属器的信号，当它为低电平时，表示数据有效。在信息包开始发送和传输前PSS必须设置为低电平。

### 2. FIFO访问

S3C6410的SPI支持CPU和DMA访问FIFO。CPU访问和DMA访问FIFO的数据大小可以选择8位或者32位。如果选择8位数据大小，有效位为0到7位。通过触发用户定义的阈值，CPU访问正常打开和关闭。每个FIFO的触发器级别被设置从0~64字节。SPI\_MODE\_CFG 寄存器的TxDMA0n或RxDMA0n位必须设置DMA访问。DMA访问仅仅支持单传输和四个脉冲传输。在发送FIFO时，DMA要求信号为高电平直到FIFO满。在接收FIFO时，如果FIFO 非空，DMA要求信号是高电平。

### 3. 在接收FIFO中的结尾字节

在中断模式下，接收FIFO中采样的数量小于阈值的值，或者是DMA 四个脉冲模式，并且没有其它数据被接收，该保留字节被称做结尾字节。为了在接收FIFO中移动这些字节，需要用到内部时钟和中断信号。基于APB总线时钟，内部时钟的值能被设置到1024时钟。当定时器值为0时，中断信号发生，并且CPU能在FIFO中移动结尾字节。

### 4. 信息包数目控制

在主控器模式下，SPI可以控制接收的信息包的数量。如果有信息包要被接收，则设置SFR(Packet\_Count\_reg)有多少包要接收。当包的数量和设置的数量相同时，SPI 停止产生SPICLK。在该功能被再次装载前，它严格遵循软件和硬件复位（软件复位能清除了特殊功能的寄存器外的所有的寄存

器，但是硬件复位清除所有的寄存器）。

5. NCS控制

nCS可以选择自动控制和手动控制。对于手动控制，Auto\_n\_Manual必须设置为默认值0。nCS电平设置和设置的nSSout位相同。自动控制下，nCS能被固定在包与包之间。Auto\_n\_Manual设置为1，只要nCS不活动，nCS\_time\_count必须设置。这时nSSout是可用的。

6. SPI传输格式

S3C6410支持4种不同的格式来传输数据。如图29-1所示，描述了SPICLK的4种波形。

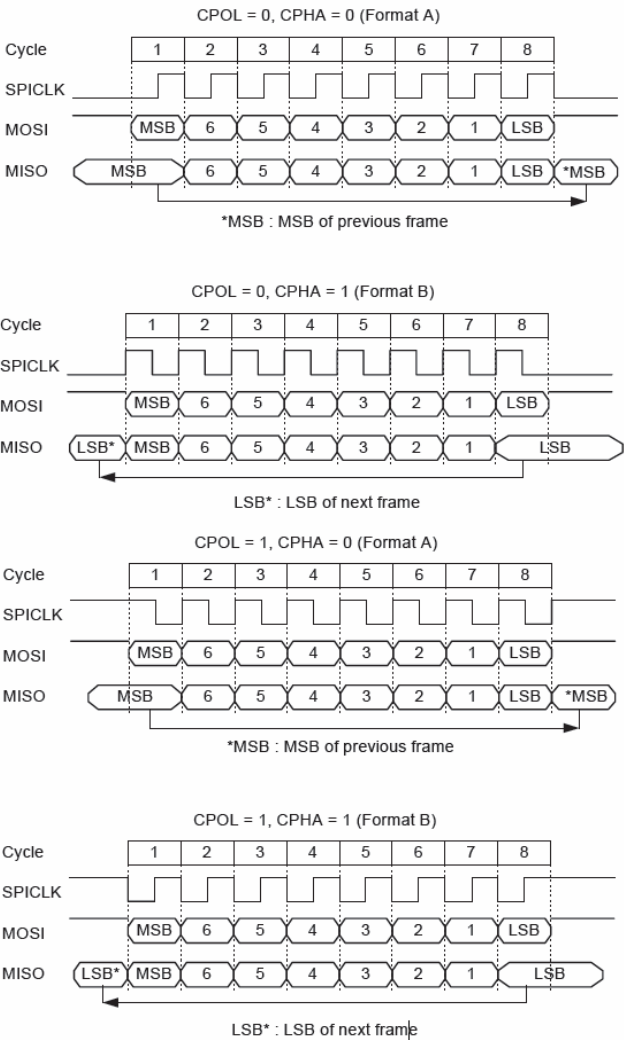


图 29-1 SPI 传输格式

# 29.4 特殊功能的寄存器描述

特殊功能的寄存器必须以下面的顺序设置 (nCS 为手动模式)：

- (1) 设置传输类型 (CPOL 或 CPHA 设置)。
  - (2) 设置时钟设置寄存器。
  - (3) 设置 SPI MODE 设置寄存器。
  - (4) 设置 API INT\_EN 寄存器。
  - (5) 如果需要的话，设置信息包数量配置寄存器。
  - (6) 设置发送和接收通道启动。
  - (7) 设置 nSSout 为低位来开始发送或接收操作。
- 设置nSSout为低位，那么启动发送数据写。
  - 如果自动芯片选择位被设置，则不应当控制nCS。

## 特殊功能寄存器

寄存器	地址	读/写	描述	初始值
CH_CFG (Ch0)	0x7F00B000	读/写	SPI配置寄存器。	0x0
CH_CFG (Ch1)	0x7F00C000	读/写	SPI配置寄存器。	0x0

CH_CFG	位	读/写	描述	初始状态
SW_RST	[5]	读/写	软复位。 0:不活动 1: 活动	1' b0
SLAVE	[4]	读/写	SPI通道是主控器还是从属器。 0: 主控器1: 从属器	1' b0
CPOL	[3]	读/写	确定一个有效的高位或者低位时钟。 0: 有效的高位 1:有效的低位	1' b0
CPHA	[2]	读/写	从两个基本的不同传输格式中选择一个。 0: 格式A 1: 格式 B	1' b0
RxCh0n	[1]	读/写	SPI 接收通道打开。 0: 通道关闭 1: 通道打开	1' b0

TxCh0n	[0]	读/写	SPI 发送通道打开。 0: 通道关闭1: 通道打开	1' b0
--------	-----	-----	-------------------------------	-------

寄存器	地址	读/写	描述	初始值
Clk_CFG (Ch0)	0x7F00B004	读/写	时钟配置寄存器。	0x0
Clk_CFG (Ch1)	0x7F00C004	读/写	时钟配置寄存器。	0x0

Clk_CFG	位	读/写	描述	初始状态
ClkSel	[10:9]	读/写	选择时钟源来产生SPI 时钟输出。 00 : PCLK 01 : USBCLK 10 : Ep11 时钟 11 : 保留 *对于使用 USBCLK 源, USB_SIG_MASK 在系统控制器必须设置为打开。	2' b0
ENCLK	[8]	读/写	时钟开/关。 0 : 无效 1 : 有效	1' b0
Prescaler Value	[7:0]	读/写	SPI 时钟输出分频频率。 SPI 时钟输出 = 时钟源 / ( 2 × (预分频值 +1))	8' h0

寄存器	地址	读/写	描述	初始值
MODE_CFG (Ch0)	0x7F00B008	读/写	SPI FIFO控制寄存器。	0x0
MODE_CFG (Ch1)	0x7F00C008	读/写	SPI FIFO控制寄存器。	0x0

MODE_CFG	位	读/写	描述	初始状态
Ch_tran_size	[30:29]	读/写	00 : 字节 01 : 半字 10 : 字 11 : 保留	2' b0
Trailing Count	[28:19]	读/写	计数值, 从接收FIFO中写入的最后数据到覆盖FIFO中结尾的字节。	10' b0
BUS transfer size	[18:17]	读/写	00: 字节 01: 半字	2' b0

			10 :字 11:保留	
RxTrigger	[16:11]	读/写	在中断模式，接收FIFO触发电平从6' h0~6' h40。该值是接收FIFO的字节数。	6' b0
TxTrigger	[10:5]	读/写	在中断模式，发送FIFO触发电平从6' h0~6' h40。该值是发送FIFO的字节数。	6' b0
reserved	[4:3]	-	-	-
RxDMA On	[2]	读/写	DMA模式打开/关闭。 0 : DMA 模式关闭 1 : DMA 模式打开	1' b0
TxDMA On	[1]	读/写	DMA模式打开/关闭 0 : DMA 模式关闭 1 : DMA 模式打开	1' b0
DMA transfer	[0]	读/写	DMA 传输类型，单个或者四个脉冲。 0 : 单个 1 : 4 个脉冲 设置DMA传输大小必须和SPI中一样。	1' b0

通道传输大小应当小于或等于总线传输大小。

寄存器	地址	读/写	描述	初始值
Slave_slection_reg(Ch0)	0x7F00B00C	读/写	从属器选择寄存器。	0x1
Slave_slection_reg(Ch1)	0x7F00C00C	读/写	从属器选择寄存器。	0x1

Slave_slection_reg	位	读/写	描述	初始状态
nCS_time_count	[9:4]	读/写	nSSout 无效时间 = ((nCS时间计数+3)/2) × SPICLKout)	6' b0
reserved	[3:2]		保留。	
Auto_n_Manual	[1]	读/写	芯片选择设置分手动或自动选择。 0: 手动 1: 自动	1' b0
nSSout	[0]	读/写	从属器选择信号（只用于手动） 0: 有效 1: 无效	1' b1

寄存器	地址	读/写	描述	初始值
SPI_INT_EN(Ch0)	0x7F00B010	读/写	SPI中断启动寄存器。	0x0
SPI_INT_EN(Ch1)	0x7F00C010	读/写	SPI中断启动寄存器。	0x0

SPI_INT_EN	位	读/写	描述	初始状态
IntEnTrailing	[6]	读/写	用于结尾计数设置为0的中断启动。 0: 无效 1:有效	1' b0
IntEnRxOverrun	[5]	读/写	用于接收超限运行的中断启动。 0: 无效 1:有效	1' b0
IntEnRxUnderrun	[4]	读/写	用于接收欠载运行的中断启动。 0: 无效 1:有效	1' b0
IntEnTxOverrun	[3]	读/写	用于发送超限运行的中断启动。 0: 无效1:有效	1' b0
IntEnTxUnderrun	[2]	读/写	用于发送欠载运行的中断启动。在从属器模式， 转向从属器发送路径后，该位必须首先清除。 0:无效 1:有效	1' b0
IntEnRxFifoRdy	[1]	读/写	用于 RxFifoRdy(中断模式)的中断启动。 0: 无效 1:有效	1' b0
IntEnTxFifoRdy	[0]	读/写	用于TxFifoRdy(中断模式)的中断启动。 0: 无效 1:有效	1' b0

寄存器	地址	读/写	描述	初始值
SPI_STATUS(Ch0)	0x7F00B014	读	SPI状态寄存器。	0x0
SPI_STATUS(Ch1)	0x7F00C014	读	SPI状态寄存器。	0x0

SPI_STATUS	位	读/写	描述	初始状态
TX_done	[21]	读	表示传输完成。 0：所有情况除了熔断情况	1' b0

			1：发送FIFO和移位寄存器为空时	
Trailing_byte	[20]	读	表示结尾计数是0。	1' b0
RxFifoLvl	[19:13]	读	数据水平在接收 FIFO。 0~7' h40 字节	7' b0
TxFifoLvl	[12:6]	读	数据水平在发送 FIFO。 0~7' h40字节	7' b0
RxOverrun	[5]	读	接收FIFO超限错误。 0: 无错误, 1: 错误	1' b0
RxUnderrun	[4]	读	接收 FIFO欠载运行错误。 0: 无错误, 1: 错误	1' b0
TxOverrun	[3]	读	发送 FIFO 超限错误。 0: 无错误r, 1: 错误	1' b0
TxUnderrun	[2]	读	发送 FIFO欠载运行错误。	1' b0
RxFifoRdy	[1]	读	0：FIFO中数据少于触发器电平。 1：FIFO中数据多于触发器电平。	1' b0
TxFifoRdy	[0]	读	0：FIFO中数据多于触发器电平。 1：FIFO中数据少于触发器电平。	1' b0

寄存器	地址	读/写	描述	初始值
SPI_TX_DATA (Ch0)	0x7F00B018	写	SPI发送数据寄存器。	0x0
SPI_TX_DATA (Ch1)	0x7F00C018	写	SPI发送数据寄存器。	0x0

SPI_TX_DATA	位	读/写	描述	初始状态
TX_DATA	[31:0]	写	该区域包含要通过SPI通道发送的数据。	32' b0

寄存器	地址	读/写	描述	初始值
SPI_RX_DATA (Ch0)	0x7F00B01C	读	SPI 接收数据寄存器。	0x0
SPI_RX_DATA (Ch1)	0x7F00C01C	读	SPI 接收数据寄存器。	0x0

SPI_RX_DATA	位	读/写	描述	初始状态
RX_DATA	[31:0]	读	该区域包含通过SPI通道被接收到的数据。	32' b0

寄存器	地址	读/写	描述	初始值
Packet_Count_reg (Ch0)	0x7F00B020	读/写	计数，主控器收到多少数据。	0x0
Packet_Count_reg (Ch1)	0x7F00C020	读/写	计数，主控器收到多少数据。	0x0

Packet_Count_reg	位	读/写	描述	初始状态
Packet_Count_En	[16]	读/写	启动位，用于信息包计数。 0：无效 1：有效	1' b0
Count Value	[15:0]	读/写	包计数值。	16' b0

寄存器	地址	读/写	描述	初始值
SWAP_CFG (Ch0)	0x7F00B028	读/写	交换配置寄存器。	0x0
SWAP_CFG (Ch1)	0x7F00C028	读/写	交换配置寄存器。	0x0

SWAP_CFG	位	读/写	描述	初始状态
RX_Half-word swap	[7]	读/写	0：关闭 1：交换	1' b0
RX_Byte swap	[6]	读/写	0：关闭 1：交换	1' b0
RX_Bit swap	[5]	读/写	0：关闭 1：交换	1' b0
RX_SWAP_en	[4]	读/写	交换启动。 0：正常 1：交换	1' b0
TX_Half-word swap	[3]	读/写	0：关闭 1：交换	1' b0
TX_Byte swap	[2]	读/写	0：关闭 1：交换	1' b0
TX_Bit swap	[1]	读/写	0：关闭 1：交换	1' b0



TX_SWAP_en	[0]	读/写	交换启动。 0：正常 1：交换	1' b0
------------	-----	-----	--------------------	-------

寄存器	地址	读/写	描述	初始值
FB_Clk_sel (Ch0)	0x7F00B02C	读/写	反馈时钟选择寄存器。	0x3
FB_Clk_sel (Ch1)	0x7F00C02C	读/写	反馈时钟选择寄存器。	0x3

SWAP_CFG	位	读/写	描述	初始状态
SPICLKout delay	[2]	读/写	0：没有额外延迟 1：2.7ns 延迟（基于典型）	1' b0
FB_Clk_sel	[1:0]	读/写	00：0ns额外延迟 01：3ns额外延迟 10：6ns额外延迟 11：9ns额外延迟 *延迟基于典型情况。	2' b3

29.5 SPI 接口应用举例

本小节主要介绍 SPI 在 ARM11 处理器中的编程实现，结合以上对 SPI 接口的理解，相信读者很容易地掌握 SPI 接口的功能及特性。

以下是 SPI 部分代码的具体实现：

1. SPI 复位：功能是复位某一个 SPI 通道。

输入：SPI\_channel

输出：NONE.

```
void SPI_reset( SPI_channel * ch ) {
// 带有时钟延迟的复位
    Outp32( &ch->m_cBase->ch_cfg, Inp32(&ch->m_cBase->ch_cfg) & ~(0x3F) ); // 清除寄存器
    Outp32( &ch->m_cBase->ch_cfg, Inp32(&ch->m_cBase->ch_cfg) | (1<<5) );
    Delay(10);
}
```

// 释放复位信号

```
Outp32( &ch->m_cBase->ch_cfg, Inp32(&ch->m_cBase->ch_cfg) & ~(1<<5) );
```

## 2. SPI 通道初始化：功能是初始化某一个 SPI 通道。

输入：SPI\_channel

输出：NONE.

```
SPI_channel* SPI_channel_Init( int channel ) {
    SPI_channel* ch = &SPI_current_channel[channel];
    memset ( (void*)ch, 0, sizeof(SPI_channel) );
    ch->m_ucChannelNum = channel;
    if ( channel == 0 ) {
        ch->m_cBase = (SPI_SFR*)SPI0_BASE;
        ch->m_ucIntNum = NUM_SPI0;
        ch->m_fDMA= SPI_DMADoneChannel0;
        ch->m_fISR = SPI_interruptChannel0;
#ifdef SPI_NORMAL_DMA
        ch->m_ucDMACon = DMA0;
        SYSC_SelectDMA( eSEL_SPI0_TX, 1 ); // 标准的 DMA 设置
        SYSC_SelectDMA( eSEL_SPI0_RX, 1 ); // 标准的 DMA 设置
#else
        ch->m_ucDMACon = SDMA0;
        SYSC_SelectDMA( eSEL_SPI0_TX, 0 ); // 安全的 DMA 设置
        SYSC_SelectDMA( eSEL_SPI0_RX, 0 ); // 安全的 DMA 设置
#endif
    }
    else if ( channel == 1 ) {
        ch->m_cBase = (SPI_SFR*)SPI1_BASE;
        ch->m_ucIntNum = NUM_SPI1;
        ch->m_fDMA = SPI_DMADoneChannel1;
```

```

        ch->m_fISR = SPI_interruptChannel1;
#ifdef SPI_NORMAL_DMA
        ch->m_ucDMACon = DMA1;
        SYSC_SelectDMA( eSEL_SPI1_TX, 1 ); // 标准的 DMA 设置
        SYSC_SelectDMA( eSEL_SPI1_RX, 1 ); // 标准的 DMA 设置
#else
        ch->m_ucDMACon = SDMA1;
        SYSC_SelectDMA( eSEL_SPI1_TX, 0 ); // 安全的 DMA 设置.
        SYSC_SelectDMA( eSEL_SPI1_RX, 0 ); // 安全的 DMA 设置.
#endif
    }
    else {
        Assert(0);
    }

    Outp32(&ch->m_cBase->slave_sel, Inp32(&ch->m_cBase->slave_sel) | (1<<0) ); // 片选 OFF – 激活 LOW.

    SPI_GPIOPortSet(channel); // 设置通道 GPIO.
    return ch;
}

```

### 3. SPI 基本寄存器的设置

输入：SPI\_channel

输出：NONE

```

void SPI_setBasicRegister( SPI_channel* ch ) {
    Outp32( &ch->m_cBase->ch_cfg, //清除寄存器
    (ch->m_eClockMode<<4)| // 主/从模式
    (ch->m_eCPOL<<3)| // CPOL 高态有效/行
    (ch->m_eCPHA<<2) ); // CPHA 传输格式

    Outp32( &ch->m_cBase->clk_cfg, (Inp32(&ch->m_cBase->clk_cfg) & ~(0x7ff))|

```

(ch->m_eClockSource<<9)	//清除寄存器
(( (ch->m_eClockMode==SPI_MASTER)?(1):(0) )<<8)	//时钟设置
ch->m_cPrescaler);	//时钟使能
	// 预定标器设置
Outp32( &ch->m_cBase->mode_cfg, (Inp32(&ch->m_cBase->mode_cfg)&(u32)(1<<31))	
	//清除寄存器
(ch->m_eChSize<<29)	//通道传输大小
(ch->m_uTrailingCnt<<19)	// trailing 计数
(ch->m_eBusSize<<17)	// 总线传输大小
(ch->m_ucRxLevel<<11)	// Rx 触发级
(ch->m_ucTxLevel<<5)	// Tx 触发级
(ch->m_eDMAType<<0) );	// DMA 类型
}	

## 30 IIC 总线接口

这一节主要讲述S3C6410 RISC中IIC总线接口的功能和使用方法。

### 30.1 IIC 总线接口概述

S3C6410 RISC处理器能支持一个多主控器IIC串行接口。一个专用的串行数据线（SDA）和一个串行时钟线（SCL）在总线主控器和连接到IIC总线的外部设备之间传输数据。SDA和SCL线是双向的。

在多主控制IIC总线模式下，多个S3C6410 RISC处理器能发送（或接收）串行数据到从属设备。主控器S3C6410能开始和结束IIC总线上的数据传输。在S3C6410中IIC总线使用标准的总线仲裁程序。

为了控制多个IIC总线操作，必须将值写入下面的寄存器：

- 多主控器IIC总线控制寄存器，IICCON；
- 多主控器IIC总线控制/状态寄存器，IICSTAT；
- 多主控器IIC总线发送/接收数据移位寄存器，IICDS；
- 读主控器IIC总线地址寄存器，IICADD。

当IIC总线空闲，SDA和SCL线必须是高电平。SDA从高到低转换能启动一个开始条件。当SCL处于高电平，保持稳定时，SDA从低位到高位传输能启动一个停止条件。

主设备能一直产生开始和停止条件。开始条件产生后，主控器通过在第一次输出的数据字节中写入7位的地址来选择从属器设备。第8位用于确定传输方向（读或写）。

在到SDA总线上的每一个数据字节总数上必须是8位。在总线传输操作期间，发送或接收字节没有限制。数据一直是先从最高有效位（MSB）发送，并且每个字节后面必须立即跟随确认（ACK）位。

IIC总线模块图，如图30-1所示。

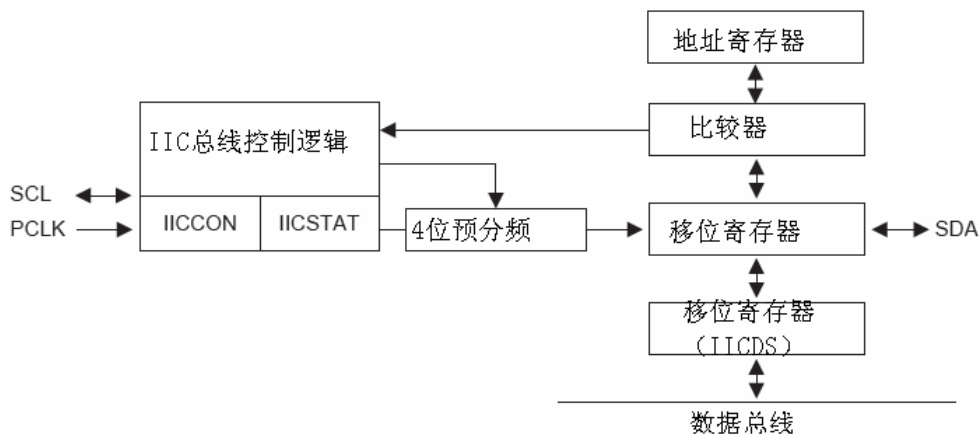


图 30-1 IIC 总线模块图

## 30.2 IIC 总线接口操作模式

S3C6410 IIC总线接口有四种操作模式：

- 主控器发送模式；
- 主控器接收模式；
- 从属器发送模式；
- 从属器接收模式。

这些操作模式之间的功能关系描述如下：

### 1. 开始和停止条件

IIC总线接口无效时，它通常是在从属器模式下。换句话说就是，在SDA线检测一个开始条件前（当时钟信号SCL在高位时，SDA线发生高位到低位的跃变，开始条件启动），接口必须在从属器模式下。当接口状态变为主控器模式时，在SDA线上的数据传输开始，并且产生SCL信号。

开始条件能通过SDA线传输一个字节的串行数据。一个停止条件能结束该数据传输。由主控器能一直产生开始和停止条件。当一个开始条件产生后，IIC总线获得繁忙信号。停止条件将使IIC总线空闲。

当主控器发起一个开始条件，它将发送一个从属地址来通知从属器设备。一个字节的地址域包含7位地址和1位传输方向指示器（表示写或读）。如果位8是0，表示写操作（发送操作）；如果位8是1，表示请求读取数据（接收操作）。

主控器通过发出一个停止条件来完成传输操作。如果主控器想继续将数据发送到主线，它将产生另一个开始条件和一个从属地址。通过这种方式，读写操作能在不同的格式下被执行。

开始和停止条件模块图，如图30-2所示。

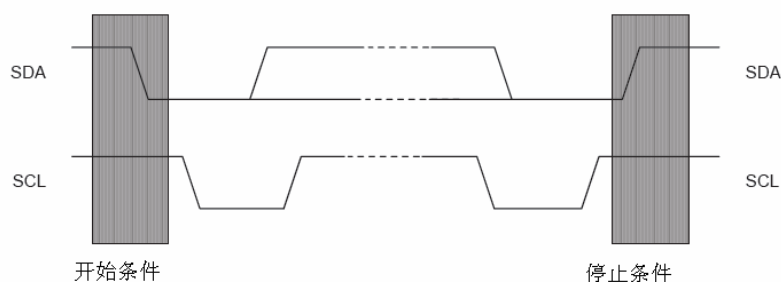


图 30-2 开始和停止条件模块图

## 2. 数据传输格式

在 SDA 线上的每一个字节长度必须是 8 位。起始条件后的第一个字节有一个地址域。当 IIC 主线在主控器模式下操作时，地址域能通过主控器被传输。每一个字节后面跟随一个 ACK（acknowledgement）位。MSB 位始终首先发送。IIC 总线数据传输的模块图，如图 30-3 所示。

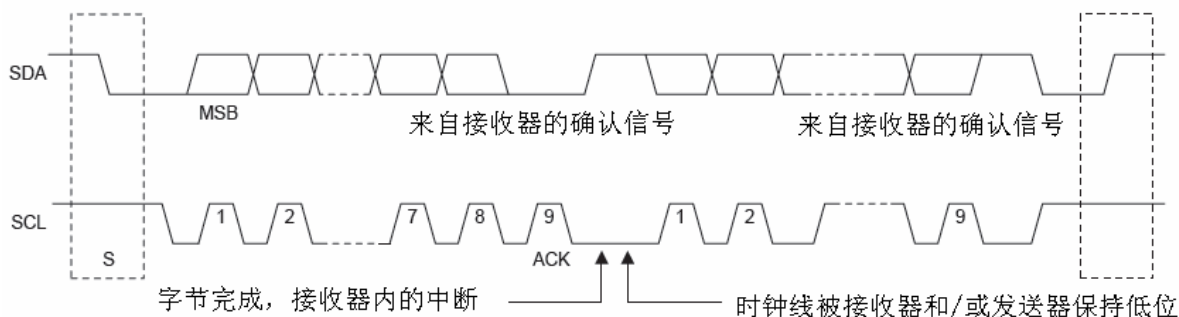


图 30-3 IIC 总线数据传输的模块图

## 3. ACK 信号传输

为了完成一个字节的发送操作，接收器必须将一个 ACK 位发送到发送器。ACK 脉冲在 SCL 线的第九个时钟产生。对于发送一个字节来说，八个时钟是必要的。主控器将产生一个时钟脉冲来发送一个 ACK 位。

当 ACK 时钟脉冲被接收时，通过使 SDA 置高位，发送器释放 SDA 线。在传送 ACK 时钟脉冲期间，接收器驱使 SDA 线置低位，以使 SDA 线在第九个 SCL 脉冲的高位时期保持低位。

ACK 位传输功能能通过软件（IICSTAT）来激活或者禁止。然而，在 SCL 的第九个时钟，ACK 脉冲被要求

来完成一个字节的数据传输操作。

IIC总线上的确认模块图，如图30-4所示。

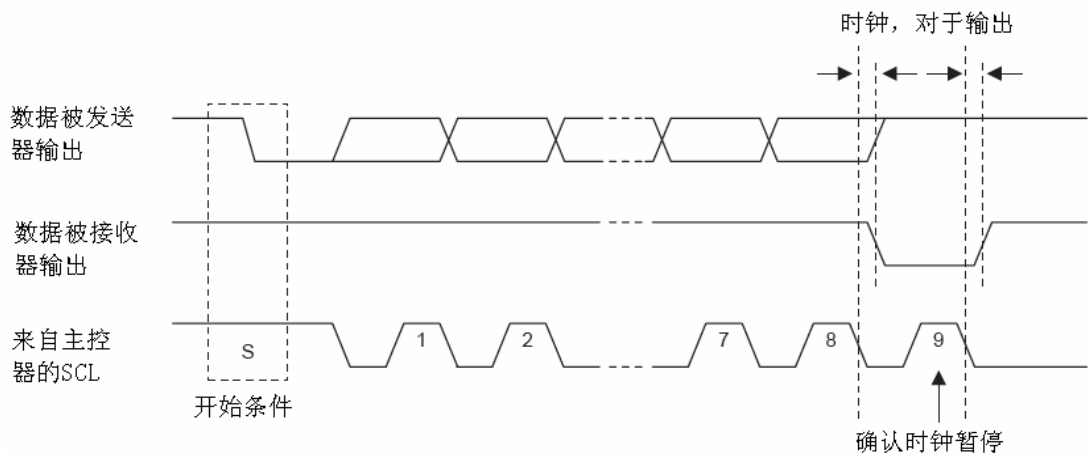


图 30-4 IIC 总线上的确认模块图

#### 4. 读写操作

在发送模式下，当发送数据时，IIC总线接口将一直等待，直到数据移位（IICDS）寄存器接收到一个新的数据。新的数据写入寄存器之前，SCL线将被保持在低位，数据写入后释放。S3C6410保持中断来确定当前数据发送完成。CPU接收中断请求后，它将新的数据写入到IICDS寄存器。

在接收模式下，当接收数据时，IICDS寄存器被读取前，IIC总线接口将一直等待。在新的数据被读出前，SCL线将保持低位，读取后释放。S3C6410保持中断来确认新的数据接收完成。CPU接收到中断请求后，它从IICDS寄存器读取数据。

#### 5. 异常中断条件

如果一个从属接收器不承认该从属地址，它将保持SDA线为高位。在这种情况下，主控器产生一个中断条件中断传输。

中断传输和主控器的接收器是有关的。来自从属器的最后数据字节被接收后，通过取消一个ACK的产生，通知从属发送器操作结束。从属发送器释放SDA来允许主控器产生一个停止条件。

#### 6. IIC 总线配置

为了控制串行时钟的频率（SCL），在IICCON寄存器中，4位的预分频值被执行。IIC总线接口地址被存储在IIC总线地址（IICADD）寄存器。由于默认，IIC总线地址有一个未知值。

#### 7. 每个模块的操作流程图

在IIC发送/接收操作前必须执行下面的步骤：



- (1) 如果需要的话，在IICADD寄存器写入自己的从属器地址；
- (2) 设置IICCON寄存器；
  - 启动中断
  - 定义SCL周期
- (3) 设置IICSTAT以能够连续输出。

主控器/发送器操作模式的流程图，如图 30-5 所示。

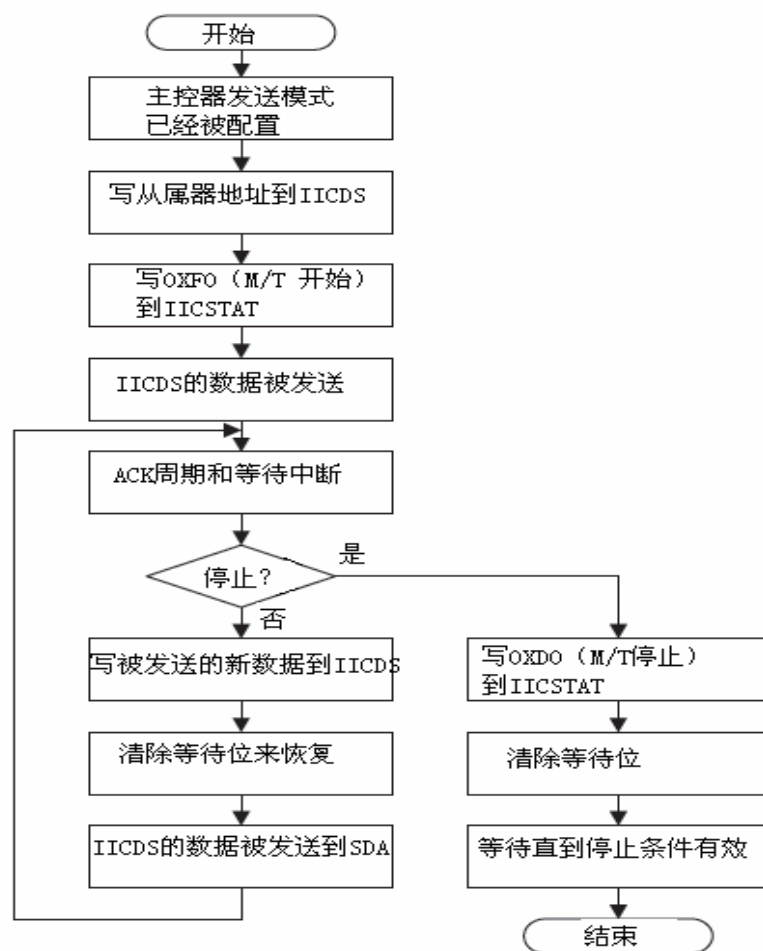


图 30-5 主控器/发送器操作模式

主控器/接收器操作模式的流程图，如图 30-6 所示。

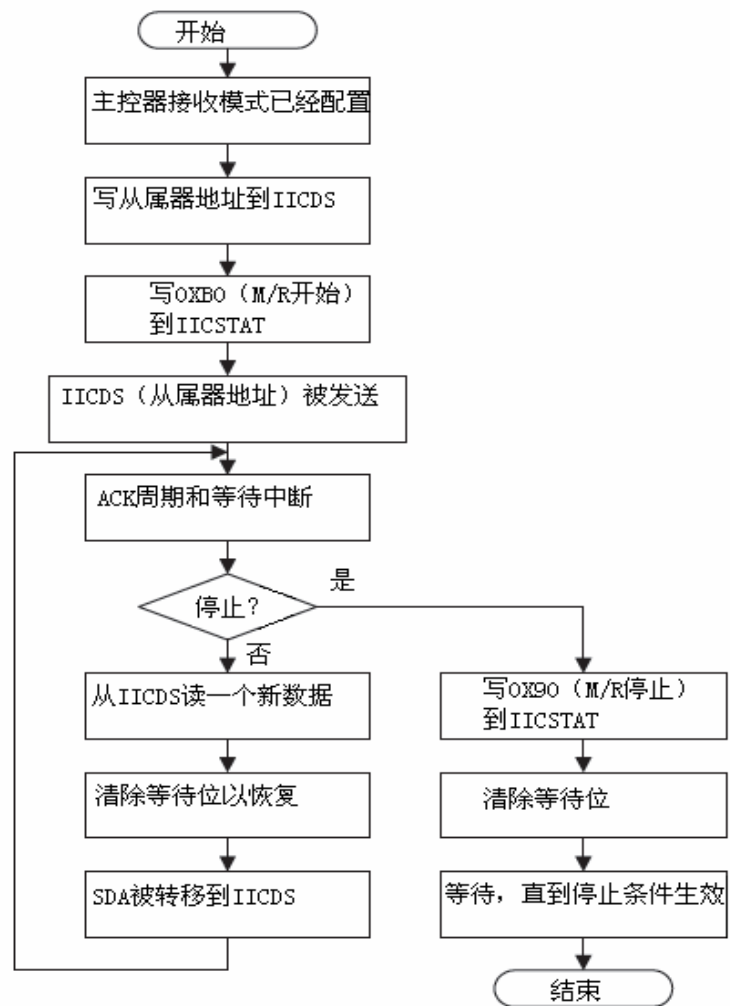


图 30-6 主控制器/接收器操作模式

从属器/发送器操作模式的操作流程图，如图 30-7 所示。

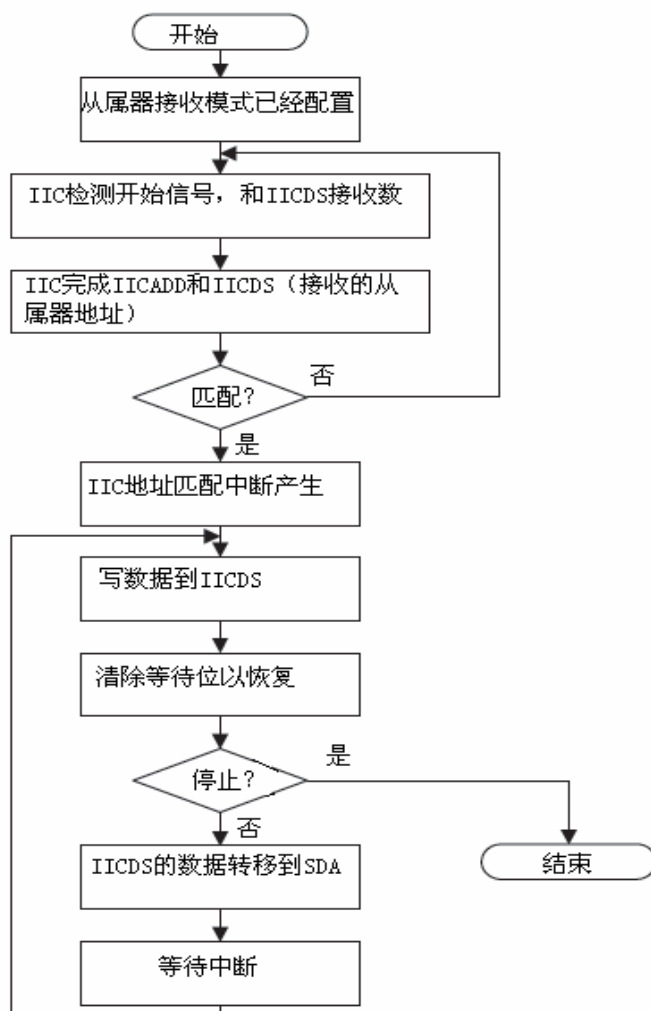


图 30-7 从属器/发送器操作模式

从属器/接收器操作模式的操作流程图，如图 30-8 所示。

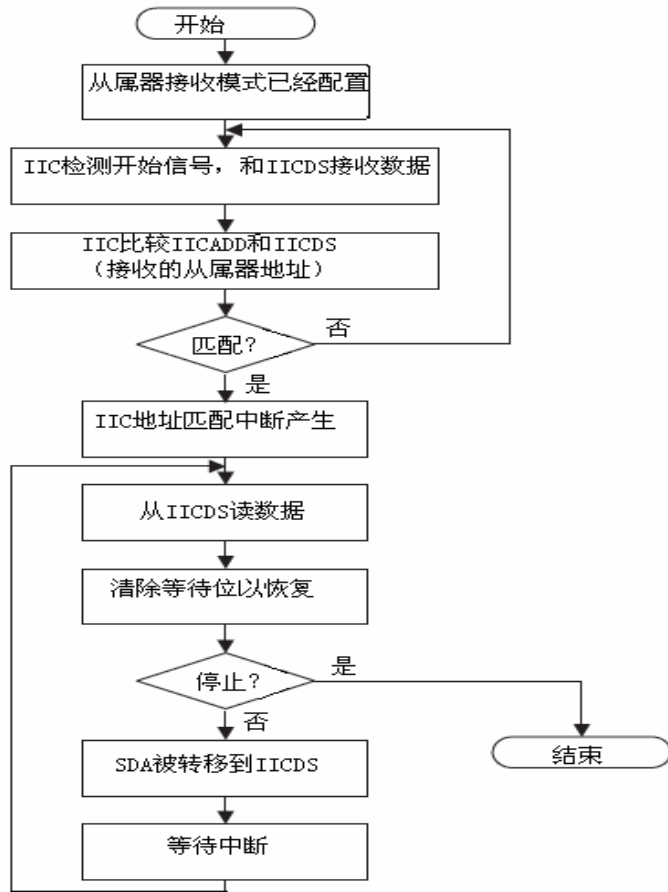


图 30-8 从属器/接收器操作模式

## 30.3 IIC 总线接口特殊的寄存器

IIC 总线接口特殊的寄存器介绍如下：

### 30.3.1. 多主控器 IIC 总线控制（IICCON）寄存器

寄存器	地址	读/写	描述	复位值
IICCON	0x7F004000	读/写	IIC总线控制寄存器。	0x0X

IICCON	位	描述	初始状态
确认产生(1)	[7]	IIC总线确认有效位。 0: 无效 1: 有效 发送模式下，在确认期间，IICSDA空闲；接收模式下，在确认期间，IICSDA为L。	0
发送时钟源选择	[6]	IIC总线发送时钟预分频选择位的源时钟。 0: IICCLK = fPCLK /16 1: IICCLK = fPCLK /512	0
发送/接收中断 (5)	[5]	IIC总线发送/接收中断有效/无效位。 0: 无效 1: 有效	0
中断等待标志 (2) (3)	[4]	IIC总线发送/接收中断等待标志。当该位以1被读取时，IICSDL连接到L并且IIC停止。为了恢复操作，清除该位为0。 0: (1) 无中断等待（读时）； (2) 清除等待条件并且恢复操作（写时） 1: (1) 等待中断（读时）； (2) N/A（写时）。	0
发送时钟值 (4)	[3:0]	IIC总线发送时钟预分频。 IIC总线发送时钟频率由4位预分频值决定，下面的格式： 发送时钟 = IICCLK/(IICCON[3:0]+1)。	未定义

注：

(1) EEPROM接口，在接收模式下，为了产生停止条件而读取最后的数据前，ACK的产生可能无效。

(2) 一个IIC总线中断产生：

- 当一个字节传输或者一个接收操作完成；
- 当一个通用调用或者一个从属器地址匹配发生；
- 如果总线裁定失败。

(3) 为了在SCL上升边缘调整SDA的设置时间，在清除IIC中断等待之前，不得不写入IICDS。

(4) IICCLK由IICCON [6]决定。

通过SCL改变时间，能改变发送时钟。

当IICCON[6]=0时，不能使IICCON[3:0]=0x0或者0x1成立。

(5)如果IICCON[5]=0, IICCON[4]没有正确操作。因此，尽管不使用IIC中断，也推荐设置IICCON[5]=1。

### 30.3.2. 多主控器 IIC 总线控制/状态（IICSTAT）寄存器

寄存器	地址	读/写	描述	复位值
IICSTAT	0x7F004004	读/写	IIC总线控制/状态寄存器。	0x0

IICSTAT	位	描述	初始状态
模式选择	[7:6]	IIC总线主控器/从属器发送/接收模式选择位。  00:从属器接收模式  01:从属器发送模式  10:主控器接收模式  11:主控器发送模式	00
繁忙信号状态 / START STOP 条件	[5]	IIC总线繁忙信号状态位。  0:（读）不繁忙（当读取时） （写）停止信号产生  1:（读）繁忙（当读取时） （写）START信号产生。  在开始信号后，IICDS中的数据将被自动发送。	0
连续输出	[4]	IIC总线数据输出有效/无效位。  0:无效接收/发送 1:有效接收/发送	0
仲裁状态标志	[3]	IIC总线裁定程序状态标志位  0:总线裁定成功  1:在串行I/O过程中，总线裁定失败	0

地址总线作为从属状态标志	[2]	IIC总线地址作为从属状态标志位。 0:读IIC总线寄存器后清除 1:接收的从属器地址匹配IICADD中的地址值	0
地址0状态标志	[1]	IIC总线地址0状态标志位。 0:当开始/停止条件被检测到时清除 1:接收的从属器地址是00000000b	0
最后接收位状态标志	[0]	IIC总线最后接收位状态标志位。 0:最后接收位为0（ACK被接收） 1:最后接收位为1（ACK没有被接收）	0

### 30.3.3. 多主控器 IIC 总线地址（IICADD）寄存器

寄存器	地址	读/写	描述	复位值
IICADD	0x7F004008	读/写	IIC总线地址寄存器。	0xXX

IICADD	位	描述	初始状态
从属器地址	[7:0]	7位从属器地址。  当IICSTAT中串行输出有效=0时，IICADD写有效。不管当前串行输出有效位（IICSTAT）的设置怎样，IICADD值都能被读取。  从属器地址：[7:1]  无映射：[0]	XXXXXXXX

### 30.3.4. 多主控制器 IIC 总线发送/接收数据移位（IICDS）寄存器

寄存器	地址	读/写	描述	复位值
IICDS	0x7F00400C	读/写	IIC总线发送/接收数据移位寄存器。	0xXX

IICDS	位	描述	初始状态
数据移位	[7:0]	用于IIC总线发送/接收操作的8位数据移位寄存器。 当IICSTAT中串行输出有效=1，IICDS写入有效。无论当前串行输出有效位（IICSTAT）设置怎样，IICDS值都能被读。	XXXXXXXX

### 30.3.5. 多主控器 IIC 总线控制寄存器

寄存器	地址	读/写	描述	复位值
IICLC	0x7F004010	读/写	IIC总线主控器线控制寄存器。	0x00

IICLC	位	描述	初始状态
滤波器有效	[2]	IIC总线过滤器有效位。 当SDA端口用于输入操作，该位应当置高位。在两倍的PCLK时间期间，过滤器能预防由于失灵而发生的错误。	0
SDA 输出延迟	[1:0]	IIC总线SDA线延迟长度选择位。 SDA线被以下面的时钟时间延迟（PCLK）： 00: 0 时钟 01: 5 时钟 10: 10 时钟 11: 15 时钟	00

## 30.4 IIC 总线寄存器编程举例

通过前面对IIC总线接口的概述及操作模式的理解，相信大家对于IIC总线接口内容的掌握应该很容易了。以下是针对IIC部分的相关实例代码，有助于更好地理解和掌握该部分的功能及特性。结合8.11.3小节中寄存器的描述进行说明。

1. **IIC\_MasterWrP函数：**功能主要是通过轮询操作进行的主控器发送模式。

输入：cSlaveAddr [8bit SlaveDeviceAddress],

pData [pointer of Data which you want to Tx]



输出: NONE

```
void IIC_MasterWrP(u8 cSlaveAddr,u8 * pData)
{
    u32 uTmp0;
    u32 uTmp1;
    u8 cCnt;
    s32 sDcnt = 100;
    u32 uPT = 0;

    uTmp0 = Inp32(rIICSTAT);
    while(uTmp0 & (1 << 5))    // 等待, 直到IIC总线被释放
    {
        uTmp0 = Inp32(rIICSTAT);
    }
    uTmp1 = Inp32(rIICCON);
    uTmp1 |= (1 << 7);
    Outp32(rIICCON,uTmp1);    // Ack发生使能

    Outp32(rIICDS,cSlaveAddr);

    Outp32(rIICSTAT,0xf0);    // 主控器发送开始

    while(!(sDcnt == -1))
    {
        if(Inp8(rIICCON)&0x10)
        {
            if((sDcnt--)== 0)
            {
                Outp32(rIICSTAT,0xd0);    //停止主控器发送条件,ACK标志清除
            }
        }
    }
}
```

```

        uTmp0 = Inp32(rIICCON);
        uTmp0 &= ~(1<<4);          //清除等待位，重新开始
        Outp32(rIICCON,uTmp0);

        Delay(1);                  //等待，直到停止条件生效
        break;
    }
    Outp8(rIICDS,pData[uPT++]);
    for(cCnt=0;cCnt<10;cCnt++);    //用于建立时间 (IIC_SCL的上升沿)

    uTmp0 = Inp32(rIICCON);
    uTmp0 &= ~(1<<4);          //清除等待位，重新开始
    Outp32(rIICCON,uTmp0);
}
}
}

```

## 2. IIC\_MasterRdP函数：功能主要是通过轮询操作进行的主控器接收模式。

输入： cSlaveAddr [8bit SlaveDeviceAddress],

pData [pointer of Data which you want to Rx]

输出： NONE

```
void IIC_MasterRdP(u8 cSlaveAddr,u8 * pData)
```

```

{
    u32 uTmp0;
    u32 uTmp1;
    u8 cCnt;

    uTmp0 = Inp32(rIICSTAT);

```

```

while(uTmp0 & (1 << 5))          //等待，直到IIC总线被释放
{
    uTmp0 = Inp32(rIICSTAT);
}
uTmp1 = Inp32(rIICCON);
uTmp1 |= (1 << 7);
Outp32(rIICCON, uTmp1);          // Ack 发生使能

Outp32(rIICDS, cSlaveAddr);

Outp32(rIICSTAT, 0xB0);          //主控制器接收开始

while((Inp8(rIICSTAT) & 0x1))
{
}
cCnt = 0;

while(cCnt < 101)
{
    if(Inp8(rIICCON) & 0x10)
    {
        pData[cCnt] = Inp8(rIICDS);
        cCnt++;

        uTmp0 = Inp32(rIICCON);
        uTmp0 &= ~(1 << 4);          //清除等待位，重新开始
        Outp32(rIICCON, uTmp0);
    }
}

```

```

    Outp8(rIICSTAT,0x90);          //停止位产生
}

```

**3. IIC\_SlaveRdP函数：**功能主要是通过轮询操作进行的从属器接收模式。

输入： pSlaveAddr [pointer of 8bit SlaveDeviceAddress],

pData [pointer of Data which you want to Rx]

输出： NONE

```

void IIC_SlaveRdP(u8 *pSlaveAddr,u8 *pData)
{
    u32 uTmp0;
    u32 uTmp1;
    u8 cCnt;

    // g_PcIIC_BUFFER =pData;
    // g_uIIC_PT=0;

    uTmp0 = Inp32(rIICSTAT);
    while(uTmp0&(1<<5))          //等待，直到IIC总线被释放
    {
        uTmp0 = Inp32(rIICSTAT);
    }

    uTmp1 = Inp32(rIICCON);
    uTmp1 |= (1<<7);
    Outp32(rIICCON,uTmp1);      //Ack发生使能

    uTmp0 = Inp32(rIICSTAT);
    uTmp0 &= ~(1<<4);
}

```

```

    Outp32(rIICSTAT,uTmp0);          //禁用接收/发送，用于设置从属器地址
    Outp8(rIICADD,*pSlaveAddr);

    Outp32(rIICSTAT,0x10);          //从属器接收开始

    printf("Wait for Slave Addr\n");
    cCnt=0;

// while(!((Inp8(rIICSTAT)>>2)&(0x1)));
    while(cCnt<101)
    {
        if(Inp8(rIICCON)&0x10)
        {
//          printf("IICSTAT = %x  ",Inp32(rIICSTAT));

            pData[cCnt]=Inp8(rIICDS);
            cCnt++;

            uTmp0 = Inp32(rIICCON);
            uTmp0 &= ~(1<<4);          //清除等待位，重新开始
            Outp32(rIICCON,uTmp0);
        }
    }
    *pSlaveAddr = pData[0];
    Outp8(rIICSTAT,0x0);
}

```

#### 4. IIC\_SlaveWrP函数：功能主要是通过轮询操作进行的从属器发送模式。

输入： pSlaveAddr [pointer of 8bit SlaveDeviceAddress],

pData [pointer of Data which you want to Tx]

输出: NONE

```
void IIC_SlaveWrP(u8 *pSlaveAddr,u8 *pData)
{
    u32 uTmp0;
    u32 uTmp1;
    u8 cCnt;
    s32 sDcnt = 100;
    u32 uPT = 0;

    // g_PcIIC_BUFFER = pData;
    // g_uIIC_PT = 0;
    uTmp0 = Inp32(rIICSTAT);
    while(uTmp0 & (1 << 5)) //等待，直到IIC总线被释放
    {
        uTmp0 = Inp32(rIICSTAT);
    }

    uTmp1 = Inp32(rIICCON);
    uTmp1 |= (1 << 7);
    Outp32(rIICCON,uTmp1); //Ack发生使能

    uTmp0 = Inp32(rIICSTAT);
    uTmp0 &= ~(1 << 4);
    Outp32(rIICSTAT,uTmp0); //禁用接收/发送，用于设置从属器地址

    Outp8(rIICADD,*pSlaveAddr);

    Outp32(rIICSTAT,0x50); //从属器发送开始
```

```

// while(!((Inp8(rIICSTAT)>>2)&(0x1)));
while(!(sDcnt == -1))
{
    if(Inp8(rIICCON)&0x10)
    {
        // printf("IICSTAT = %x ",Inp32(rIICSTAT));
        if((sDcnt--)== 0)
        {
            Outp32(rIICSTAT,0xd0);          //停止从属器发送条件,ACK标志清除

            uTmp0 = Inp32(rIICCON);
            uTmp0 &= ~(1<<4);                //清除等待位, 重新开始
            Outp32(rIICCON,uTmp0);

            Delay(1);                        //等待, 直到停止条件生效
            break;
        }
        Outp8(rIICDS,pData[uPT++]);
        for(cCnt=0;cCnt<10;cCnt++);        //用于建立时间 (IICSCL上升沿)

        uTmp0 = Inp32(rIICCON);
        uTmp0 &= ~(1<<4);                //清除等待位, 重新开始
        Outp32(rIICCON,uTmp0);
    }
}
}

```

## 31 UART 接口

这一节介绍 S3C6410 RSIC 微处理器上的通用异步接收/发送器 (UART) 串行端口。

该 S3C6410 通用异步接收和发送器 (UART) 提供了四个独立的异步串行 I / O (SI0) 端口。每个异步串行 I/O (SI0) 端口通过中断或者直接存储器存取 (DMA) 模式来操作。换句话说, UART 是通过产生一个中断或 DMA 请求, 在 CPU 和 UART 之间传输数据的。该 UART 使用系统时钟的时间可以支持的比特率最高为 115.2kb/s。如果一外部设备提供 ext\_uclk0 或 ext\_uclk1, 则 UART 可以以更高的速度运行。每个 UART 的通道包含了两个 64 字节收发 FIFO 存储器。

该 S3C6410 的 UART 包括可编程波特率, 红外线 (IR) 的传送/接收, 一个或两个停止位插入, 5 位, 6 位, 7 位或 8 位数据的宽度和奇偶校验。

### 31.1 UART 接口特性

UART 的特性包括:

- (1) 基于 rxd0, txd0, rxd1, txd1, rxd2, txd2, rxd3 和 txd3 的 DMA 或中断来操作。
- (2) UART 通道 0、1、2 符合 IrDA 1.0 要求, 且具有 16 字节的 FIFO。
- (3) UART 通道 0、1 具有 nRTS0, nCTS0, nRTS1 和 nCTS1。
- (4) 支持收发时握手模式。

每个 UART 包含一个波特率发生器, 发送器, 接收器和控制单元。该波特率发生器由 pclk, ext\_uclk0 或 ext\_uclk1 进行时钟控制。发射器和接收器包含 64 字节的 FIFO 存储器和数据移位寄存器。发送数据之前, 首先将数据写入 FIFO 存储器, 然后复制到发送移位寄存器。通过发送数据的引脚 (txdn) 将数据发送, 同时, 通过数据接收的引脚 (rxdn) 将接收到的数据从接收移位寄存器复制到 FIFO 存储器。

UART 的结构框图, 如图 31-1 所示。



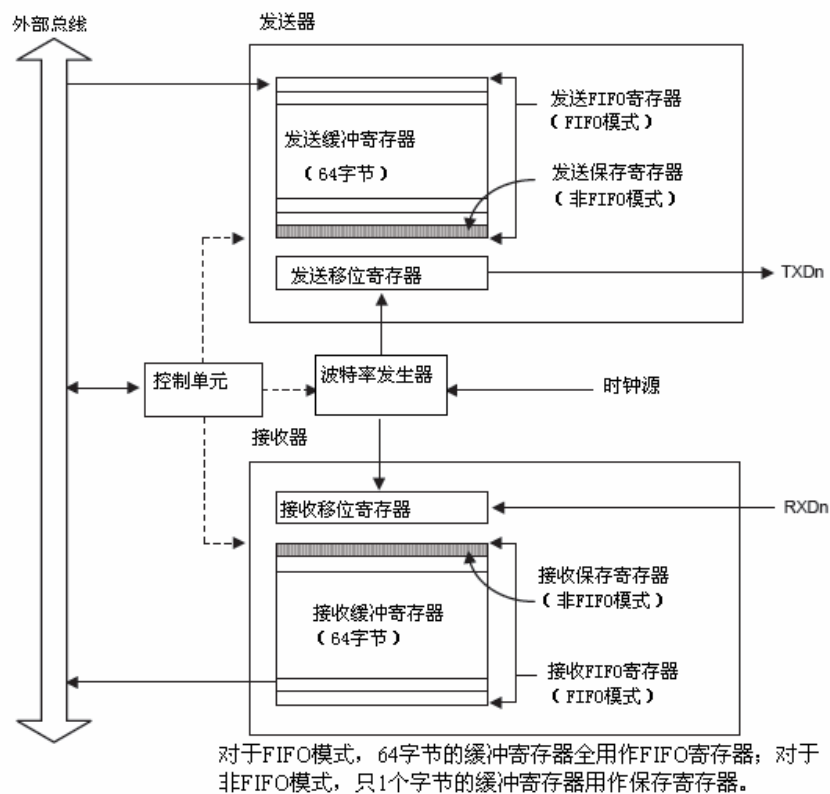


图 31-1 UART 结构框图

## 31.2 UART 的操作

下面介绍 UART 的操作，包括数据传输，数据接收，中断产生，波特率产生，环回模式，红外线模式和自动流量控制。

### 1. 数据发送

数据帧发送是可编程的。它由一个起始位，5~8 个数据位，一个可选的奇偶位和 1~2 个可由行控制寄存器（ULCONn）指定的停止位组成。发送器也可以产生中断条件，在传输过程中，它通过置位逻辑状态 0 来强制串行输出。当目前的发送完全传输完成后，发送中断信号。然后不断传送数据到发送 FIFO 寄存器（在非 FIFO 的模式下，发送保存寄存器）。

## 2. 数据接收

和数据发送一样，数据帧接收也是可编程的。它是由一个起始位，5~8 个数据位，一个可选的奇偶位和行控制寄存器指定的 1~2 个停止位组成。接收器可以检测到溢出错误、奇偶错误、帧错误和中断条件，并为它们设置错误标志。

溢出错误说明在数据被读取之前，新的数据已经将原有的数据覆盖。

奇偶错误说明接收器已经检测到一个意外的奇偶条件。

帧错误表示收到的数据没有有效的停止位。

中断条件表明接收过程中置位逻辑状态 0 的时间比发送一帧的时间长。

当三个字的时间（间隔由设置的字长决定）间隔内没有接收任何数据，并且 FIFO 模式下接收 FIFO 寄存器不为空，接收超时条件发生。

## 3. 自动流量控制（AFC）

该 S3C6410 的 UART0 和 UART1 通过 nRTS 和 nCTS 信号支持自动流量控制。某种情况下，它可以连接到外部 UART。如果想要将 UART 和一台调制解调器连接，必须在 UMCOn 寄存器中禁用自动流量控制位，并且通过软件控制信号 nRTS。

在自动流量控制过程中，nRTS 依靠接收器的条件，nCTS 信号控制发送器的运作。只有当 nCTS 信号被激活（在 AFC 中，nCTS 意味着另一个 UART 的 FIFO 寄存器准备接收数据），UART 的发送器发送 FIFO 寄存器中的数据。在 UART 接收数据之前，如果它接收 FIFO 寄存器超过两个字节以上的空间，nRTS 则被激活；如果它的接收 FIFO 寄存器只有不足一个字节的空间，nRTS 则停止活动（在 AFC 中，nRTS 意味着它本身的接收 FIFO 寄存器已经真被接收数据）。如图 31-2 所示显示了 UART AFC 接口的发送和接受状态图。

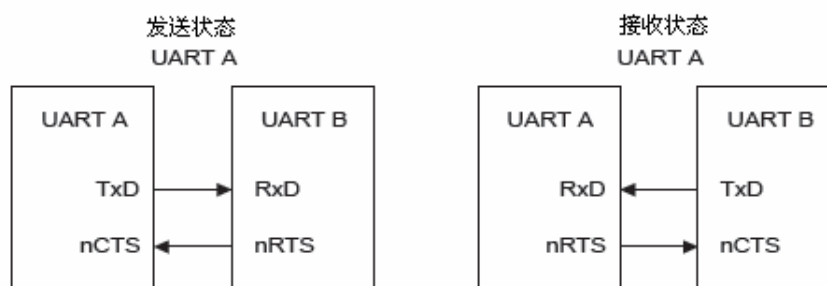


图 31-2 UART AFC 接口

非自动流量控制的例子。（通过软件控制 nRTS 和 nCTS）

#### 4. 接收 FIFO 的操作

- (1) 选择接收模式（中断或 DMA 模式）。
- (2) 在 UFSTATn 中，查看 RX FIFO 计数器的值。如果该值小于 15，必须设置 UMCOn[0] 值为 ‘1’（激活 nRTS）；如果该值等于或大于 15，必须先设定 UMCOn[0] 值为 ‘0’（停止 nRTS）。
- (3) 重复步骤（2）。

#### 5. 发送 FIFO 的操作

- (1) 选择发送模式（中断或 DMA 模式）。
- (2) 检查 UMSTATn[0] 的值，如果这个值是 ‘1’（激活 nCTS），则写数据到发送 FIFO 寄存器。
- (3) 重复步骤（2）。

#### 6. RS-232C 接口

要将 UART 连接到调制解调器接口（而不是零调制解调器），需要 nRTS，nCTS，nDSR，nDTR，DCD 和 nRI 信号。在这种情况下，可以通过软件来控制这些信号与一般的 I / O 端口。因为 AFC 不支持 RS - 232c 接口。

#### 7. 中断/DMA 请求的产生

每个 S3C6410 的 UART 有七个状态（发射/接收/错误）信号：溢出错误，奇偶错误，帧错误，中断，接收缓冲区数据就绪，传输缓冲区为空，发送移位寄存器为空。其状态信号靠相应的 UART 的状态寄存器（UTRSTATn / UERSTATn）来指示。

溢出错误，奇偶错误，帧错误，中断条件是指由于收到错误的信息。每一种都可以引起错误接收错误状态中断请求，如果在控制寄存器 UCONn 中将接收错误状态中断使能位设置为 1，当检测到一个接收错误状态中断请求，可通过读 UERSTATn 的值来辨别信号。

当接收器将数据从接收移位寄存器传送到接收 FIFO 寄存器（在 FIFO 模式下），并且数量达到 RX FIFO 触发电平，则接收中断产生。如果控制寄存器（UCONn）中接收模式设置为 1（中断请求或轮询模式），则接收中断产生。

非 FIFO 模式中，在中断请求和轮询模式下，数据从接收移位寄存器传输到接收保存寄存器时会引发接收中断。

当发送器将数据从发送 FIFO 寄存器传输到它的发送移位寄存器，并且发送 FIFO 剩余的数据数量达到 TX FIFO 触发水平，发送中断产生。如果控制器的传输模式选定为中断请求或轮询模式，发送中断产生。

非 FIFO 模式中，在中断请求和轮询模式下，数据从发送保存寄存器传输到发送移位寄存器会引发发送中断。

注意的是无论什么时候在发送 FIFO 中数据的数量是小于触发水平，发送中断一直请求。这就是说，只要发送中断被激活就请求中断，除非你先添满发送缓冲区。建议先添满发送缓冲区，然后再激活发送中断。

S3C6410 的中断控制器是一级触发类型，当你为 UART 控制寄存器编程时，必须建立中断类型为“一级”。在上述情况下，如果接收模式和发送模式的控制器获得 DMA 请求，则 DMA 请求代替接收中断和发送中断。

与 FIFO 有关的中断，如表 31-1 所示。

表 31-1 与 FIFO 相连的中断

类型	FIFO 模式	Non-FIFO 模式
接受中断	如果每次接收的数据达到了接收 FIFO 的触发水平，则 Rx 中断产生。 如果 FIFO 非空并且在 3 字时间内（接收超时）没有接收到数据，则 Rx 中断也将产生。这段时间间隔由字的长度设置决定。	如果每次接收缓冲区满时，接收保持寄存器产生一个中断。
发送中断	如果每次发送的数据达到了发送 FIFO 的触发水平，则 Tx 中断产生。	当发送缓冲区的数据变为空，发送保持寄存器产生一
错误中断	当溢出错误、奇偶错误、帧错误、中断信号被检测到时出发。	错误发生时产生，如果同时另一个错误发生，只产生一

8. UART 错误状态 FIFO

除了 Rx FIFO 寄存器之外，UART 还具有一个错误状态 FIFO。错误状态 FIFO 中表示了在 FIFO 寄存器中，哪一个数据在接收时出错。错误中断发生在有错误的的数据被读取时。为清除错误状态 FIFO，寄存器 URXHn 和 UERSTATn 会被读取。

例如：

假设 UART 的 Rx FIFO 连续接收到 A, B, C, D, 字符，并且在接收 B 字符时发生了帧错误（即该字符没有停止位），在接收 D 字符时发生了奇偶校验错。

虽然 UART 错误发生了，错误中断不会产生，因为含有错误的字符还没有被 CPU 读取。

当字符被读出时错误中断才会发生。UART 接收五个字节其中包含两个错误的情况，如表 31-2 和图 31-3

所示。

表 31-2 UART 接收五个字节其中包含两个错误

时间	队列顺序	错误中断	说明
#0	没有读取字符		
#1	接收 A、B、C、D 和 E		
#2	读取 A 后	帧错误（对于 B）中断产生	必须读取 ‘B’
#3	读取 B 后		
#4	读取 C 后	奇偶错误（对于 D）中断产生	必须读取 ‘D’
#5	读取 D 后		
#6	读取 E 后		

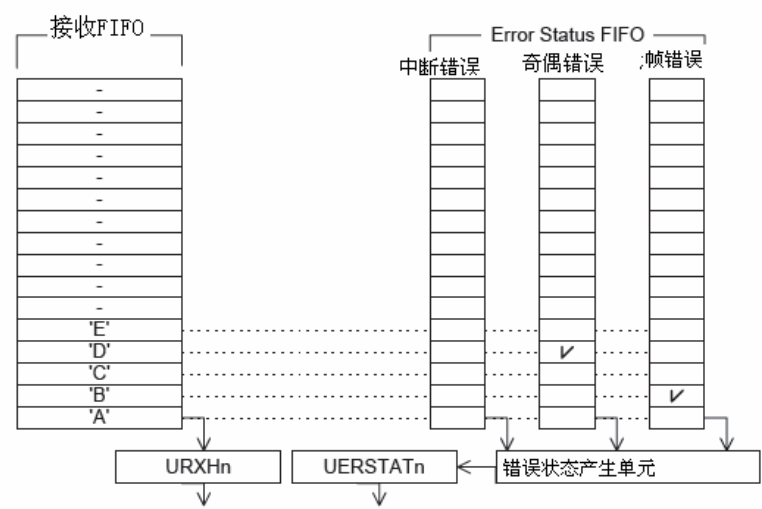


图 31-3 UART 接收五个字节其中包含两个错误的情况

9. 红外线（IR）模式

S3C64100 的 UART 模块支持红外线 (IR) 发送和接收，可以通过设置 UART 控制寄存器（ULCONn）中的红外模式位来选择这一模式。如图 31-4 所示为如何实现 IR 模式。

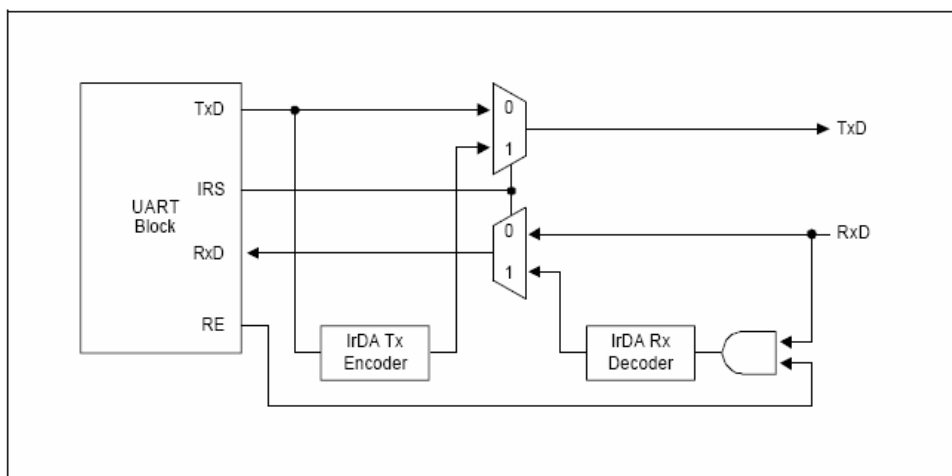
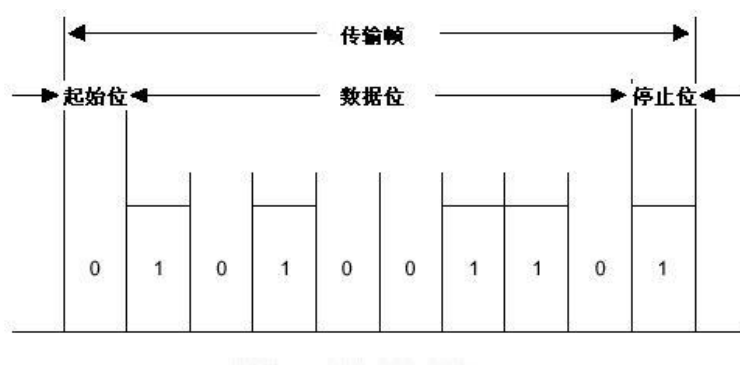
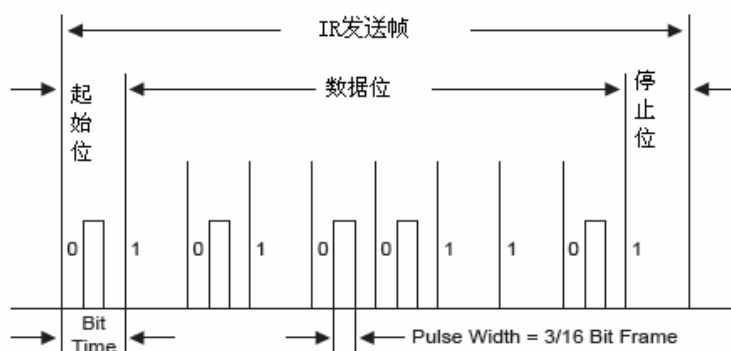


图 31-4 IrDA 功能模块框图

在 IR 发送模式下，发送阶段通过正常串行发送占空比 3/16 的脉冲波调制（当传送的数据位为 0）；在 IR 接收模式下，接收必须检测 3/16 脉冲波来识别 0 值。如图 31-5 所示。



(a) 通常情况上传输帧的时序图



(b) 红外线发送模式下时序图



(c) 红外线接收模式下时序图

图 31-5 红外线传输模式时序

## 31.3 外部接口

UART 外部接口，如表 31-3 所示。

表 31-3 UART 外部接口

名称	类型	源/目的	描述
XuRXD[0]	输入	Pad	UART0 接收数据
XuTXD[0]	输出	Pad	UART0 发送数据
XuCTSn[0]	输入	Pad	UART0 清除发送（低位有效）
XuRTSn[0]	输出	Pad	UART0 请求发送（低位有效）
XuRXD[1]	输入	Pad	UART1 接收数据
XuTXD[1]	输出	Pad	UART1 发送数据
XuCTSn[1]	输入	Pad	UART1 清除发送（低位有效）
XuRTSn[1]	输出	Pad	UART1 请求发送（低位有效）
XuRXD[2]	输入	Pad	UART2 接收数据
XuTXD[2]	输出	Pad	UART2 发送数据
XuRXD[3]	输入	Pad	UART3 接收数据
XuTXD[3]	输出	Pad	UART3 发送数据

注： UART 与其它的接口处理器 (CFCON, IrDA 等等) 共享外部信息包。为了使用这些信息包, 必须提前设置通用 I/O 口。

### 31.4 寄存器描述

存储器映像，如表 31-4 所示。

表 31-4 UART 的寄存器

寄存器	地址	读/写	说明	初始值
ULCON0	0x7F005000	读/写	UART 通道 0 行控制寄存器	0x00
UCON0	0x7F005004	读/写	UART 通道 0 控制寄存器	0x00
UFCON0	0x7F005008	读/写	UART 通道 0 FIFO 控制寄存器	0x0
UMCON0	0x7F00500C	读/写	UART 通道 0 调制解调器 (Modem) 控制寄存器	0x0
UTRSTAT0	0x7F005010	读	UART 通道 0 发送/接收状态寄存器	0x6
UERSTAT0	0x7F005014	读	UART 通道 0 接收错误状态寄存器	0x0
UFSTAT0	0x7F005018	读	UART 通道 0 FIFO 状态寄存器	0x00
UMSTAT0	0x7F00501C	读	UART 通道 0 调制解调器 (Modem) 状态寄存器	0x0
UTXH0	0x7F005020	写	UART 通道 0 发送缓冲寄存器	-
URXH0	0x7F005024	读	UART 通道 0 接收缓冲寄存器	0x00
UBRDIV0 UART	0x7F005028	读/写	通道 0 波特率分频寄存器	0x0000
UDIVSLOT0	0x7F00502C	读/写	UART 通道 0 分频插槽寄存器	0x0000
UINTP0 UART	0x7F005030	读/写	通道 0 中断处理寄存器	0x0
UINTSP0	0x7F005034	读/写	UART 通道 0 中断源处理寄存器	0x0
UINTM0	0x7F005038	读/写	UART 通道 0 中断屏蔽寄存器	0x0
ULCON1	0x7F005400	读/写	UART 通道 1 行控制寄存器	0x00
UCON1	0x7F005404	读/写	UART 通道 1 控制寄存器读	0x00
UFCON1	0x7F005408	读/写	UART 通道 1 FIFO 控制寄存器	0x0
UMCON1	0x7F00540C	读/写	UART 通道 1 调制解调器 (Modem) 控制寄存器	0x0
UTRSTAT1	0x7F005410	读	UART 通道 1 发送/接收状态寄存器	0x6



UERSTAT1	0x7F005414	读	UART 通道 1 接收错误状态寄存器	0x0
UFSTAT1	0x7F005418	读	UART 通道 1 FIFO 状态寄存器	0x00
UMSTAT1	0x7F00541C	读	UART 通道 1 调制解调器 (Modem) 状态寄存器	0x0
UTXH1	0x7F005420	写	UART 通道 1 发送缓冲寄存器	–
URXH1	0x7F005424	读	UART 通道 1 接收缓冲寄存器。	0x00
UBRDIV1	0x7F005428	读/写	UART 通道 1 波特率分频寄存器。	0x0000
UDIVSLOT1	0x7F00542C	读/写	UART 通道 1 分频插槽寄存器。	0x0000
UINTP1	0x7F005430	读/写	UART 通道 1 中断处理寄存器。	0x0
UINTSP1	0x7F005434	读/写	UART 通道 1 中断源处理寄存器。	0x0
UINTM1	0x7F005438	读/写	UART 通道 1 中断屏蔽寄存器。	0x0
ULCON2	0x7F005800	读/写	UART 通道 2 行控制寄存器。	0x00
UCON2	0x7F005804	读/写	UART 通道 2 控制寄存器。	0x00
UFCON2	0x7F005808	读/写	UART 通道 2 FIFO 控制寄存器。	0x0
UTRSTAT2	0x7F005810	读	UART 通道 2 发送/接收状态寄存器。	0x6
UERSTAT2	0x7F005814	读	UART 通道 2 接收错误状态寄存器。	0x0
UFSTAT2	0x7F005818	读	UART 通道 2 FIFO 状态寄存器。	0x00
UTXH2	0x7F005820	写	UART 通道 2 发送缓冲寄存器。	–
URXH2	0x7F005824	读	UART 通道 2 接收缓冲寄存器。	0x00
UBRDIV2	0x7F005828	读/写	UART 通道 2 波特率分频寄存器。	0x0000
UDIVSLOT2	0x7F00582C	读/写	UART 通道 2 分频插槽寄存器。	0x0000
INTP2	0x7F005830	读/写	UART 通道 2 中断处理寄存器。	0x0
UINTM2	0x7F005838	读/写	UART 通道 2 中断屏蔽寄存器。	0x0
ULCON3	0x7F005C00	读/写	UART 通道 3 行控制寄存器。	0x00
UCON3	0x7F005C04	读/写	UART c 通道 3 控制寄存器。	0x00
UFCON3	0x7F005C08	读/写	UART 通道 3 FIFO 控制寄存器。	0x0
UTRSTAT3	0x7F005C10	读	UART 通道 3 发送/接收状态寄存器。	0x6
UERSTAT3	0x7F005C14	读	UART 通道 3 接收错误状态寄存器。	0x0
UFSTAT3	0x7F005C18	读	UART 通道 3 FIFO 状态寄存器。	0x00

UTXH3	0x7F005C20	写	UART 通道 3 发送缓冲寄存器。	-
URXH3	0x7F005C24	读	UART 通道 3 接收缓冲寄存器。	0x00
UBRDIV3	0x7F005C28	读/写	UART 通道 3 波特率分频寄存器。	0x0000
UDIVSLOT3	0x7F005C2C	读/写	UART 通道 3 分频插槽寄存器。	0x0000
INTP3	0x7F005C30	读/写	UART 通道 3 中断处理寄存器。	0x0
UINTSP3	0x7F005C34	读/写	UART 通道 3 中断源处理寄存器。	0x0
UINTM3	0x7F005C38	读/写	UART 通道 3 中断屏蔽寄存器。	0x0

下面主要针对个别寄存器进行介绍。这里我们是以列表的形式给出的。

### 31.4.1 . UART 行控制寄存器

在 UART 模块包括四个行控制寄存器，即 ULCON0、ULCON1、ULCON2 和 ULCON3。下面就来看看行控制寄存器的位定义。

寄存器	地址	读/写	描述	复位值
ULCON0	0x7F005000	读/写	UART 0 通道行控制寄存器。	0x00
ULCON1	0x7F005400	读/写	UART1 通道行控制寄存器。	0x00
ULCON2	0x7F005800	读/写	UART2 通道行控制寄存器。	0x00
ULCON3	0x7F005C00	读/写	UART3 通道行控制寄存器。	0x00

位名称	位	描述	初始状态
Reserved	[7]	保留。	0
Infra-Red Mode	[6]	确定是否采用红外模式： 0 = 普通操作模式 1= 红外线输出/接收模式	0

Parity Mode	[5:3]	确定奇偶产生类型和校验，在 UART 发送/接收操作过程中。  0xx = 无校验  100 =奇校验  101 = 偶校验  110 = 奇偶强制/校验为 1  111 = 奇偶强制/校验为 0  111 = 强制为 0	000
Number of Stop Bit	[2]	确定每帧中停止位个数：  0 = 每帧 1 位停止位  1 = 每帧 2 位停止位	0
Word Length	[1:0]	确定每帧中数据位的个数：  00 = 5 位 01 = 6 位  10 = 7 位 11 = 8 位	00

### 31.4.2. UART 控制寄存器

UART 控制寄存器也有四个，即 UCON0、UCON1、 UCON2 和 ULCON3。

寄存器	地址	读/写	描述	复位值
UCON0	0x7F005004	读/写	UART 0 通道控制器。	0x00
UCON1	0x7F005404	读/写	UART1 通道控制器。	0x00
UCON2	0x7F005804	读/写	UART2 通道控制器。	0x00
UCON3	0x7F005C04	读/写	UART3 通道控制器。	0x00

位名称	位	描述	初始状态
Clock Selection	[11: 10]	选择 PCLK 或者 EXT_UCLK0 <sup>0</sup> 作为 UART 波特率时钟。  x0=PCLK:DIV_VAL = (PCLK / (b/s × 16) ) -1  01=EXT_UCLK0:DIV_VAL = (EXT_UCLK / (b/s × 16) ) -1  11=EXT_UCLK1:DIV_VAL= (EXT_UCLK / (b/s × 16) ) -1	0

Tx Interrupt Type	[9]	中断请求类型 <sup>2)</sup>  0 = 脉冲（当非 FIFO 模式下的发送缓冲区中的数据发送完毕或者 FIFO 模式下发送 FIFO 达到了触发水平时，中断产生）。  1 = 电平（当非 FIFO 模式下的发送缓冲区中的数据发送完毕或者 FIFO 模式下发送 FIFO 达到了触发水平时，中断产生）。	0
Rx Interrupt Type	[8]	接收中断请求类型 <sup>3)</sup>  0 = 脉冲（当 Rx 缓冲器）  （在非 FIFO 模式下接收缓冲区接收到数据或者在 FIFO 模式下达到接收 FIFO 触发水平时，请求中断）。  1 = 电平（当非 FIFO 模式下接收缓冲区接收到数据或者在 FIFO 模	0
Rx Time Out Enable	[7]	使能/禁止接收超时中断，当 UART FIFO 使能时。  0 = 禁止      1 = 使能	0
Rx Error Status Interrupt Enable	[6]	在接收过程中，如果发生帧错误或溢出错误，使能/禁止 UART 产生中断。  0 = 不产生接收错误状态中断  1 = 产生接收错误状态中断	0
Loop-back Mode	[5]	设置环回位为 1，使 UART 进入环回模式。此模式仅为测试目的使用。  0 = 普通操作      1 = 环回模式	0
Send Break Signal	[4]	设置环回位为 1 使 UART 进入环回模式。这种模式只为测试提供参考。  0=正常发送    1= 发送中断信号	0
Transmit Mode	[3:2]	确定哪个模式可以写发送数据到 UART 发送缓冲寄存器。  00= 禁止  01= 中断请求或轮询模式  10= DMA0 请求(仅用于 UART0)，DMA3 请求（请求信号 0）  11= DMA1 请求(仅请求信号 1)	00

Receive Mode	[1:0]	确定哪个模式可以从 UART 接收缓冲寄存器读数据。  00 = 禁止  01 = 中断请求或轮询模式  10= DMA0 请求 (仅用于 UART0)，DMA3 请求（请求信号 0）	00
--------------	-------	--	----

注：

（1）DIV\_VAL = UBRDIVn + (在 UDIVSL0Tn 上 1 的数量)/16. 涉及 UART 波特率配置寄存器。

（2）是 S3C6410 用的一个水平触发中断控制器。因此每次发送时这个位必须设置为 1。

（3）当 UART 没有达到 FIFO 触发水平或在 FIFO 下 DMA 接收模式中的三个字的时间内没

有接收到数据，Rx 中断会发生（接收超时），并且用户应该检测 FIFO 状态读取中断。

（4）EXT\_UCLK0 是外部时钟。（XpwmECLK PAD 输入）。

EXT\_UCLK1 时钟产生是由 syscon 。SYSCON 产生 EXT\_UCLK1 为分频 EPLL 或 MPLL 输出。

31.4.3. UART 的 FIFO 控制寄存器

UART 模块中含有四个 UART FIFO 控制寄存器。

寄存器	地址	读/写	描述	复位值
UFCON0	0x7F005008	读/写	UART0 通道 FIFO 控制寄存器。	0x0
UFCON1	0x7F005408	读/写	UART1 通道 FIFO 控制寄存器。	0x0
UFCON2	0x7F005808	读/写	UART2 通道 FIFO 控制寄存器。	0x0
UFCON3	0x7F005C08	读/写	UART3 通道 FIFO 控制寄存器。	0x0

位名称	位	描述	初始状态
Tx FIFO Trigger Level	[7:6]	确定发送 FIFO 的触发条件。  00 = 空                    01 = 4 字节  10 = 8 字节            11 = 12 字节	00
Rx FIFO Trigger Level	[5:4]	确定接收 FIFO 的触发条件。  00 = 4 字节            01 = 8 字节  10 = 12 字节    11 = 16 字节	00
Reserved	[3]	保留。	0

Tx FIFO Reset	[2]	Tx 复位，该位在 FIFO 复位后自动清除。 0 = 正常      1= Tx FIFO 复位	0
Rx FIFO Reset	[1]	Rx 复位，该位在 FIFO 复位后自动清除。 0 = 正常      1= Rx FIFO 复位	0
FIFO Enable	[0]	0 =FIFO 禁止      1 = FIFO 模式	0

注意：在 FIFO DMA 接收模式下，当 UART 没有达到 FIFO 触发水平或者在三个字的时间内没有接收到数据时，接收中断会产生（接收超时），并且用户应该检测 FIFO 状态读取中断。

#### 31.4.4. UART Modem 控制寄存器

UART 模块中有两个 UART MODEM 控制寄存器 UMCN0 和 UMCN1。

寄存器	地址	读/写	描述	复位值
UMCN0	0x7F00500C	读/写	UART0 通道 Modem 控制寄存器。	0x0
UMCN1	0x7F00540C	读/写	UART1 通道 Modem 控制寄存器。	0x0
Reserved	0x7F00580C	-	保留。	未定义
Reserved	0x7F005C0C	-	保留。	未定义

位名称	位	描述	初始状态
Reserved	[7:5]	当 AFC 被激活，这个位决定什么时候阻止信号。  000 =接收 FIFO 控制 63 字节 001 =接收 FIFO 控制 56 字节 010 =接收 FIFO 控制 48 字节 011 =接收 FIFO 控制 40 字节 100 =接收 FIFO 控制 32 字节 101 = 接收 FIFO 控制 24 字节	000
Auto Flow Control (AFC)	[4]	AFC 是否允许： 0 = 禁止 1 =激活	0
Reserved	[3:1]	这 3 位必须均为 0。	00

Request to Send	[0]	如果 AFC 位允许，则该位忽略，这时，S3C6410 将自动控制 nRTS。 如果 AFC 位禁止，则 nRTS 必须被软件控制。 0 = 'H' 电平 (nRTS 无效) 1 = 'L' 电平 (nRTS 有效)	0
-----------------	-----	---	---

注：UART2 不支持 AFC 功能，因为 S3C6410 没有 nRTS2 和 nCTS2。  
 UART3 不支持 AFC 功能，因为 S3C6410 没有 nRTS2 和 nCTS2。

### 31.4.5. UART 接收 (Rx) / (Tx) 发送状态寄存器

UART 模块有四个 UART 接收/发送状态寄存器：UTRSTAT0、UTRSTAT1、UTRSTAT2 和 UTRSTAT3。

寄存器	地址	读/写	描述	复位值
UTRSTAT0	0x7F005010	读	UART 0 通道 Tx/Rx 状态寄存器。	0x6
UTRSTAT1	0x7F005410	读	UART 1 通道 Tx/Rx 状态寄存器。	0x6
UTRSTAT2	0x7F005810	读	UART 2 通道 Tx/Rx 状态寄存器。	0x6
UTRSTAT3	0x7F005C10	读	UART 3 通道 Tx/Rx 状态寄存器。	0x6

位名称	位	描述	初始状态
Transmitter empty	[2]	在发送缓冲寄存器没有有效数据或发送移位寄存器为空时，该位自动置 1。 0 = 不空 1 = 发送器（发送缓冲寄存器和移位寄存器）空	1
Transmit buffer empty	[1]	当发送缓冲寄存器为空时，该位自动置 1。 0 =发送缓冲寄存器不空 1 =空  (在非 FIFO 模式，中断或 DMA 被申请。在 FIFO 模式，当 Tx FIFO 触发水平被设置为 00（空）时，中断或 DMA 被申请。）  如果 UART 使用 FIFO，则用户应该检查 UFSTAT 寄存器的 Tx FIFO 计数位和 Tx FIFO 满标志位，以代替检查该位。	1

Receive buffer data ready	[0]	<p>无论何时接收缓冲寄存器包含在 RXDn 接口接收的有效数据，该位自动置 1。</p> <p>0 = 空</p> <p>1 = 接收缓冲寄存器有接收数据（在非 FIFO 模式，中断或 DMA 被申请）。</p> <p>如果 UART 使用 FIFO，则用户应该检查 UFSTAT 寄存器中的 Rx FIFO 计数位和 Rx FIFO 满标志位以代替检查该位。</p>	0
---------------------------	-----	---	---

### 31.4.6. UART 错误状态寄存器

UART 模块有四个 UART 错误状态寄存器：UERSTAT0、UERSTAT1、UERSTAT2 和 UERSTAT3。

寄存器	地址	读/写	描述	复位值
UERSTAT0	0x7F005014	读	UART 0 通道错误状态寄存器。	0x0
UERSTAT1	0x7F005414	读	UART 1 通道错误状态寄存器。	0x0
UERSTAT2	0x7F005814	读	UART 2 通道错误状态寄存器。	0x0
UERSTAT3	0x7F005C14	读	UART 3 通道错误状态寄存器。	0x0

位名称	位	描述	初始状态
Break Detect	[3]	<p>自动设置为 1 说明中断信号接收中断信号。</p> <p>0 = 没有中断信号</p> <p>1 = 中断接收 (请求中断)</p>	0
Frame Error	[2]	<p>在接收过程中无论何时发生帧错误，该位自动置 1。</p> <p>0 = 没发生帧错误</p> <p>1 = 发生帧错误 (请求中断)</p>	0
Parity Error	[1]	<p>在接收过程中无论何时发生奇偶错误，该位自动置 1。</p> <p>0 = 没发生奇偶错误</p> <p>1 = 发生奇偶错误 (请求中断)</p>	0
Overrun Error	[0]	<p>在接收过程中无论何时发生溢出错误时，该位自动置 1。</p> <p>0 = 没发生溢出错误</p> <p>1 = 发生溢出错误 (请求中断)</p>	0

注意：当 UART 错误状态寄存器被读时，这些位会自动清 0。



31.4.7. UART 的 FIFO 状态寄存器

UART 模块有四个 FIFO 状态寄存器：UFSTAT0、UFSTAT1 、UFSTAT2 和 UFSTAT3。

寄存器	地址	读/写	描述	复位值
UFSTAT0	0x7F005018	读	UART 0 通道 FIFO 状态寄存器。	0x00
UFSTAT1	0x7F005418	读	UART 1 通道 FIFO 状态寄存器。	0x00
UFSTAT2	0x7F005818	读	UART 2 通道 FIFO 状态寄存器。	0x00
UFSTAT3	0x7F005C18	读	UART 3 通道 FIFO 状态寄存器。	0x00

位名称	位	描述	初始状态
保留	[15]	保留。	0
Tx FIFO Full	[14]	无论何时发送 FIFO 满时，该位自动置 1。  0 = 0 字节 ≤ Tx FIFO 中的数据 ≤ 63 字节  1 = Tx FIFO 中的数据满	0
Tx FIFO Count	[13: 8]	Tx FIFO 数据中的数量。	0
Reserved	[7]	保留。	
Rx FIFO Full	[6]	无论何时接收 FIFO 满时，该位自动置 1。  0 = 0 字节 ≤ Tx FIFO 数据 ≤ 63 字节  1 = Rx FIFO 中的数据满	0
Rx FIFO Count	[5: 0]	Rx FIFO 数据中的数量。	0

31.4.8. UART Modem 状态寄存器

UART 模块有两个 UART Modem 状态寄存器：UMSTAT0 和 UMSTAT1。

寄存器	地址	读/写	描述	复位值
UMSTAT0	0x7F00501C	读	UART0 通道 Modem 状态寄存器。	0x0
UMSTAT1	0x7F00541C	读	UART1 通道 Modem 状态寄存器。	0x0
Reserved	0x7F00581C	-	保留。	未定义
Reserved	0x7F005C1C	-	保留。	未定义

位名称	位	描述	初始状态
Reserved	[7: 5]	保留。	000
Delta CTS	[4]	该位指示输入到 S3C6410 的 nCTS 信号自从上次读后是否已经改变状态。（如图 6-56 所示） 0 = 没有改变	0
Reserved	[3:1]	保留。	00
Clear to Send	[0]	0 = CTS 信号没有改变(nCTS 引脚为高电平)。 1 = CTS 信号改变(nCTS 引脚为低电平)。	0

nCTS 和 Delta CTS 时序表的显示，如图 31-6 所示。

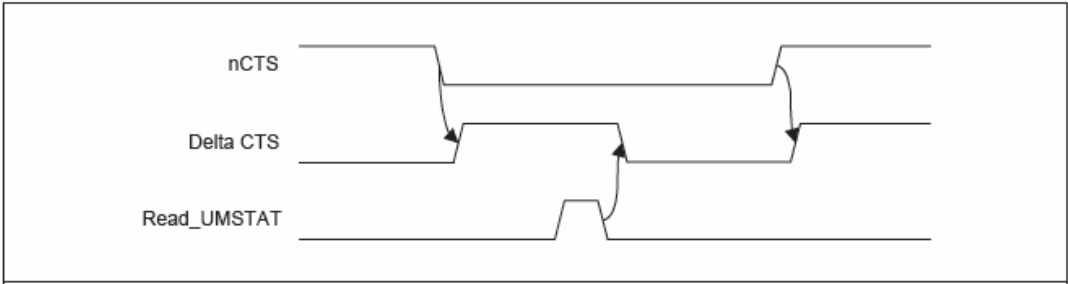


图 31-6 nCTS 和 Delta CTS 时序表

31.4.9. UART 发送缓冲寄存器（保存寄存器和 FIFO 寄存器）

UART 模块有四个发送缓冲寄存器寄存器：UTXH0、UTXH1、UTXH2 和 UTXH3。UTXHn 有一个 8 位数据作为发送数据。

寄存器	地址	R/写	描述	复位值
UTXH0	0x7F005020	写	UART 0 通道发送缓冲寄存器。	-
UTXH1	0x7F005420	写	UART 1 通道发送缓冲寄存器。	-
UTXH2	0x7F005820	写	UART 2 通道发送缓冲寄存器。	-
UTXH3	0x7F005C20	写	UART 3 通道发送缓冲寄存器。	-

位名称	位	描述	初始值
TXDATAn	[7:0]	UARTn 的发送数据。	-

#### 31.4.10 .UART 接收缓冲寄存器（保存寄存器和 FIFO 寄存器）

UART 模块有四个接收缓冲寄存器寄存器：URXH0、URXH1、URXH2 和 URXH3。URXHn 有一个 8 位数据作为发送数据。

寄存器	地址	读/写	描述	复位值
URXH0	0x7F005024	读	UART 0 通道接收缓冲寄存器。	-
URXH1	0x7F005424	读	UART 1 通道接收缓冲寄存器。	-
URXH2	0x7F005824	读	UART 2 通道接收缓冲寄存器。	-
URXH3	0x7F005C24	读	UART 3 通道接收缓冲寄存器。	-

URXHn	位	描述	初始状态
RXDATAn	[7:0]	UARTn 的接收数据。	-

注意：当溢出错误产生时，URXHn 必须被读。否则，即使该 UERSTATn 溢出错误位清 0，下一个接收数据也会产生溢出错误。

#### 31.4.11. UART 波特率分频寄存器

UART 模块有四个波特率分频寄存器：UBRDIV0、UBRDIV1、UBRDIV2 和 UBRDIV3，如表 6-90 所示。UBRDIVn 中的值决定串行 Tx/Rx 时钟波特率，如下：

$$\text{DIV\_VAL} = \text{UBRDIVn} + (\text{UDIVSLOTn 中 1 的量})/16$$

$$\text{DIV\_VAL} = (\text{PCLK} / (\text{b/s} \times 16)) - 1$$

$$\text{DIV\_VAL} = (\text{EXT\_UCLK0} / (\text{b/s} \times 16)) - 1$$

或者

$$\text{DIV\_VAL} = (\text{EXT\_UCLK1} / (\text{b/s} \times 16)) - 1$$

除数的范围为 1 到 (216 - 1)，并且 UEXTCLK 应该比 PCLK 小。

利用 UDIVSLOT, 能够得到更准确的波特率。例如，如果波特率是 115200 b/s PCLK、EXT\_UCLK0 或

EXT\_UCLK1 是 40 MHz ， UBRDIVn 和 UDIVSLOTn 是：

$$\text{DIV\_VAL} = (40000000 / (115200 \times 16)) - 1$$
$$= 21.7 - 1$$
$$= 20.7$$

UBRDIVn = 20 (DIV\_VAL 的整数部分 )

(UDIVSLOTn 中 1 的数量)/16 = 0.7

这时， (UDIVSLOTn 中 1 的数量) = 11

因此， UDIVSLOTn 为 16’ b1110\_1110\_1110\_1010 或者 16’ b0111\_0111\_0111\_0101 等。

UDIVSLOTn 选择如下表 31-5 所示：

表 31-5 UDIVSLOTn

Num of 1’ s	UDIVSLOTn	Num of 1’ s	UDIVSLOTn
0	0x0000(0000_0000_0000_0000b)	8	0x5555(0101_0101_0101_0101b)
1	0x0080(0000_0000_0000_1000b)	9	0xD555(1101_0101_0101_0101b)
2	0x0808(0000_1000_0000_1000b)	10	0xD5D5(1101_0101_1101_0101b)
3	0x0888(0000_1000_1000_1000b)	11	0xDDD5(1101_1101_1101_0101b)
4	0x2222(0010_0010_0010_0010b)	12	0xDDDD(1101_1101_1101_1101b)
5	0x4924(0100_1001_0010_0100b)	13	0xDFDD(1101_1111_1101_1101b)
6	0x4A52(0100_1010_0101_0010b)	14	0xDFDF(1101_1111_1101_1111b)
7	0x54AA(0101_0100_1010_1010b)	15	0xFFDF(1111_1111_1101_1111b)

31. 4. 12. 波特率错误容限

UART 帧错误率应当限制在1. 87%(3/160) 以内。

$$t_{UPCLK} = (UBRDIVn + 1) \times 16 \times 1\text{帧} / PCLK$$
  $t_{UPCLK}$ :实际 UART时钟。

$$t_{EXTUARTCLK} = 1\text{帧} / \text{波特率}$$
  $t_{EXTUARTCLK}$ : 理想 UART 时钟。

$$\text{UART错误} = (t_{UPCLK} - t_{EXTUARTCLK}) / t_{EXTUARTCLK} \times 100\%。$$

注：1FRAME = START 位+DATA 位+ PARITY 位+STOP 位。

寄存器	地址	读/写	描述	复位值
UBRDIV0	0x7F005028	读/写	波特率分频寄存器 0。	0x0000
UBRDIV1	0x7F005428	读/写	波特率分频寄存器 1。	0x0000
UBRDIV2	0x7F005828	读/写	波特率分频寄存器 2。	0x0000
UBRDIV3	0x7F005C28	读/写	波特率分频寄存器 3。	0x0000

UBRDIV n	位	描述	初始状态
UBRDIV	[15:0]	波特率分频值。UBRDIVn >0	-

寄存器	地址	读/写	描述	复位值
UDIVSLOT0	0x7F00502C	读/写	波特率分频寄存器 0。	0x0000
UDIVSLOT1	0x7F00542C	读/写	波特率分频寄存器 1。	0x0000
UDIVSLOT2	0x7F00582C	读/写	波特率分频寄存器 2。	0x0000
UDIVSLOT3	0x7F005C2C	读/写	波特率分频寄存器 3。	0x0000

UDIVSLOT n	位	描述	初始状态
UDIVSLOT	[15:0]	选择由 (UBRDIV + 2) 产生时钟产生分频时钟源的插槽。	–

### 31. 4. 13. UART 中断处理寄存器

中断处理寄存器包括产生中断的信息。

寄存器	地址	读/写	描述	复位值
UINTP0	0x7F005030	读/写	UART 0 通道中断处理寄存器。	0x0
UINTP1	0x7F005430	读/写	UART 1 通道中断处理寄存器。	0x0
UINTP2	0x7F005830	读/写	UART 2 通道中断处理寄存器。	0x0
UINTP3	0x7F005C30	读/写	UART 3 通道中断处理寄存器。	0x0

UINTPn	位	描述	初始状态
MODEM	[3]	产生 Modem 中断。	0x0
TXD	[2]	产生发送中断。	0x0
ERROR	[1]	产生错误中断。	0x0
RXD	[0]	产生接收中断	0x0

当 4 位有一位置位逻辑 ‘1’ 时，UART 每个通道都产生中断。 在中断服务程序这个寄存器被清理。 可以通过置 ‘1’ 在指定的位来清理 UINTP 特殊的位。

### 31. 4. 14. UART 中断源处理寄存器

中断源处理寄存器包含产生中断的信息（不管中断屏蔽为何值）。

寄存器	地址	读/写	描述	复位值
UINTSP0	0x7F005034	读/写	中断源处理寄存器 0。	0x0
UINTSP1	0x7F005434	读/写	中断源处理寄存器 1。	0x0
UINTSP2	0x7F005834	读/写	中断源处理寄存器 2。	0x0
UINTSP3	0x7F005C34	读/写	中断源处理寄存器 3。	0x0

UINTSPn	位	描述	初始状态
MODEM	[3]	产生 Modem 中断。	00
TXD	[2]	产生发送中断。	0
ERROR	[1]	产生错误中断。	0
RXD	[0]	产生接收中断。	0

31. 4. 15. UART 中断屏蔽寄存器

中断屏蔽寄存器包含屏蔽中断信息。如果一个特殊位被置为‘1’，尽管相应的中断产生，但不产生到中断控制器的中断请求信号（在这中情况下，UINTSPn 寄存器相应位置为‘1’）。如果屏蔽为是‘0’，中断请求能从相应的中断源得到响应（在这中情况下，UINTSPn 寄存器相应位置为‘1’）。

寄存器	地址	读/写	描述	复位值
UINTM0	0x7F005038	读/写	UART 0 通道中断屏蔽寄存器。	0x0
UINTM1	0x7F005438	读/写	UART 1 通道中断屏蔽寄存器。	0x0
UINTM2	0x7F005838	读/写	UART 2 通道中断屏蔽寄存器。	0x0
UINTM3	0x7F005C38	读/写	UART 3 通道中断屏蔽寄存器。	0x0

UINTMn	位	描述	初始状态
MODEM	[3]	屏蔽 Modem 中断。	0
TXD	[2]	屏蔽发送中断。	0
ERROR	[1]	屏蔽错误中断。	0
RXD	[0]	屏蔽接收中断。	0

31.5 UART 接口应用举例

UART 接口的应用十分广泛，是学习嵌入式开发所必不可少的内容，下面是 UART 接口在 ARM11 处理器中的实例应用。针对以上对 UART 接口特性及操作的理解，再参照各个寄存器功能的描述，具体的程序代码分析如下：

```
UART 接口的配置：UART_Config 函数主要功能是由用户选择建立 UART 接口。
输入：NONE
输出：NONE
u8 UART_Config(void)
{
    u8 cCh;
```

```
s32 iNum = 0;
```

```
volatile UART_CON *pUartCon;
```

```
g_uOpClock = 0;
```

```
// 选择通道
```

```
printf("Note : [D] mark means default value. If you press ENTER key, default value is selected.\n");
```

```
printf("Select Channel(0~3) [D=0] : ");
```

```
cCh = (u8)GetIntNum();
```

```
if ( cCh>3 )    cCh = 0;           // 默认 UART0
```

```
pUartCon = &g_AUartCon[cCh];
```

```
printf("\n\nConnect PC[COM1 or COM2] and UART%d of S3C6410 with a serial cable for test!!! \n", cCh);
```

```
//设置其他选项
```

```
printf("\nSelect Other Options\n 0. Nothing[D]   1.Send Break Signal   2. Loop Back Mode
```

```
   \n Choose : ");
```

```
switch(GetIntNum())
```

```
{
```

```
    default :
```

```
        pUartCon->cSendBreakSignal = 0x0;
```

```
        pUartCon->cLoopTest = 0x0;
```

```
        break;
```

```
    case 1 :
```

```
        pUartCon->cSendBreakSignal = 1;
```

```
        return cCh;
```

```
    case 2 :
```

```
        pUartCon->cLoopTest = 1;
```

```
        break;
```

```
}
```

```
//设置奇偶模式
```

```
printf("\nSelect Parity Mode\n 1. No parity[D] 2. Odd 3. Even 4. Forced as '1' 5. Forced as '0' \n Choose : ");
```

```
switch(GetIntNum())
```

```
{
```

```
    default :
```

```
        pUartCon->cParityBit = 0;
```

```
        break;
```

```
    case 2 :
```

```
        pUartCon->cParityBit = 4;
```

```
        break;
```

```
    case 3 :
```

```
        pUartCon->cParityBit = 5;
```

```
        break;
```

```
    case 4 :
```

```
        pUartCon->cParityBit = 6;
```

```
        break;
```

```
    case 5 :
```

```
        pUartCon->cParityBit = 7;
```

```
        break;
```

```
}
```

```
//设置停止位的数量
```

```
printf("\n\nSelect Number of Stop Bit\n 1. One stop bit per frame[D] 2. Two stop bit per frame");
```

```
switch(GetIntNum())
```

```
{
```

```
    default :
```

```
        pUartCon->cStopBit = 0;
```



```

                break;
        case 2 :
                pUartCon->cStopBit = 1;
                break;
    }

```

//设置字长度

```
printf("\n\nSelect Word Length\n 1. 5bits 2. 6bits 3. 7bits 4. 8bits \n Choose : ");
```

```
switch(GetIntNum())
```

```

{
    case 1 :
        pUartCon->cDataBit = 0;
        break;
    case 2 :
        pUartCon->cDataBit = 1;
        break;
    case 3 :
        pUartCon->cDataBit = 2;
        break;
    default :
        pUartCon->cDataBit = 3;
        break;
}

```

// 设置操作时钟

```
printf("\n\nSelect Operating Clock\n 1. PCLK[D] 2. EXT_CLK0(pwm) 3. EXT_CLK1(EPLL/MPLL) \n
```

```
Choose : ");
```

```
switch (GetIntNum())
```

```

{

```

case 2 :

```
pUartCon->cOpClock = 1;
// 连接 CLKOUT 和 UEXTCLK
printf("\nInput PWM EXT_CLK by Pulse Generater\n");
printf("How much CLK do you input through the pwmECLK?");
printf("Mhz : ");
g_uOpClock = GetIntNum()*1000000;
GPIO_SetFunctionEach(eGPIO_F,eGPIO_13,2);
break;
```

case 3 :

```
pUartCon->cOpClock = 3;
printf("\nSelect Clock SRC\n 1.EPLL  2.MPLL \n Choose: ");
switch(GetIntNum())
{
    case 1:
        SYSC_SetPLL(eEPLL,32,1,1,0);    //EPLL=192Mhz
        SYSC_ClkSrc(eEPLL_FOUT);
        SYSC_ClkSrc(eUART_MOUTEPLL);
        SYSC_CtrlCLKOUT(eCLKOUT_EPLLOUT,0);
        g_uOpClock = CalcEPLL(32,1,1,0);
        printf("EPLL = %dMhz\n",(g_uOpClock/1000000));
        break;
    case 2:
        SYSC_ClkSrc(eMPLL_FOUT);
        SYSC_ClkSrc(eUART_DOUTMPLL);
        Delay(100);
        g_uOpClock = (u32)g_MPLL/2;
        printf("MPLL = %dMhz\n",(g_uOpClock/1000000));
        break;
```

```

        default:
            SYSC_ClkSrc(eMPLL_FOUT);
            SYSC_ClkSrc(eUART_DOUTMPLL);
            Delay(100);
            g_uOpClock = (u32)g_MPLL/2;
            printf("MPLL = %dMhz\n", (g_uOpClock/1000000));
            break;
    }
    break;
default :
    pUartCon->cOpClock = 0; // PCLK
    break;
}

// 选择 UART 或 IrDA 1.0
printf("\n\nSelect External Interface Type\n 1. UART[D]    2. IrDA mode\n Choose : ");
if (GetIntNum() == 2)
    pUartCon->cSelUartIrda = 1;        // IrDA 模式
else
    pUartCon->cSelUartIrda = 0;        // URAT 模式

// 设置波特率
printf("\n\nType the baudrate and then change the same baudrate of host, too.\n");
printf(" Baudrate (ex 9600, 115200[D], 921600) : ");
pUartCon->uBaudrate = GetIntNum();
if ((s32)pUartCon->uBaudrate == -1)
    pUartCon->uBaudrate = 115200;

// 选择 UART 操作模式

```

```
printf("\n\nSelect Operating Mode\n 1. Interrupt[D]   2. DMA\n Choose : ");
```

```
if (GetIntNum() == 2)
```

```
{
```

```
    pUartCon->cTxMode = 2;           // DMA0 模式
```

```
    pUartCon->cRxMode = 3;           // DMA1 模式
```

```
}
```

```
else
```

```
{
```

```
    pUartCon->cTxMode = 1;           // Int 模式
```

```
    pUartCon->cRxMode = 1;           // Int 模式
```

```
}
```

```
// 选择 UART FIFO 模式
```

```
printf("\n\nSelect FIFO Mode (Tx/Rx[byte])\n 1. no FIFO[D]   2. Empty/1   3. 16/8   4. 32/16   5. 48/32\n Choose : ");
```

```
iNum = GetIntNum();
```

```
if ( (iNum>1)&&(iNum<6) )
```

```
{
```

```
    pUartCon->cEnableFifo = 1;
```

```
    pUartCon->cTxTrig = iNum -2;
```

```
    pUartCon->cRxTrig = iNum -2;
```

```
}
```

```
else
```

```
{
```

```
    pUartCon->cEnableFifo = 0;
```

```
}
```

```
// 选择 AFC 模式使能/禁用
```

```
printf("\n\nSelect AFC Mode\n 1. Disable[D]   2. Enable\n Choose : ");
```

```

if (GetIntNum() == 2)
{
    pUartCon->cAfc = 1;          // AFC 模式使能
    printf("Select nRTS trigger level(byte)\n 1. 63[D]    2. 56    3. 48    4. 40    5. 32    6. 24    7. 16
        8. 8\n Choose : ");
    iNum = GetIntNum();

    if ( (iNum>1)&&(iNum<9) )
        pUartCon->cRtsTrig = iNum -1;
    else
        pUartCon->cRtsTrig = 0;    // 默认 63 字节
}
else
{
    pUartCon->cAfc = 0;          // AFC 模式禁用
}
}

#if 1
printf("SendBreakSignal=%d\n",pUartCon->cSendBreakSignal);
printf("Brate = %d\n,          SelUartIrda = %d\n,          Looptest= %d\n,          Afc = %d\n,
EnFiFO = %d\n,          OpClk = %d\n,          Databit = %d\n,          Paritybit = %d\n,          Stopbit =
%d\n,          Txmode = %d\n,          TxTrig = %d\n,          RxMode = %d\n,          RxTrig = %d\n,
RtsTrig = %d\n,          SendBsig = %d\n",pUartCon->uBaudrate
,pUartCon->cSelUartIrda,
pUartCon->cLoopTest,
pUartCon->cAfc,
pUartCon->cEnableFifo,
pUartCon->cOpClock,
pUartCon->cDataBit,
pUartCon->cParityBit,

```

```
    pUartCon->cStopBit,  
    pUartCon->cTxMode,  
    pUartCon->cTxTrig,  
    pUartCon->cRxMode,  
    pUartCon->cRxTrig,  
    pUartCon->cRtsTrig,  
    pUartCon->cSendBreakSignal);  
#endif  
    return cCh;  
}
```

## 32 PWM 定时器

该 S3C6410 RISC 微处理器由五个 32 位定时器组成。这些计时器用来产生内部中断到 ARM 子系统。此外，定时器 0, 1, 2 和 3 包含一个 PWM 功能（脉宽调制），它可以驱动外部的 I/O 信号。定时器 0 上的 PWM 能够产生一个可选的死区发生器。它可能被用来支持大量的通用装置。定时器 4 只是一个没有输出引脚的内部定时器。

该定时器的时钟频率通常是 APB-PCLK 分频的版本。定时器 0 和 1 共享一个可编程的 8 位预定标器，它提供第一层为 PCLK。计时器 2, 3, 4 共享不同的 8 位预定标器。每个定时器拥有它自己的时钟分频器，个别时钟分频器提供有一个二级时钟版本（预定标器由 2, 4, 8 或 16 进行分频）。另外，定时器从外部引脚选择时钟来源。定时器 0 和 1 可以选择外部时钟 TCLK0。定时器 2, 3 和 4 可以选择外部时钟 TCLK1。每个定时器有自己的 32 位向下计数器，被定时器时钟驱动。下数计数器是最初被加载来自定时器计数缓冲寄存器（TCNTBn）。当下数计数器达到零，定时器产生中断请求通知 CPU，定时器操作完成。当定时器下数计数器达到零，相应的 TCNTBn 能被自动重新进入下数计数器的下一个周期的开始。然而，如果定时器停止，例如，在定时器运行模式下，通过清除定时器的使能位 TCONn，TCNTBn 的值将不被加载到计数器中。

脉冲宽度调制功能（PWM）使用 TCMPBn 寄存器的值，当下数计数器的值匹配于定时器控制器逻辑中的比较寄存器的值时，定时器控制器逻辑改变输出标准。因此，比较寄存器决定一个 PWM 输出的打开时间（或关闭时间）。TCNTBn 和 TCMPBn 寄存器是双缓冲，允许定时器参数在循环中更新。直到目前的定时器周期完成，新的值才能生效。

一个 PWM 的周期的简单的例子如图 32-1 所示。

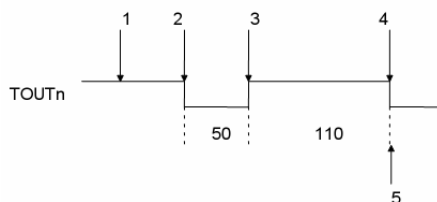


图 32-1 PWM 周期方框图的简单例子

- (1) 初始化带有 160 (50+110) 的 TCNTBn 和带有 110 位的 TCMPBn。

TCNTBn 的 160 值载入下数计数器, 输出为低电平。

(4) 当下数计数器达到零, 产生中断请求。

(5) 同时, 通过 TCNTBn (重新启动循环) 下数计数器是自动重新载入。

The diagram illustrates the internal architecture of the timer module. It consists of four main functional blocks: INTF\_PWM, COUNT\_PWM, CONTROL\_PWM, and TIMER\_REQ. The data and control flow is as follows:

- INTF\_PWM** receives **PCLK** and **APB** signals. It outputs to **COUNT\_PWM**.
- COUNT\_PWM** receives **PCLK** and outputs to **CONTROL\_PWM**.
- CONTROL\_PWM** receives **PCLK** and outputs **TOUT0..4**. It also outputs to **TIMER\_REQ**.
- TIMER\_REQ** receives **PCLK**, **DMAAC**, **INTRGEN\_SEL**, and a signal from **CONTROL\_PWM**. It outputs **DMAREQ** and **INT0..4**.

图 32-2 PWM 定时器的结构框图

The diagram illustrates the internal architecture of the TCC3888, showing four parallel channels. Each channel starts with a prescaler (8BIT) and a frequency divider (1/2, 1/4, 1/8, 1/16) that takes a clock signal (CLK) as input. The output of the frequency divider is fed into a 5-bit mixer (MIX). The output of the mixer is then fed into a control logic block (控制逻辑0, 控制逻辑1, 控制逻辑2, 控制逻辑3, 控制逻辑4). The control logic blocks are also fed by TCMPB0, TCNTB0, TCMPB1, TCNTB1, TCMPB2, TCNTB2, TCMPB3, TCNTB3, and TCNTB4. The output of the control logic blocks is then fed into a dead zone generator (死区发生器) and an output comparator. The output of the dead zone generator is fed into the output comparator. The output of the output comparator is then fed into the output driver (TOUT0, TOUT1, No pin). The output driver is also fed by a dead zone (死区) signal.



图 32-3 PWM 定时器的时钟树状框图

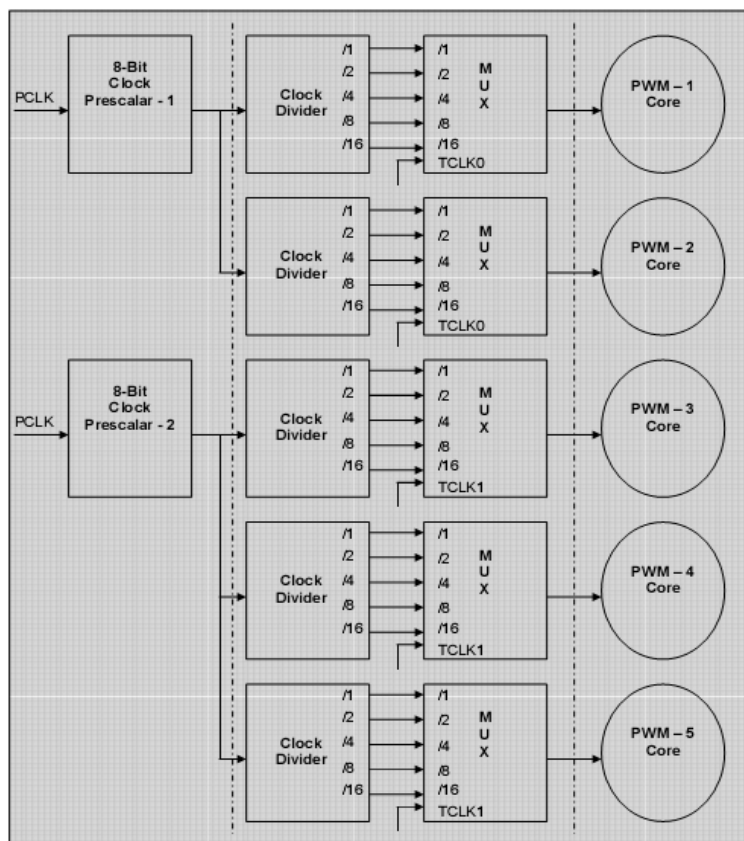


图 32-4 PWM 定时器的详细时钟树状框图

## 32.1 PWM 定时器的特性

PWM 支持的特性分别如下：

- 五个 32 位的定时器。
- 两个 8 位的时钟预定标器，提供第一级版本，用于 PCLK，五个时钟分频器和多路复用器提供第二级版本，用于分频器时钟和两个外部时钟。
- 可编程的时钟，用于个别 PWM 通道选择逻辑。
- 四个独立的脉宽调制通道，具有可编程控制的作用和极性。
- 静态配置：脉宽调制为停止状态。
- 动态配置：脉宽调制正在运行。

- 支持自动重新装载模式和一次触发脉冲模式。
- 支持两个外部输入到启动 PWM。
- 死区发生器在两个 PWM 上输出。
- 支持 DMA 传输。
- 可选脉冲或中断级产生。

PWM 有两个运行模式：

（1）自动重新载入模式

连续的 PWM 脉冲产生，基于编程作用循环和极性。

（2）一次触发脉冲模式

只有一个 PWM 脉冲的产生是基于编程作用循环和极性的。

PWM 的控制功能，提供 18 个特殊功能寄存器。PWM 是一个可编程输出，双重时钟输入 AMBA 次模块并连接先进的外设总线（APB）。PWM 内的 18 个特殊功能寄存器通过 APB 处理被存取。

## 32.2 PWM 的操作

### 1. 预定标器与分配器

一个 8 位预定标器和 3 位分配器提出以下输出频率：如表 32-1 所示。

表 32-1 定时器最大、最小输出周期

4 位分配器设置	最低分辨率 (prescaler=0)	最高分辨率 (prescaler=255)	最大间隔 (TCNTBn=65535)
1/1 (PCLK=66MHz)	0.015 μs (66.0MHz)	3.87 μs (258kHz)	0.23s
1/2 (PCLK=66MHz)	0.031 μs (33.0MHz)	7.75 μs (129kHz)	0.50s
1/4 (PCLK=66MHz)	0.060 μs (16.5MHz)	15.5 μs (64.5kHz)	1.02s
1/8 (PCLK=66MHz)	0.121 μs (8.25MHz)	31.0 μs (32.2kHz)	2.03s
1/16 (PCLK=66MHz)	0.242 μs (4.13MHz)	62.1 μs (16.1kHz)	4.07s

### 2. 定时器的基本操作

其操作的基本流程，如图 32-5 所示。

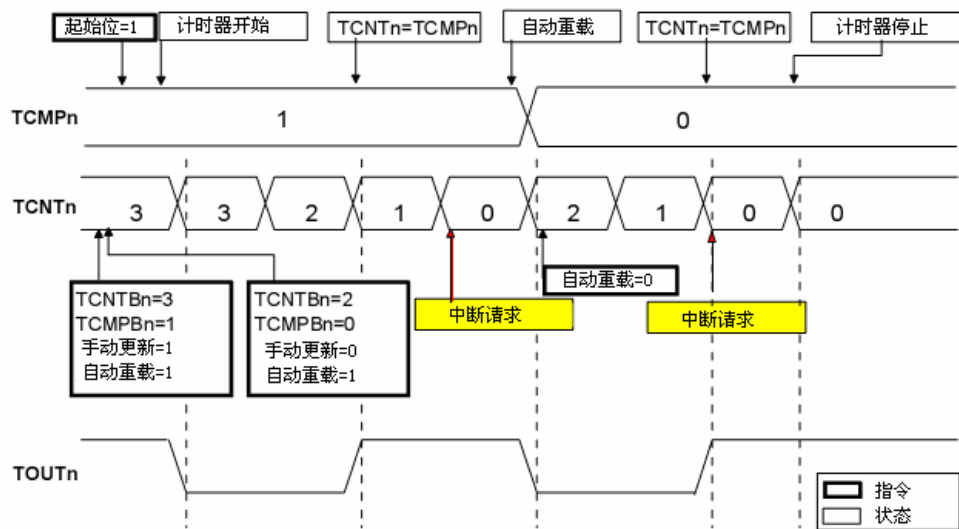


图 32-5 定时器的操作

定时器（除定时器通道 5）包括 TCMPBn、TCMPn、TCMPBn 和 TCMPn。当定时器置 '0' 时，TCMPBn 和 TCMPBn 载入 TCNTn 和 TCMPn 中。当 TCNTn 置 '0' 时，如果中断信号启动，则将产生中断请求。（TCNTn 和 TCMPn 是内部寄存器的名称。可以从 TCNT0n 寄存器中读取 TCNTn 寄存器。

### 3. 自动重新加载和双缓冲

定时器有一个双缓冲功能，在没有停止当前定时器操作基础上，它可以改变加载数值以适合于下一定时器的操作。虽然新的定时器值被设定，但当前定时器的操作已经被成功完成。定时器的值可以被写入 TCNTBn（定时器计数缓冲寄存器）以及定时器的当前计数器值从 TCNT0n 中被读取（定时器计数观察寄存器）。如果读 TCNTBn，这个值是下一个定时器的重载值不是当前计数器的状态。

自动重新载入是一个操作，当 TCNTn 置 "0" 时，它复制 TCNTBn 到 TCNTn。值写入 TCNTBn，当 TCNTn 达到 "0" 并自动重新启动时，它只能被加载到 TCNTn 中。如图 32-6 所示，描述了双缓冲功能的框图实例。

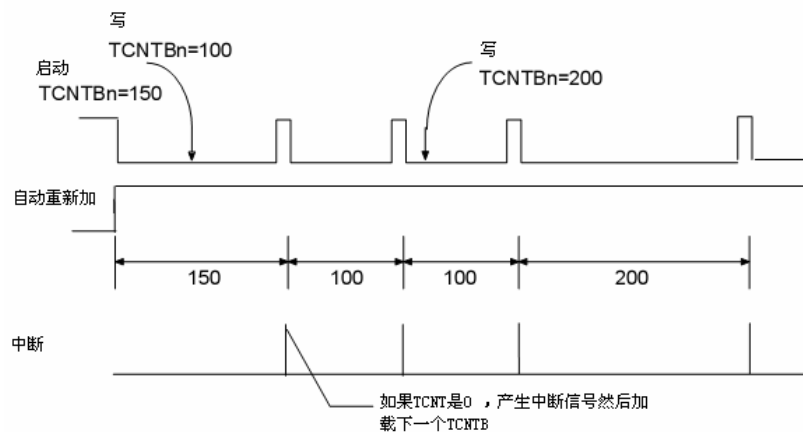


图 32-6 双缓冲功能框图实例

#### 4. 定时器操作实例

显示下列程序的结果，如图 32-7 所示。

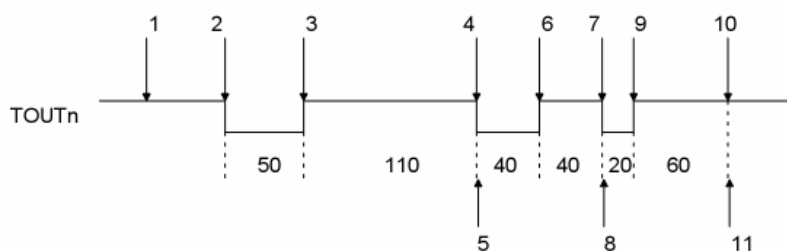


图 32-7 定时器的实例图

- (1) 启用自动重新载入功能。设置 TCNTBn 为 160 (50+110)，TCMPBn 为 110。设置手动更新位和反转位 (开/关)。该手动更新位设置 TCNTn 和 TCMPn 为 TCNTBn 和 TCMPBn 的值。然后，设置 TCNTBn 和 TCMPBn 作为 80 (40+40) 和 40，以确定下一个重新载入的值。
- (2) 启动定时器即设置启动位和关闭手动更新位。
- (3) 当 TCNTn 和 TCMPn 有相同值时，TOUTn 的逻辑电平由低变为高。
- (4) 当 TCNTn 达到“0”时，TCNTn 和 TCNTBn 自动重装。在同一时间产生中断请求。
- (5) 在 ISR (中断服务程序) 中，TCNTBn 和 TCMPBn 被设置为 80 (60+20) 和 60，它被用于下一个持续时间。
- (6) 当 TCNTn 和 TCMPn 有相同值时，TOUTn 的逻辑电平由低变为高。

## 6. 输出电平控制

如图 32-9 所示，显示了逆变器的开/关。

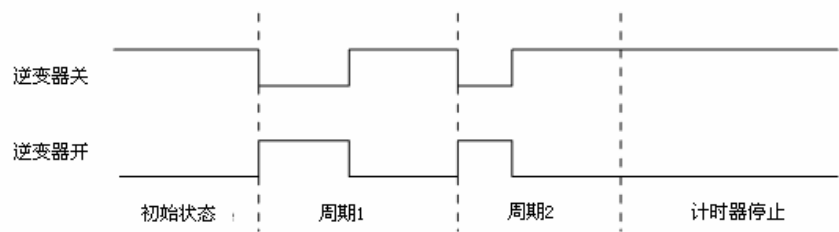


图 32-9 逆变器的开/关

下列方法是用来控制 TOUT，作为高电平或低电平。（假定逆变器是关闭状态）

（1）关闭自动重新载入位。然后，TOUTn 达到高电平，同时定时器被停止，之后 TCNTn 达到 0。建议用这种方法。

（2）通过清空定时器的开始/停止位为 0 来停止定时器。如果  $TCNTn \leq TCMPn$ ，输出高电平。如果  $TCNTn > TCMPn$ ，输出低电平。

（3）在 TCON 中，通过逆变器的开/关位，TOUTn 可以被转换。逆变器移除额外的电路以调节输出电平。

### 7. 死区发生器

该死区是用于电源设备的 PWM 控制。这种功能用于在关闭一个开关设备和打开另一个开关设备之间插入。这个定时器禁止两个开关设备同时转向，即很短的时间。其死区特性开启时输出波形比较，如图 32-10 所示。

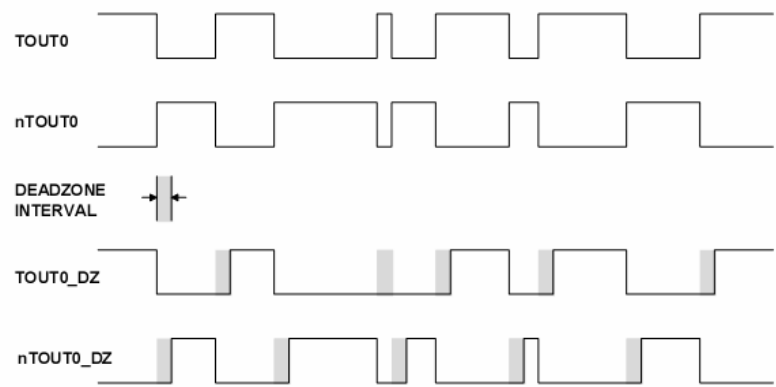


图 32-10 死区特性开启时输出波形比较

## 8. DMA 请求模式

在下计数器的周期结束时，代替发送中断，定时器可以配置成发送一个 DMA 请求信号到 DMA 的一个通道中。这种模式是让 DMA 传输在源和目标之间以固定的时间段发生。

**DMA 请求过程：**定时器可以在任意时间产生 DMA 请求，并且保持 DMA 请求信号（nDMA\_REQ）为低，直到定时器收到 ACK 信号。当定时器收到 ACK 信号时，它使请求信号变得无效。只有一个定时器的时间可以被设定为 DMA 请求源。设置 DMA 模式位（在 TCFG1 寄存器中）决定了定时器产生 DMA 请求。如果 DMA 请求模式设定一个特定的定时器，即计时器发送 DMA 请求，并产生正常的 ARM 中断请求。而其它的定时器未被设定为 DMA 模式，仍然可以生成正常的 ARM 中断。如果没有的计时器被设定在 DMA 模式，那么它们都可以产生正常的中断。

如图 32-11 所示，显示直到 DMA 的发送的 ACK 信号，如何 DMA 请求将仍然有效。

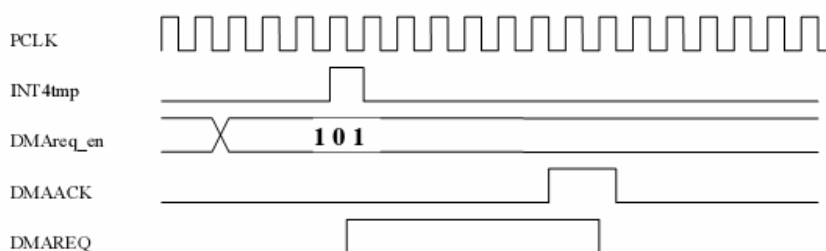


图 32-11 DMA 操作

如表 32-2 所示，显示在下数计数器中，完成的每个定时器的作用。

表 32-2 DMA 模式和 DMA 请求配置

DMA 模式	DMA 请求	INT0	INT1	INT2	INT3	INT4
000	无选择	ON	ON	ON	ON	ON
001	定时器 0	ON	ON	ON	ON	ON
010	定时器 1	ON	ON	ON	ON	ON
011	定时器 2	ON	ON	ON	ON	ON
100	定时器 3	ON	ON	ON	ON	ON
101	定时器 4	ON	ON	ON	ON	ON
110	无选择	ON	ON	ON	ON	ON

## 9. 定时器中断的产生

通过控制 ‘INTRGEN\_SEL’ 端口状态，PWMTIMER 提供产生脉冲中断和电平中断的灵活性。当

‘INTRGEN\_SEL’ 端口状态是逻辑 1，将产生可选电平或可选脉冲中断。在 PWMTIMER 内部写入具体值到 ‘TINT\_CSTAT’ 寄存器，控制中断产生。中断产生是基于 ‘TINT\_CSTAT’ 寄存器中设置的值。

## 32.3 编程模型

为控制和观察 PWM 的状态，可使用下面的寄存器：

- (1) TCFG0: 时钟预定标器和死区结构。
- (2) TCFG1: 时钟多路复用器和 DMA 模式的选择。
- (3) TCON: 定时器控制寄存器。
- (4) TCNTB0: 定时器 0 计数缓冲寄存器。
- (5) TCMPB0: 定时器 0 比较缓冲寄存器。
- (6) TCNT00: 定时器 0 计数观察寄存器。
- (7) TCNTB1: 定时器 1 计数缓冲寄存器。
- (8) TCMPB1: 定时器 1 比较缓冲寄存器。
- (9) TCNT01: 定时器 1 计数观察寄存器。
- (10) TCNTB2: 定时器 2 计数缓冲寄存器。
- (11) TCMPB2: 定时器 2 比较缓冲寄存器。
- (12) TCNT02: 定时器 2 计数观察寄存器。
- (13) TCNTB3: 定时器 3 计数缓冲寄存器。
- (14) TCMPB3: : 定时器 3 比较缓冲寄存器。
- (15) TCNT03: 定时器 3 计数观察寄存器。
- (16) TCNTB4: 定时器 4 计数缓冲寄存器。
- (17) TCNT04: 定时器 4 计数观察寄存器。
- (18) TINT\_CSTAT: 定时器中断控制和状态寄存器。



# 32. 4 特殊功能寄存器

## 32. 4. 1. 寄存器图表

寄存器	偏移量	读/写	描述	复位值
TCFG0	0x7F006000	读/写	定时器配置寄存器 0 时配置两个 8 位 预定标器和死区长度。	0x0000_0101
TCFG1	0x7F006004	读/写	定时器配置寄存器 1 时，5-MUX 和 DMA 模式选择寄存器。	0x0000_0000
TCON	0x7F006008	读/写	定时器控制寄存器。	0x0000_0000
TCNTB0	0x7F00600C	读/写	定时器 0 计数缓冲器。	0x0000_0000
TCMPB0	0x7F006010	读/写	定时器 0 比较缓冲寄存器。	0x0000_0000
TCNT00	0x7F006014	读	定时器 0 计数观察寄存器。	0x0000_0000
TCNTB1	0x7F006018	读/写	定时器 1 计数缓冲器。	0x0000_0000
TCMPB1	0x7F00601c	读/写	定时器 1 比较缓冲寄存器。	0x0000_0000
TCNT01	0x7F006020	读	定时器 1 计数观察寄存器。	0x0000_0000
TCNTB2	0x7F006024	读/写	定时器 2 计数缓冲器。	0x0000_0000
TCMPB2	0x7F006028	读/写	定时器 2 比较缓冲寄存器。	0x0000_0000
TCNT02	0x7F00602c	读	定时器 2 计数观察寄存器。	0x0000_0000
TCNTB3	0x7F006030	读/写	定时器 3 计数缓冲器。	0x0000_0000
TCMPB3	0x7F006034	读/写	定时器 3 比较缓冲寄存器。	0x0000_0000
TCNT03	0x7F006038	读	定时器 3 计数观察寄存器。	0x0000_0000
TCNTB4	0x7F00603c	读/写	定时器 4 计数缓冲器。	0x0000_0000
TCNT04	0x7F006040	读	定时器 4 计数观察寄存器。	0x0000_0000
TINT_CSTAT	0x7F006044	读/写	定时器中断控制和状态寄存器。	0x0000_0000

### 32.4.2. TCFG0 (定时器配置寄存器 0)

寄存器	偏移量	读/写	描述	复位值
TCFG0	0x7F006000	读/写	定时器配置寄存器 0，可配置两个 8 位预定标器和死区长度。	0x0000_0101

定时器输入时钟频率 = PCLK / {预定标器的值 + 1} / {分频值}

{预定标器的值}=0~255;

{分频值}=1, 2, 4, 8, 16, TCLK

TCFG0	位	读/写	描述	初始状态
Reserved	[31:24]	读	保留	0x00
Dead zone length	[23:16]	读/写	死区的长度	0x00
Prescaler 1	[15:8]	读/写	预定标器 1 的值，用于定时器 2、3 和 4	0x01
Prescaler 0	[7:0]	读/写	预定标器 0 的值，用于定时器 0 和 1	0x01

### 32.4.3. TCFG1 (定时器配置寄存器 1)

寄存器	地址	读/写	描述	复位值
TCFG1	0x7F006004	读/写	定时器配置寄存器 1，可控制 5 MUX 和 DMA 模式选择位	0x0000_0000

TCFG1	位	读/写	描述	初始状态
Reserved	[31:24]	读	保留位。	0x00
DMA mode	[23:20]	读/写	选择 DMA 请求通道选择位：  0000 : 无选择	

Divider MUX 4	[19:16]	读/写	选择定时器 4 的 MUX 输入： 0000:1/1                      0001:1/2 0010:1/4                      0011:1/8 0100: 1/16                    0101: 外部 TCLK1	0x00
Divider MUX 3	[15:12]	读/写	选择定时器 3 的 MUX 输入： 0000:1/1                      0001:1/2 0010:1/4                      0011:1/8 0100: 1/16                    0101: 外部 TCLK1	0x00
Divider MUX 2	[11:8]	读/写	选择定时器 2 的 MUX 输入： 0000:1/1                      0001:1/2 0010:1/4                      0011:1/8 0100: 1/16                    0101: 外部 TCLK1	0x00
Divider MUX 1	[7:4]	读/写	选择定时器 1 的 MUX 输入： 0000:1/1                      0001:1/2 0010:1/4                      0011:1/8 0100: 1/16                    0101: 外部 TCLK1	0x00
Divider MUX 0	[3:0]	读/写	选择定时器 0 的 MUX 输入： 0000:1/1                      0001:1/2 0010:1/4                      0011:1/8 0100: 1/16                    0101: 外部 TCLK1	0x00

#### 32. 4. 4. TCON（定时控制寄存器）

寄存器	地址	读/写	描述	复位值
TCON	0x7F006008	读/写	定时器控制寄存器	0x0000_0000

TCON	位	读/写	描述	初始值
Reserved	[31:23]	读	保留。	0x000d

Timer 4 Auto Reload on/off	[22]	读/写	确定定时器 4 的自动加载开/关。 0 = One-shot 1 = 间隔模式(自动重载)	0x0
Timer 4 Manual Update (note)	[21]	读/写	确定定时器 4 的手动更新。 0 = 无操作 1 = 更新 TCNTB4	0x0
Timer 4 Start/Stop	[20]	读/写	确定定时器 4 的启动/停止。 0 = 停止 1 = 开始定时器 4	0x0
Timer 3 Auto Reload on/off	[19]	读/写	确定定时器 3 的自动加载开/关。 0 = One-shot 1 =间隔模式(自动重载)	0x0
Timer 3 output inverter on/off	[18]	读/写	确定定时器 3 的输出反转器开/关。 0 = 逆变器关闭 1 = 逆变器开, 用于 TOUT3	0x0
Timer 3 Manual Update (note)	[17]	读/写	确定定时器 3 的手动更新。 0 = 无操作 1=更新 TCNTB3 或 CMPB3	0x0
Timer 3 Start/Stop	[16]	读/写	确定定时器 3 的启动/停止。 0 = 停止 1 = 开始定时器 3	0x0
Timer 2 Auto Reload on/off	[15]	读/写	确定定时器 2 的自动加载开/关。 0 = One-shot 1 =间隔模式(自动重载)	0x0
Timer 2 Output Inverter on/off	[14]	读/写	确定定时器 2 的输出反转器开/关。 0 =逆变器关闭 1 =逆变器开, 用于 TOUT2	0x0
Timer 2 Manual Update (note)	[13]	读/写	确定定时器 2 的手动更新。 0 =无操作 1 = 更新 TCNTB2 或 TCMPB2	0x0
Timer 2 Start/Stop	[12]	读/写	确定定时器 2 的启动/停止。 0 = 停止 1 =开始定时器 2	0x0
Timer 1 Auto Reload on/off	[11]	读/写	确定定时器 1 的自动加载开/关。 0 = One-shot 1 =间隔模式(自动重载)	0x0
Timer 1 Output Inverter on/off	[10]	读/写	确定定时器 1 的输出反转器开/关。 0 =逆变器关闭 1 = 逆变器开, 用于 TOUT1	0x0
Timer 1 Manual Update (note)	[9]	读/写	确定定时器 1 的手动更新。 0 =无操作 1 = 更新 TCNTB1 或 TCMPB1	0x0

Timer 1 Start/Stop	[8]	读/写	确定定时器 1 的启动/停止。 0 = 停止 1 = 开始定时器 1	0x0
Reserved	[7:5]	读/写	保留。	0x0
Dead Zone Enable	[4]	读/写	确定死区的操作。 0 = 禁用 1 = 使能	0x0
Timer 0 Auto Reload on/off	[3]	读/写	确定定时器 0 的自动加载开/关。 0 = One-shot 1 = 间隔模式(自动重载)	0x0
Timer 0 output inverter on/off	[2]	读/写	确定定时器 0 的输出反转器开/关。 0 = 逆变器关闭 1 = 逆变器开, 用于 TOUT0	0x0
Timer 0 Manual Update (note)	[1]	读/写	确定定时器 0 的手动更新。 0 = 无操作 1 = 更新 TCNTB0 或 TCMPB0	0x0
Timer 0 Start/Stop	[0]	读/写	确定定时器 0 的启动/停止。 0 = 停止 1 = 开始定时器 0	0x0

#### 32.4.5. TCNTB0 (定时器 0 计数寄存器)

寄存器	地址	读/写	描述	复位值
TCNTB0	0x7F00600C	读/写	定时器 0 计数缓冲器	0x0000_0000

TCNTB0	位	读/写	描述	初始状态
Timer 0 Count Buffer	[31:0]	读/写	设置定时器 0 的计数缓冲器的值。	0x00000000

#### 32.4.6. TCMPB0 (定时器 0 比较寄存器)

寄存器	地址	读/写	描述	复位值
TCMPB0	0x7F006010	读/写	定时器 0 比较缓冲寄存器	0x0000_0000

TCMPB0	位	读/写	描述	初始状态
Timer 0 Compare Buffer	[31:0]	读/写	设置定时器 0 的比较缓冲器的值	0x00000000

#### 32.4.7. TCNT00（定时器 0 计数观察寄存器）

寄存器	地址	读/写	描述	复位值
TCNT00	0x7F006014	读	定时器 0 计数观察寄存器	0x0000_0000

TCNT00	位	读/写	描述	初始状态
Timer 0 Count Observation	[31:0]	读	设置定时器 0 计数观察寄存器的值	0x00000000

#### 32.4.8. TCNTB1（定时器 1 计数寄存器）

寄存器	地址	读/写	描述	复位值
TCNTB1	0x7F006018	读/写	定时器 1 计数缓冲器	0x0000_0000

TCNTB1	位	读/写	描述	初始状态
Timer 1 Count Buffer	[31:0]	读/写	设置定时器 1 的计数缓冲器的值	0x00000000

#### 32.4.9. TCMPB1（定时器 1 比较寄存器）

寄存器	地址	读/写	描述	复位值
TCMPB1	0x7F00601c	读/写	定时器 1 比较缓冲寄存器	0x0000_0000

TCMPB1	位	读/写	描述	初始状态
Timer 1 Compare Buffer	[31:0]	读/写	设置定时器 1 的比较缓冲器的值	0x00000000

32. 4. 10. TCNT0

1（定时器 1 计数观察寄存器）

寄存器	地址	读/写	描述	复位值
TCNT01	0x7F006020	读	定时器 1 计数观察寄存器	0x0000_0000

TCNT01	位	读/写	描述	初始状态
Timer 1 Count Observation	[31:0]	读	设置定时器 1 计数观察寄存器的值	0x00000000

32. 4. 11. TCNTB2（定时器 2 计数寄存器）

寄存器	地址	读/写	描述	复位值
TCNTB2	0x7F006024	读/写	定时器 2 计数缓冲器	0x0000_0000

TCNTB2	位	读/写	描述	初始状态
Timer 2 Count Buffer	[31:0]	读/写	设置定时器 2 的计数缓冲器的值	0x00000000

32. 4. 12. TCMPB2（定时器 2 比较寄存器）

寄存器	地址	读/写	描述	复位值
TCMPB2	0x7F006028	读/写	定时器 2 比较缓冲寄存器	0x0000_0000

TCMPB2	位	读/写	描述	初始状态
Timer 2 Compare Buffer	[31:0]	读/写	设置定时器 2 的比较缓冲器的值	0x00000000

32. 4. 13. TCNT02（定时器 2 计数观察寄存器）

寄存器	地址	读/写	描述	复位值
TCNT02	0x7F00602c	读	定时器 2 计数观察寄存器	0x0000_0000

TCNT02	位	读/写	描述	初始状态
Timer 2 Count Observation	[31:0]	读	设置定时器 2 计数观察寄存器的值	0x00000000

32. 4. 14. TCNTB3（定时器 3 计数寄存器）

寄存器	地址	读/写	描述	复位值
TCNTB3	0x7F006030	读/写	定时器 3 计数缓冲器	0x0000_0000

TCNTB3	位	读/写	描述	初始状态
Timer 3 Count Buffer	[31:0]	读/写	设置定时器 3 的计数缓冲器的值	0x00000000

32. 4. 15. TCMPB3（定时器 3 比较寄存器）

寄存器	地址	读/写	描述	复位值
TCMPB3	0x7F006034	读/写	定时器 3 比较缓冲寄存器	0x0000_0000

TCMPB3	位	读/写	描述	初始状态
Timer 3 Compare Buffer	[31:0]	读/写	设置定时器 3 的比较缓冲器的值	0x00000000

32. 4. 16. TCNT03（定时器 3 计数观察寄存器）

寄存器	地址	读/写	描述	复位值
TCNT03	0x7F006038	读	定时器 3 计数观察寄存器	0x0000_0000



TCNT03	位	读/写	描述	初始状态
Timer 3 Count Observation	[31:0]	读	设置定时器 3 计数观察寄存器的值	0x00000000

#### 32. 4. 17. TCNTB4（定时器 4 计数寄存器）

寄存器	地址	读/写	描述	复位值
TCNTB4	0x7F00603c	读/写	定时器 4 计数缓冲器	0x0000_0000

TCNTB4	位	读/写	描述	初始状态
Timer 4 Count Buffer	[31:0]	读/写	设置定时器 4 的计数缓冲器的值	0x00000000

#### 32. 4. 18. TCNT04（定时器 4 计数观察寄存器）

寄存器	地址	读/写	描述	复位值
TCNT04	0x7F006040	读	定时器 4 计数观察寄存器	0x0000_0000

TCNT04	位	读/写	描述	初始状态
Timer 4 Count Observation	[31:0]	读	设置定时器 4 计数观察寄存器的值	0x00000000

#### 32. 4. 19. TINT\_CSTAT（定时器中断控制和状态寄存器）

寄存器	位	读/写	描述	复位值
TINT_CSTAT	0x7F006044	读/写	定时器中断控制和状态寄存器	0x0000_0000

TINT_CSTAT	位	读/写	描述	初始状态
Reserved	[31:10]	读	保留位	0x00000
Timer 4 Interrupt Status	[9]	读/写	定时器 4 中断状态位。通过写 ‘1’ 清除该位	0x0

Timer 3 Interrupt Status	[8]	读/写	定时器 3 中断状态位。通过写 ‘1’ 清除该位	0x0
Timer 2 Interrupt Status	[7]	读/写	定时器 2 中断状态位。通过写 ‘1’ 清除该位	0x0
Timer 1 Interrupt Status	[6]	读/写	定时器 1 中断状态位。通过写 ‘1’ 清除该位	0x0
Timer 0 Interrupt Status	[5]	读/写	定时器 0 中断状态位。通过写 ‘1’ 清除该位	0x0
Timer 4 interrupt Enable	[4]	读/写	定时器 4 中断启动。 1: 启动 0: 禁止	0x0
Timer 3 interrupt Enable	[3]	读/写	定时器 3 中断启动。 1: 启动 0: 禁止	0x0
Timer 2 interrupt Enable	[2]	读/写	定时器 2 中断启动。 1: 启动 0: 禁止	0x0
Timer 1 interrupt Enable	[1]	读/写	定时器 1 中断启动。 1: 启动 0: 禁止	0x0
Timer 0 interrupt Enable	[0]	读/写	定时器 0 中断启动。 1: 启动 0: 禁止	0x0

## 33 RTC 实时时钟

本节介绍了实时时钟（RTC）在 S3C6410 RISC 微处理器上的功能及其使用。当系统电源关闭时，通过备用电源可以运行实时时钟（RTC）单元。数据包含的时间，即秒，分钟，小时，日期，日，月和年。RTC 单元操作一个外部 32.768kHz 的晶体，并可以执行报警功能。

### 33.1 RTC 实时时钟的特性

实时时钟包括以下功能：

- 二进制编码数据：秒，分钟，小时，日期，日，月和年。
- 闰年发生器。
- 报警功能：报警中断或从断电模式中唤醒。
- 时钟计数功能：时钟节拍中断或从断电模式中唤醒。
- 不存在千年虫问题。
- 独立地电源引脚（RTCVDD）。
- 支持毫秒标记的时间中断信号，用于 RTOS 内核时间标记。

### 33.2 实时时钟的操作说明

实时时钟的结构框图，如图 33-1 所示。

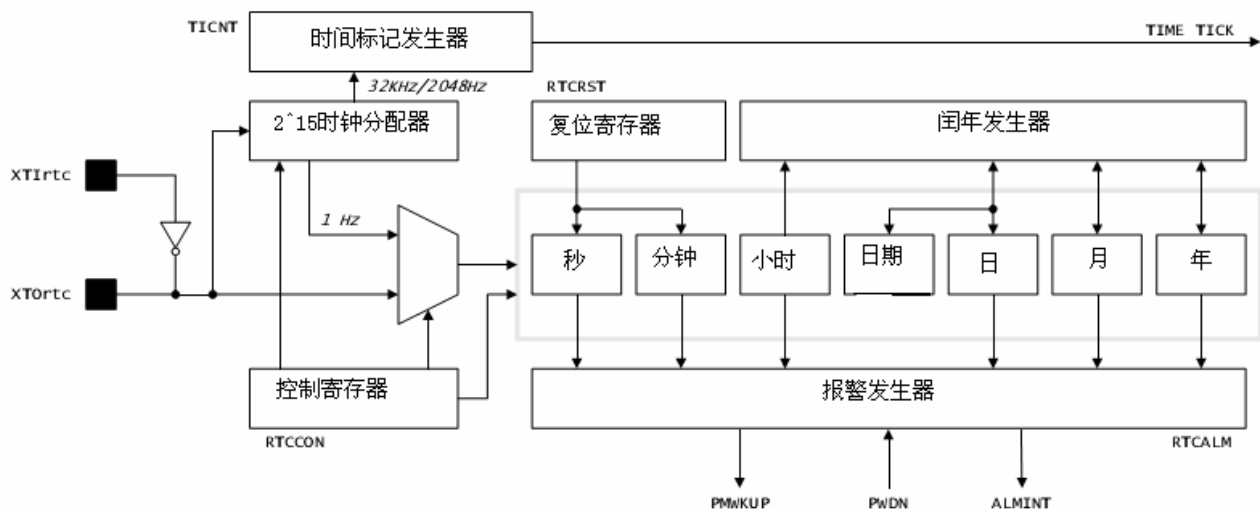


图 33-1 实时时钟的结构框图

### 1. 闰年发生器

闰年发生器通过 BCDDAY, BCDMON 和 BCDYEAR 的数据来决定每个月的最后一天是 28, 29, 30 还是 31。这个模块是通过决定最后的日期来判断闰年的。一个 8 位的计数器只能代表两个 BCD 数字，因此它不能决定 ‘00’ 年（年的最后两个数字为 ‘00’）是不是闰年。举例来说，它不能区分 1900 年和 2000 年。要解决这个问题，S3C6410 中的实时时钟模块，在 2000 年中，硬连接逻辑支持闰年。注意 1900 年不是闰年，而 2000 年是闰年。因此在 S3C6410 中的 ‘00’ 的两个数字表示 2000 而不是 1900。

### 2. 读/写寄存器

RTCCON 寄存器的位 0 必须被设置为高位，原因是可以正常写入实时时钟模块中的 BCD 寄存器，以显示秒, 分钟, 小时, 日期, 日, 月和年。CPU 必须分别在 RTC 模块的 BCDSEC, BCDMIN, BCDHOUR, BCDDATE, BCDDAY, BCDMON 和 BCDYEAR 寄存器中读取数据。但是，因为多个寄存器被读取，所以可能有一秒的偏差存在。例如，当用户从 BCDYEAR 到 BCDMIN 读取寄存器时，结果假设为 2059（年），12（月），31（日期），23（小时）和 59（分钟）。当用户读取 BCDSEC 寄存器及值范围从 1 到 59（秒）时，没有问题，但值为 0 秒，年，月，日，小时和分钟将被改变为 2060（年），1（月），1（日期），0（小时）和 0（分钟），就是因为这一秒的变差。在这种情况下，如果 BCDSEC 置 0，用户必须从 BCDYEAR 到 BCDSEC 重新读取。

### 3. 备份电源操作

通过备用电池可以驱动实时时钟逻辑，它是通过 RTCVDD 引脚进入实时时钟模块来提供电源的，即使

系统电源被关闭。当系统关闭时，CPU 的接口和实时时钟的逻辑必须被封锁，备用电池只有驱动振荡电路和 BCD 计数器，产生最小电源消耗。

4. 报警功能

实时时钟在断电模式或正常操作模式的某一特定时间内产生一个报警信号。在正常的操作模式下，报警中断（ALMINT）被激活。在断电模式下，电源管理唤醒（PMWKUP）信号与 ALMINT 一样充分被激活。实时时钟报警寄存器(RTCALM)，决定了报警启用/禁用的状态和报警时间设置的条件。

5. 标记时间中断

实时时钟标记时间被用于中断请求。TICNT 寄存器有一个中断使能位和一个中断计数值。当标记时间中断发生时，计数器的值达到 ‘0’ 。中断周期如下：

周期= (n+1) /32768 秒 (n=标记计数器的值)

注意：

RTC 时间标记可用于实时操作系统（RTOS）内核时间标记。如果时间标记是通过 RTC 时间标记产生的，RTOS 的时间相关功能将始终同步在实时时间中。

6. 32.768KHZ X-TAL 关系实例

如图 33-2 所示，显示了在 32.768kHz 下，实时时钟单位振动的电路。

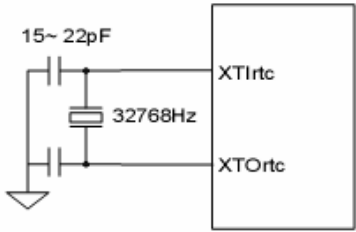


图 33-2 主振荡器电路的实例

7. 外部接口

如表 33-3 所示，显示了两种外部接口。

表 33-3 外部接口描述

名称	方向	描述
XTI	输入	32kHz RTC 振荡器的时钟输入

XTO	输入	32kHz RTC 振荡器的时钟输出
-----	----	--------------------

### 33.3 寄存器描述

下面主要介绍一下和实时时钟相关的一些寄存器，这里以列表的形式给出。  
存储器映射，如表 33-4 所示。

表 33-4 实时时钟寄存器摘要

寄存器	地址	读/写	描述	复位值
INTP	0x7E005030	读/写	中断等候寄存器	0x0
RTCCON	0x7E005040	读/写	实时时钟控制寄存器	0x0
TICNT	0x7E005044	读/写	标记时间计数寄存器	0x0
RTCALM	0x7E005050	读/写	实时时钟报警控制寄存器	0x0
ALMSEC	0x7E005054	读/写	报警秒数据寄存器	0x0
ALMMIN	0x7E005058	读/写	报警分钟数据寄存器	0x00
ALMHOUR	0x7E00505C	读/写	报警小时数据寄存器	0x0
ALMDATE	0x7E005060	读/写	报警天数据寄存器	0x01
ALMMON	0x7E005064	读/写	报警月数据寄存器	0x01
ALMYEAR	0x7E005068	读/写	报警年数据寄存器	0x0
BCDSEC	0x7E005070	读/写	BCD 秒寄存器	未定义
BCDMIN	0x7E005074	读/写	BCD 分钟寄存器	未定义
BCDHOURL	0x7E005078	读/写	BCD 小时寄存器	未定义
BCDDATE	0x7E00507C	读/写	BCD 日期寄存器	未定义
BCDDAY	0x7E005080	读/写	BCD 天寄存器	未定义
BCDMON	0x7E005084	读/写	BCD 月寄存器	未定义
BCDYEAR	0x7E005088	读/写	BCD 年寄存器。	未定义
CURTICNT	0x7E005090	读	当前标记时间计数寄存器。	0x0

下面主要针对个别寄存器进行描述。

33.3.1. 实时时钟的控制寄存器（RTCCON）

RTCCON 寄存器和 RTCEN 寄存器一样，都是由 9 位组成。它控制 BCD 设置的读/写启动，即 CNTSEL 和 TICEN 测试。

RTCEN 位能够控制 CPU 和 RTC 之间的所有接口，因此在系统复位后，它必须在 RTC 控制中设置为 ‘1’ 来启动数据读取/写入。切断电源前，RTCEN 位必须清除为 ‘0’，以预防无意写入 RTC 寄存器中。

寄存器	地址	读/写	描述	复位值
RTCCON	0x7E005040	读/写	实时时钟控制寄存器	0x0

RTCCON	位	描述	初始状态
TICEN	[8]	标记定时器启动。 0 = 禁止    1 = 启动	0
Reserved	[7:3]	保留。	
CNTSEL	[2]	BCD 计数选择。 0 = 合并 BCD 计数器 1 = 保留（分开 BCD 计数器）	0
CLKSEL	[1]	BCD 时钟选择。 0 = XTAL 1/2 <sup>15</sup> 分频时钟 1 = 保留（XTAL 时钟单独测试）	0
RTCEN	[0]	RTC 控制启动。 0 = 禁止    1 = 启动 注：当 RTCEN 启动，BCD 时间计数设置，RTC 时钟计数器复位和读取操作可以被执行。	0

33.3.2. 标记时间计数寄存器（TICNT）

寄存器	地址	读/写	描述	复位值
-----	----	-----	----	-----

TICNT	0x7E005044	读/写	标记时间计数寄存器	0x0
-------	------------	-----	-----------	-----

TICNT	位	描述	初始状态
TICK TIME COUNT	[15: 0]	16 位标记定时器计数值	0

### 33.3.3. 实时时钟报警控制寄存器 (RTCALM)

RTCALM 寄存器决定报警启动和报警时间。注意：RTCALM 寄存器产生报警信号通过 ALMINT 和 PMWKUP 断电模式，而且，在正常的运作模式下只能通过 ALMINT。

寄存器	地址	读/写	描述	复位值
RTCALM	0x7E005050	读/写	实时时钟报警控制寄存器	0x0

RTCALM	位	描述	初始状态
Reserved	[7]		0
ALMEN	[6]	通用报警启动： 0 = 禁止    1 = 启动	0
YEAREN	[5]	年报警启动： 0 = 禁止    1 = 启动	0
MONEN	[4]	月报警启动： 0 = 禁止    1 = 启动	0
DATEEN	[3]	天报警启动： 0 = 禁止    1 = 启动	0
HOUREN	[2]	小时报警启动： 0 = 禁止    1 = 启动	0
MINEN	[1]	分钟报警启动： 0 = 禁止    1 = 启动	0
SECEN	[0]	秒报警启动	0



		0 = 禁止    1 = 启动	
--	--	------------------	--

#### 33.3.4. 报警秒数据寄存器（ALMSEC）

寄存器	地址	读/写	描述	复位值
ALMSEC	0x7E005054	读/写	报警秒数据寄存器	0x0

ALMSEC	位	描述	初始状态
Reserved	[7]		0
SECDATA	[6:4]	秒报警的 BCD 值。0~5	000
	[3:0]	0~9	0000

#### 33.3.5. 报警分钟数据寄存器（ALMMIN）

寄存器	地址	读/写	描述	复位值
ALMMIN	0x7E005058	读/写	报警分钟数据寄存器	0x00

ALMMIN	位	描述	初始状态
Reserved	[7]		0
MINDATA	[6:4]	分钟报警的 BCD 值。0~5	000
	[3:0]	0~9	0000

#### 33.3.6. 报警小时数据寄存器（ALMHOUR）

寄存器	地址	读/写	描述	复位值
ALMHOUR	0x7E00505C	读/写	报警小时数据寄存器。	0x00

ALMHOUR	位	描述	初始状态
Reserved	[7: 6]		00
HOURDATA	[5:4]	时报警的 BCD 值。 0~2	00
	[3:0]	0~9	0000

33.3.7. 报警天数据寄存器（ALMDATE）

寄存器	地址	读/写	描述	复位值
ALMDATE	0x7E005060	读/写	报警天数据寄存器	0x01

ALMDATE	位	描述	初始状态
Reserved	[7: 6]		00
DATEDATA	[5:4]	天报警的 BCD 值，从 0 到 28，29，30，31。0~3	00
	[3:0]	0~9	0001

33.3.8. 报警月数据寄存器（ALMMON）

寄存器	地址	读/写	描述	复位值
ALMMON	0x7E005064	读/写	报警月数据寄存器。	0x01

ALMMON	位	描述	初始状态
Reserved	[7: 5]		00
MONDATA	4	月报警的 BCD 值。0~1	0
	[3:0]	0~9	0001

33.3.9. 报警年数据寄存器 (ALMYEAR)

寄存器	地址	读/写	描述	复位值
ALMYEAR	0x7E005068	读/写	报警年数据寄存器	0x0

ALMYEAR'	位	描述	初始状态
YEARDATA	[7:0]	年报警的 BCD 值. 00~99	0x0

33.3.10. BCD 秒寄存器 (BCDSEC)

寄存器	地址	读/写	描述	复位值
BCDSEC	0x7E005070	读/写	BCD 秒寄存器	未定义

BCDSEC'	位	描述	初始状态
SECDATA	[6:4]	秒的 BCD 值. 0~5	-
	[3:0]	0~9	-

33.3.11. BCD 分钟寄存器 (BCDMIN)

寄存器	地址	读/写	描述	复位值
BCDMIN	0x7E005074	读/写	BCD 分钟寄存器	未定义

BCDMIN'	位	描述	初始状态
MINDATA	[6:4]	分的 BCD 值. 0~5	-
	[3:0]	0~9	-

33.3.12. BCD 小时寄存器 (BCD HOUR)

寄存器	地址	读/写	描述	复位值
BCD HOUR	0x7E005078	读/写	BCD 小时寄存器	未定义

BCD HOUR	位	描述	初始状态
Reserved	[7:6]		
HOURDATA	[5:4]	时的 BCD 值。0~2	-
	[3:0]	0~9	-

33.3.13. BCD 日期寄存器 (BCD DATE)

寄存器	地址	读/写	描述	复位值
BCD DATE	0x7E00507C	读/写	BCD 日期寄存器。	未定义

BCDDAY	位	描述	初始状态
Reserved	[7:6]		
DATEDATA	[5:4]	日期的 BCD 值。0~3	-
	[3:0]	0~9	-

33.3.14. BCD 天寄存器 (BCD DAY)

寄存器	地址	读/写	描述	复位值
BCDDAY	0x7E005080	读/写	BCD 天寄存器	未定义

BCDDAY	位	描述	初始状态
Reserved	[7:3]		
DAYDATA	[2:0]	天的 BCD 值。	

		1~7	
--	--	-----	--

### 33.3.15. BCD 月寄存器 (BCDMON)

寄存器	地址	读/写	描述	复位值
BCDMON	0x7E005084	读/写	BCD 月寄存器	未定义

BCDMON	位	描述	初始状态
Reserved	[7:5]		
MONDATA	[4]	月的 BCD 值。0~1	-
	[3:0]	0~9	-

### 33.3.16. BCD 年寄存器 (BCDYEAR)

寄存器	地址	读/写	描述	复位值
BCDYEAR	0x7E005088	读/写	BCD 年寄存器	未定义

BCDYEAR'	位	描述	初始状态
YEARDATA	[7:0]	年的 BCD 值。 00~99	-

### 33.3.17. 当前标记时间计数寄存器

寄存器	地址	读/写	描述	复位值
CURTICNT	0x7E005090	读	当前标记时间计数寄存器	0x0

CURTICNT	位	描述	初始状态
----------	---	----	------

Tick counter observation	[15:0]	当前标记计数值	-
--------------------------	--------	---------	---

33. 3. 18. 中断等待寄存器

寄存器	地址	读/写	描述	复位值
INTP	0x7E005030	读/写	BCD 年寄存器	未定义

INTP	位	描述	初始状态
Reserved	[7:2]	保留	00
ALARM	[1]	报警中断等待位： 0：没有中断发生 1：中断发生	0
Time	[0]	Time TIC 中断等待位： 0：没有中断发生 1：中断发生	0

33. 4 RTC 寄存器编程举例

通过以上对 RTC 实时时钟的介绍，相信读者已经掌握了 S3C6410 中 RTC 的基本特性及寄存器的相关功能。这节我们主要结合 ARM11 来举例说明 RTC 实时时钟的具体实现，有助于更好地理解该模块的性能。

RTC 报警功能的实现：

输入：NONE(通过用户在 rtc.h 中预先定义报警值和警报组件的选择)。

输出：NONE（当 RTC 达到定义报警的值时，通过选择组件，如秒，分钟等，警报中断将发生）。

以下是 RTC 警报代码的具体实现：

```
void RTC_Alarm(void)           //RTC 实时时钟警报函数
{
    u32 uSelect;
    RTC_Init();
```

```
printf("[ RTC Alarm Test for S3C6410 ]\n"); //输出[ RTC Alarm Test for S3C6410 ]信息。
```

```
RTC_SetCON(0, 0, 0, 0, 0, 1); //没有复位, 合并 BCD 计数器, 1/32768, RTC 控制使能。
```

```
RTC_SetAlmTime(AlmYear, AlmMon, AlmDate, AlmHour, AlmMin, AlmSec);
```

```
printf("Select alarm interrupt source \n");
```

```
printf("1:sec 2:min 3:hour 4:date 5:month 6:year 7:All components\n");
```

```
uSelect = GetIntNum();
```

```
switch(uSelect)
```

```
{
```

```
case 1:
```

```
    RTC_SetAlmEn(1, 0, 0, 0, 0, 0, 1);
```

```
    break;
```

```
case 2:
```

```
    RTC_SetAlmEn(1, 0, 0, 0, 0, 1, 0);
```

```
    break;
```

```
case 3:
```

```
    RTC_SetAlmEn(1, 0, 0, 0, 1, 0, 0);
```

```
    break;
```

```
case 4:
```

```
    RTC_SetAlmEn(1, 0, 0, 1, 0, 0, 0);
```

```
    break;
```

```
case 5:
```

```
    RTC_SetAlmEn(1, 0, 1, 0, 0, 0, 0);
```

```
    break;
```

```
case 6:
```

```
    RTC_SetAlmEn(1, 1, 0, 0, 0, 0, 0);
```

```
    break;
```

```
case 7:
```

```

        RTC_SetAlmEn(1, 1, 1, 1, 1, 1, 1);
        break;
    default :
        RTC_SetAlmEn(1, 1, 1, 1, 1, 1, 1);
        break;
}

printf("After 5 sec, alarm interrupt will occur.. \n");
RTC_PrintAlm();
RTC_Print();
uCntAlarm = 0;

INTC_SetVectAddr(NUM_RTC_ALARM, Isr_RTC_Alm);
INTC_Enable(NUM_RTC_ALARM);

RTC_SetCON(0, 0, 0, 0, 0, 0); //没有复位，合并 BCD 计数器，1/32768，RTC 控制禁用。

while(uCntAlarm==0)
{

};

RTC_Print();
INTC_Disable(NUM_RTC_ALARM);
RTC_SetCON(0, 0, 0, 0, 0, 0); //RTC 控制禁用(用于电源消耗)，1/32768，正常的(合并)，没有复位}
}

```



## 34 看门狗定时器

当控制器操作被噪音或系统错误等故障打断时，S3C6410 RISC 微处理器的看门狗定时器恢复控制器的操作。它用于正常的 16 位间隔定时器来要求中断服务。看门狗定时器产生中断信号。用 WDT 代替 PWM 定时器的优点是 WDT 产生复位信号。

### 34.1 看门狗定时器的特性

看门狗定时器包含下面的特性：

- 具有中断请求的正常间隔定时器模式。
- 当定时器计数值达到0（超时），内部复位信号有效。
- 电平触发器中断机制。

### 34.2 功能描述

如图 34-1 所示，显示看门狗定时器的功能模块图。看门狗定时器用 PCLK 作为它的源时钟。PCLK 频率被预分频来产生相应的看门狗定时器时钟，并将所得的频率再次被分频。

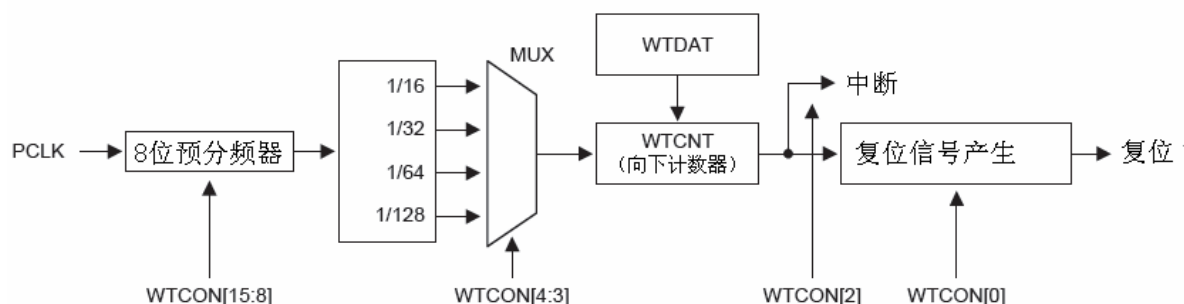


图 34-1 看门狗定时器模块图

在看门狗定时器控制（WTCN）寄存器，预分频器值和分频因子被指定。有效的预分频器值的范围是 0～

28-1。分频因子能选择 16、32、64 或 128。

用下面的等式来计算看门狗定时器时钟频率，每一个定时器的时钟周期：

看门狗定时器时钟频率 =  $1 / (PCLK / (\text{预分频值} + 1) / \text{分频因子})$

一旦看门狗定时器有效，看门狗定时器数据（WTDAT）寄存器的值将不能被自动重新载入定时器计数器（WTCNT）。在看门狗定时器开始前，一个初始值必须写入看门狗定时器计数（WTDAT）寄存器。

在 ARM11 处理器中，看门狗操作的具体代码如下：

```
// WDT_operate 函数：主要功能通过输入操作看门狗定时器。
// 输入：uEnReset, uEnInt, uEnWDT [0:禁用    1:使能]
//       uSelectCLK (时钟分频因子) [0:16    1:32    2:64    3:128]
//       uPrescaler [1~256]
//       uWTDAT [0~2^15]
//       uWTCNT [0~2^15]
// 输出：NONE

void WDT_operate(u32 uEnReset, u32 uEnInt, u32 uSelectCLK, u32 uEnWDT, u32 uPrescaler, u32
uWTDAT, u32 uWTCNT)
{
    float fWDTclk;
    Outp32(rWTCON, 0);
    Outp32(rWTDAT, 0);
    Outp32(rWTCNT, 0);
    Outp32(rWTDAT, uWTDAT);
    Outp32(rWTCNT, uWTCNT);
    Outp32(rWTCON, (uEnReset<<0) | (uEnInt<<2) | (uSelectCLK<<3) | (uEnWDT<<5) | ((uPrescaler)<<8));
    fWDTclk = (1/(float)((float)g_PCLK/((float)uPrescaler+1)/
(1<<(uSelectCLK+4))))*uWTDAT;
    printf("WDT_clk = %f sec\n", fWDTclk);
}
```

### 34.3 考虑调试环境

当S3C6410在调试模式（使用嵌入的ICE）时，看门狗定时器不能进行操作。

在调试模式下，看门狗定时器能决定来自 CPU 内核的信号（DBACK 信号）是否是当前的信号。一旦DBGACK 信号无效，看门狗定时器的复位输出禁止，并终止看门狗定时器终止。

### 34.4 特殊功能的寄存器

存储器映射如表34-1所示。

表 34-1 存储器映射

寄存器	地址	读/写	描述	复位值
WTCN	0x7E004000	读/写	看门狗定时器控制寄存器。	0x8021
WTDAT	0x7E004004	读/写	看门狗定时器数据寄存器。	0x8000
WTCNT	0x7E004008	读/写	看门狗定时器计数寄存器。	0x8000
WTCLRINT	0x7E00400c	写	看门狗定时器中断清除寄存器。	—

#### 34.4.1. 看门狗定时器控制（WTCN）寄存器

WTCN 寄存器允许用户启动/禁止看门狗定时器，从四个不同的时钟源选择时钟信号，启动/禁止看门狗定时器输出。

看门狗定时器用于恢复 S3C6410 重启故障。如果不希望控制器重启，则看门狗定时器必须禁止。如果用户想使用看门狗定时器提供的正常定时器，需要启动中断，并且禁止看门狗定时器。如表 34-2 所示。

表 34-2 WTCN 寄存器

寄存器	位	读/写	描述	复位值
WTCN	0x7E004000	读/写	看门狗定时器控制寄存器。	0x8021

WTCN	位	描述	初始状态
Prescaler value	[15:8]	预分频值。	0x80

		该值范围是 0~ (28-1)。	
Reserved	[7:6]	保留。 正常操作这两位必须是 00。	00
Watchdog timer	[5]	启动或禁止看门狗定时器的位。 0=禁止 1=有效	1
Clock select	[4:3]	决定时钟分频因子。 00: 16          01 : 32 10: 64          11 : 128	00
Interrupt generation	[2]	中断的启动或禁止位。 0=禁止 1=启动	0
Reserved	[1]	保留。 正常操作时，该位必须设置为 0。	0
Reset enable/disable	[0]	对于复位信号，启动或禁止看门狗定时器的输出位。 1:看门狗定时器超时，发出复位信号 0:禁止看门狗定时器的复位功能	1

#### 34.4.2. 看门狗定时器数据（WTDAT）寄存器

WTDAT寄存器用于指定超时时间。看门狗定时器初始化时，WTDAT的内容不能被自动载入到定时器计数器。然而，使用0x8000（初始值）可以驱使第一个超时。这种情况下，WTDAT的值将自动重载入WTCNT。如表34-3所示。

表 34-3WTDAT 寄存器

寄存器	地址	读/写	描述	复位值
WTDAT	0x7E004004	读/写	看门狗定时器数据寄存器。	0x8000

WTDAT	位	描述	初始状态
Count reload value	[15:0]	看门狗定时器重载的计数值。	0x8000

34.4.3. 看门狗定时器计数（WTCNT）寄存器

正常操作情况下，WTCNT 寄存器包含看门狗定时器的当前计数值。

注：当看门狗定时器初始化时，WTDAT 寄存器的内容不能被自动载入到定时器计数寄存器中，因此在激活它前，WTCNT 寄存器必须设置一个初始值。如表 34-4 所示。

表 34-4 WTCNT 寄存器

寄存器	地址	读/写	描述	复位值
WTCNT	0x7E004008	读/写	看门狗定时器计数寄存器。	0x8000

WTCNT	位	描述	初始状态
Count value	[15:0]	看门狗定时器的当前计数值。	0x8000

34.4.4. 看门狗定时器中断清除（WTCLRINT）寄存器

WTCLRINT 寄存器用于清除中断。中断服务完成后，中断服务程序清除相关中断。可以在该寄存器写入任意值清除中断。不允许读该寄存器。如表 34-5 所示。

表 34-5 WTCLRINT 寄存器

寄存器	地址	读/写	描述	复位值
WTCLRINT	0x7E00400c	写	看门狗定时器中断清除寄存器。	–

WTCLRINT	位	描述	初始状态
中断清除	[0]	写任意值来清除中断。	–

# 35 AC97 控制器

本节主要介绍 AC97 控制器在 S3C6410X RISC 微处理器上的功能及其使用。

## 35.1 AC97 控制器概述

S3C6410X 的 AC97 控制器单位支持 AC97 2.0 版的功能。带有 AC97 解码器的 AC97 控制器用于连接音频控制器(AC-link)，控制器发送立体声 PCM 数据给多媒体数字信号编解码器。在多媒体数字信号编解码器里，外部数字式模拟转换器（DAC）由音频采样转换为一个模拟音频波形。控制器多媒体数字信号编解码器里接收立体声 PCM 数据和单声道麦克风的数据，然后存入存储器中。这一节主要介绍了 AC97 控制器单元的设计模型。

### 1. AC97 控制器

AC97 控制器包括以下特性：

- （1）独立通道，用于立体声 PCM 输入，立体声 PCM 输出和单声道 MIC 输入。
- （2）基于 DMA 操作和基于中断操作。
- （3）变量取样速率 AC97 多媒体数字信号编解码器接口（48KHz 和低于 48KHz）。
- （4）16 位，16 个 FIFO 每通道。
- （5）只支持基本多媒体数字信号编解码器。

### 2. 信号

显示信号，如表 35-1 所示。

表 35-1 显示信号

名称	方向	说明
AC_nRESE	输出	低级多媒体数字信号编解码器复位
AC_BIT_CL	输入	12.288MHz 位速率时钟
AC_SYNC	输出	48KHz 指示器结构和同步装置
AC_SDO	输出	连续音频输出数据
AC_SDI	输入	连续音频输入数据

## 35.2 AC97 工作流程

本节主要说明 AC97 控制器的操作，如 AC-Link，省电序列和唤醒序列。

### 1. 结构框图

这是一个点对点的同步串行连接，支持全双工信号数据传输。所有的数字音频流和命令/状态信息是通过 AC-link 相通起来的。如图 35-1 所示，显示了 S3C6410 AC97 控制器的功能结构框图。

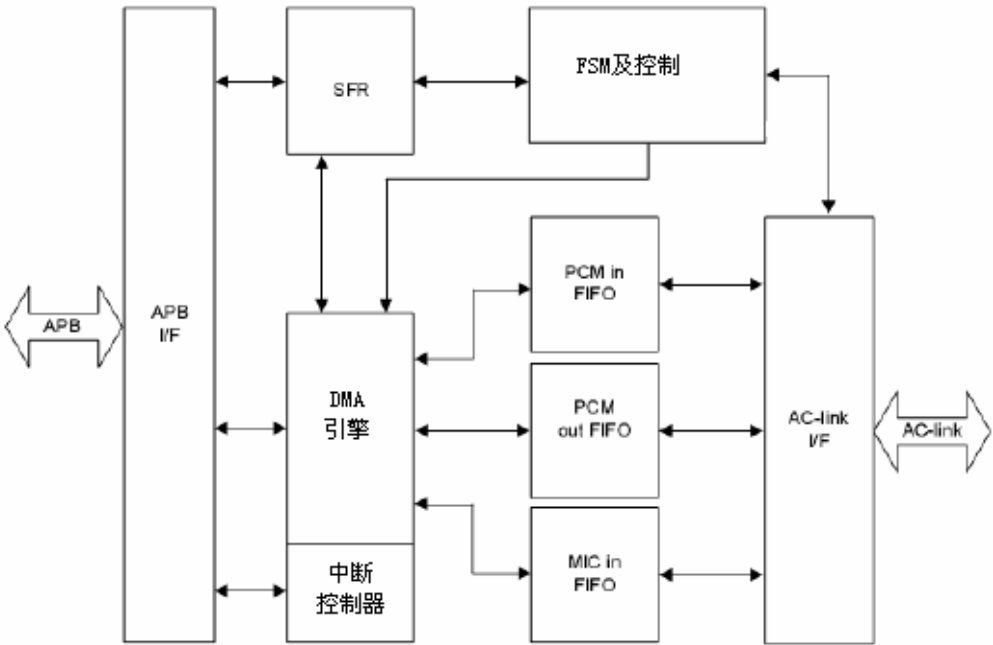


图 35-1 S3C6410 AC97 控制器的功能结构框图

### 2. 内部数据通道

它有立体声编码调制（PCM）输入，立体声 PCM 输出和单声道 Mic-in 缓冲器，其中包括 16 位和 16 个缓冲器。它也有 20 位 I/O 移位寄存器通过 AC-link。如图 35-2 所示，显示了 S3C6410 AC97 控制器的内部数据通道。

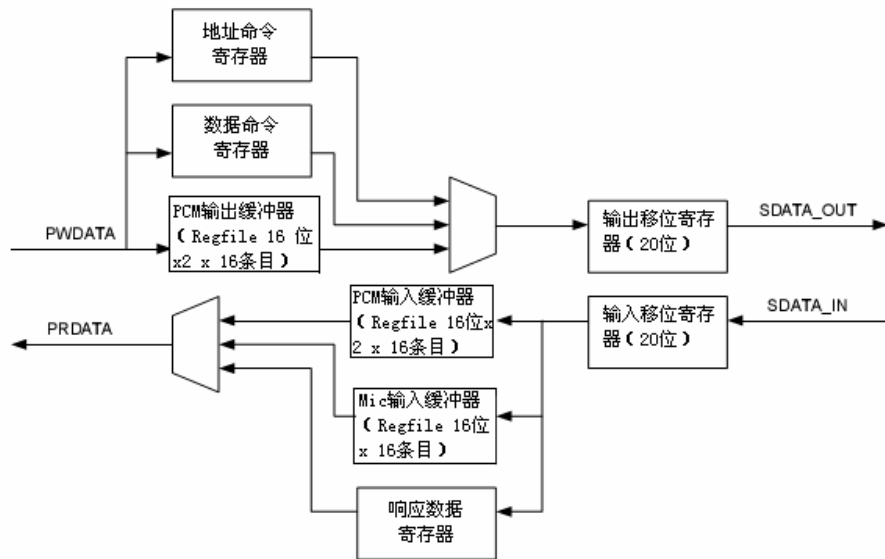


图 35-2 内部数据通道

### 3. 操作流程图

AC97 控制器在 ARM11 中，支持 AC97 2.0 版的功能。它带有 AC97 解码器，主要用于连接音频控制器 (AC-link)。

当初始化 AC97 控制器时，首先必须进行系统复位或冷复位，因为我们不知道以前的外部 AC97 音频-编解码器的状态。

ARM11 中初始化 AC97 控制器的代码如下：

```
//AC97 控制器初始化
```

```
bool AC97_Init(void)
```

```
{
```

```
    u32 i=0;
```

```
    Disp("\nAC97 Initialization...\n");
```

```
//编解码器准备就绪，检查是否使用了编解码器准备中断
```

```
    //AC97 冷复位
```

```
    AC97_ColdReset();
```

```
    uCodecReadyIrq =0;
```

```
    INTC_SetVectAddr(NUM_AC97, Isr_AC97_CodecReady);
```



```

INTC_Enable(NUM_AC97);
AC97_EnableInt(CODEC_READY_INT); //AC97 启动
while(!uCodecReadyIrq)
{
    Disp(".");
    Delay(3000);
    i++;
    if(i==20)
        break;
}
Disp("\n");
if(i>=20)
{
    Disp("\nAC97 codec is not ready.");
    Disp("\nCheck on connection between AP and AC97 CODEC.\n");
    Disp("\nBye. ");
    return false;
}
else
{
    return true;
}
}

```

AC97 控制器冷复位的代码如下:

//AC97 冷复位的一段程序

```

void AC97_ColdReset(void)
{
    u32 uGlobalCon;
    Disp("\n=>Cold Reset\n");
}

```

```

//uGlobalCon = Inp32(AC97_BASE+rACGLBCTRL);
uGlobalCon = (1<<0);
AC97Outp32(rACGLBCTRL, uGlobalCon);
Delay(1000);
uGlobalCon &= ~(1<<0);
AC97Outp32(rACGLBCTRL, uGlobalCon);
Delay(1000);
uGlobalCon = (1<<0);
AC97Outp32(rACGLBCTRL, uGlobalCon);
Delay(1000);
uGlobalCon &= ~(1<<0);
AC97Outp32(rACGLBCTRL, uGlobalCon);
Delay(1000);
AC97_ControllerState();
#if (AC97_CODEC_NAME== WM9713)
    AC97_WarmReset();
#endif
uGlobalCon |= (1<<2);
AC97Outp32(rACGLBCTRL, uGlobalCon);
AC97_ControllerState();
uGlobalCon |= (1<<3);
AC97Outp32(rACGLBCTRL, uGlobalCon);
AC97_ControllerState();
//Disp("AC97 Codec Powerdown Ctrl/Stat Reg. Value (at 0x26): 0x%x\n",
AC97_CodecCmd(READ, 0x26, 0x0000));
}

```

确保 GPIO 已经准备就绪。然后允许多媒体数字信号编解码器有准备中断信号。通过轮询和中断检查多媒体数字信号编解码器准备中断信号。当发现中断信号，必须声明多媒体数字信号编解码器准备中断信号。利用 DMA 或 PIO 能够立即从内存到寄存器或从寄存器到内存传输数据。如果内部 FIFO (TX FIFO 或 RX FIFO)

不为空，则传输数据。此外，还可以在 AC-link 上转换以前的 FIFO。如图 35-3 所示，显示 AC97 操作流程

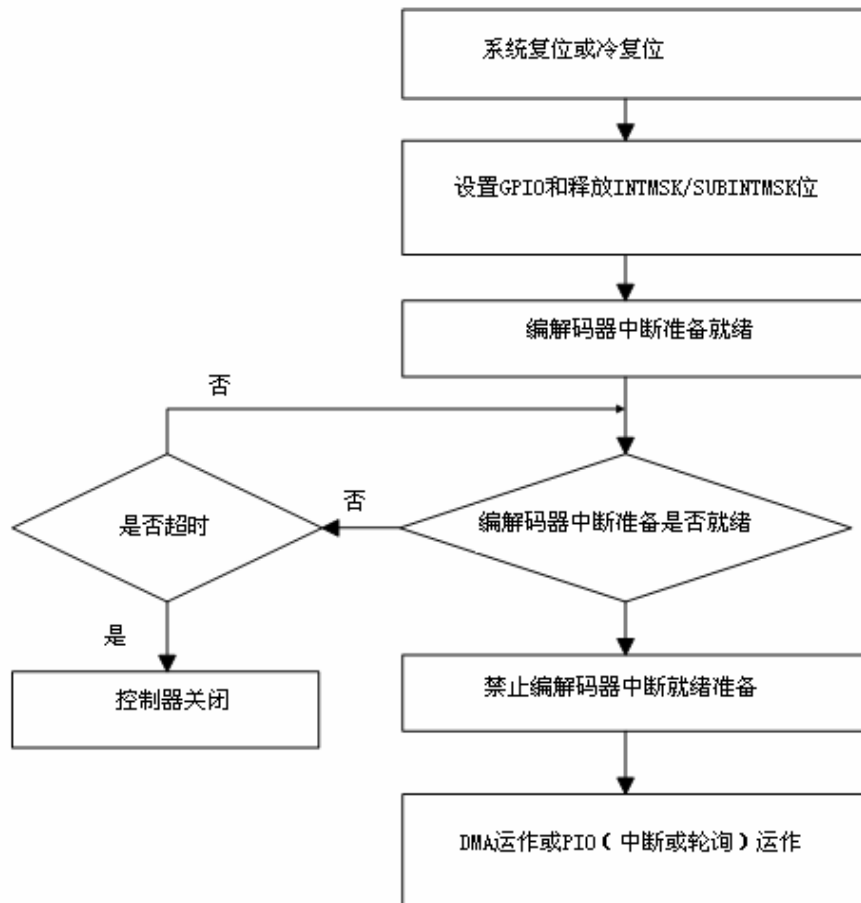


图 35-3 AC97 操作流程

下面是 ARM11 中 AC97 控制器状态控制代码部分，AC97 控制器状态分为启动状态和禁止状态两种，具体的代码实现如下：

```

//AC97 控制器状态定义
void AC97_ControllerState(void)
{
    u32 uState;
    uState= Inp32(AC97_BASE+rACGLBSTAT);
    if((uState & 0x7) == 0)

```

```

        Disp("AC97 Controller State: Idle\n");
    else if ((uState & 0x7) == 1)
        Disp("AC97 Controller State: Init\n");
    else if ((uState & 0x7) == 2)
        Disp("AC97 Controller State: Ready\n");
    else if ((uState & 0x7) == 3)
        Disp("AC97 Controller State: Active\n");
    else if ((uState & 0x7) == 4)
        Disp("AC97 Controller State: LP\n");
    else if ((uState & 0x7) == 5)
        Disp("AC97 Controller State: Warm\n");
}

```

//AC97 控制器状态-启动状态

```

void AC97_EnableInt(AC97_INT eInt)
{
    u32 uInt;
    uInt= Inp32(AC97_BASE+rACGLBCTRL);
    if(eInt == CODEC_READY_INT)
        uInt |= 1<<22;
    else if (eInt == PCMOUT_UNDERRUN_INT)
        uInt |= 1<<21;
    else if (eInt == PCMIN_OVERRUN_INT)
        uInt |= 1<<20;
    else if (eInt == MICIN_OVERRUN_INT)
        uInt |= 1<<19;
    else if (eInt == PCMOUT_THRESHOLD_INT)
        uInt |= 1<<18;
    else if (eInt == PCMIN_THRESHOLD_INT)
        uInt |= 1<<17;
}

```

```

    else if (eInt == MICIN_THRESHOLD_INT)
        uInt |= 1<<16;
    //Disp("rACGLBCTRL: 0x%x\n", uInt);
    AC97Outp32(rACGLBCTRL, uInt);
}
//AC97 控制器状态-禁止状态
void AC97_DisableInt(AC97_INT eInt)
{
    u32 uInt;

    uInt= Inp32(AC97_BASE+rACGLBCTRL);
    if(eInt == CODEC_READY_INT)
        AC97Outp32(rACGLBCTRL, uInt & ~(1<<22));
    else if (eInt == PCMOUT_UNDERRUN_INT)
        AC97Outp32(rACGLBCTRL, uInt & ~(1<<21));
    else if (eInt == PCMIN_OVERRUN_INT)
        AC97Outp32(rACGLBCTRL, uInt & ~(1<<20));
    else if (eInt == MICIN_OVERRUN_INT)
        AC97Outp32(rACGLBCTRL, uInt & ~(1<<19));
    else if (eInt == PCMOUT_THRESHOLD_INT)
        AC97Outp32(rACGLBCTRL, uInt & ~(1<<18));
    else if (eInt == PCMIN_THRESHOLD_INT)
        AC97Outp32(rACGLBCTRL, uInt & ~(1<<17));
    else if (eInt == MICIN_THRESHOLD_INT)
        AC97Outp32(rACGLBCTRL, uInt & ~(1<<16));
}

```

### 35.3 AC-LINK 数字接口协议

每个 AC97 多媒体数字信号编解码器合并了一个 5 引脚数字串行接口，它连接到 S3C6410 AC97 控制器。

AC-link 是一个全双工，固定时钟和 PCM 数字流。它采用一个时间分割多元来配置处理控制寄存器的通道和多重输入输出音频流。AC-link 结构划分每个音频结构为 12 个输出和 12 引入数据流。每个流有 29 位样本分辨率，需要一个 DAC 和一个有最低 16 位分辨率的模数转换器（ADC）。

支持 S3C6410 AC97 控制器的插槽定义。该 S3C6410 AC97 控制器在 AC-link 上提供同步的所有数据处理。如图所示，如图 35-4 所示，显示带有插槽转让的双向 AC-link。

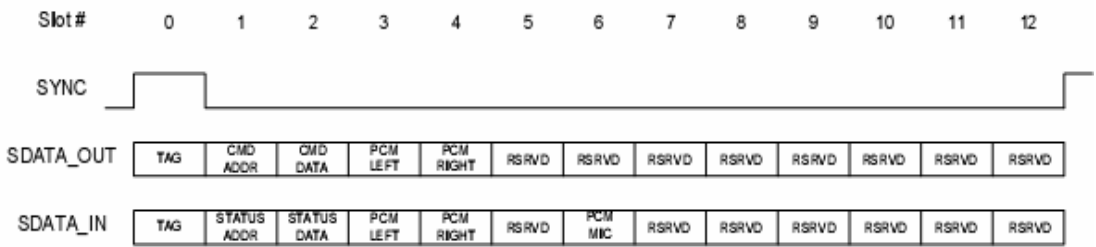


图 35-4 带有插槽转让的双向 AC-link

数据处理的 256 个信息位和 13 时段组成，即称为帧。时段 0 被称为标记相位，它为 16 位长。其它 12 个时段被称为数据相位。标记相位包含一个位，用来确定有效帧，和在数据相位确认时段的 12 位，包含有效数据。在数据相位每个时段长为 20 位。当 SYNC（同步）变高时，一个帧开始。同步是高位的时间段的数量，符合标记相位。AC97 帧发生在固定的 48kHz 间隔，并同步到 12.288MHz 位时钟，BITCLK。当发送传输数据和接受数据时，控制器和媒体数字信号编解码器用于 SYNC 和 BITCLK 的测定。在每个 BITCLK 的上升边缘，发送器转换串行数据流，在 BITCLK 的下降边缘，接收器取样串行数据流。

在其串行数据流中，发送器必须标记有效时段。有效插槽标记在插槽 0 中。在 AC-link 上的串行数据被排序从最高有效位（MSB）至最低有效位（LSB）。标记相位的首位是位 15，在数据状态里的各个插槽首位是位 19。任何插槽的最后位是位 0。

1. AC-LINK 的输出帧（SDATA\_OUT）

（1）插槽 0：标记相位

在插槽 0 中，第一位是一个位（SDAT\_OUT，位 15），其代表全部帧的有效。如果位 15 是 1，当前帧至少包含一个有效时段。下一个 12 位的位置符合每 12 时段，包含有效数据。为了编解码器寄存器的 I/O 读和写的描述，在下一项中插槽 0 的位 0 和位 1 被用于编解码器 I/O 位。在这种方式中，不同采样速率的数据流可以越过 AC-link 在其固定的 48kHz 音频帧速率上被传输。

（2）插槽 1：指令地址端口

在插槽 1 中，它传达控制寄存器地址和写/读指令信息到 AC97 控制器。

当软件访问主要编解码器时，硬件配置帧如下：

- 在插槽 0 中，设置用于插槽 1, 2 的有效位。
- 在插槽 1 中，位 19 被设置（读）或清除（写）。配置位 18-12（插槽 1 的）来指定编解码寄存器。其它的均为 0（保留）。
- 在插槽 2 中，由于输出 FRAME，用写入的数据来配置它。

（3）插槽 2：指令数据端口

在插槽 2 中，写入以 16 位分辨率的数据。（[19:4]是有效数据）

（4）插槽 3：PCM 重放左声道

插槽 3 是音频输出帧，是综合数字音频左流。如果采样的分辨率少于 16 位，在插槽为 0 中，AC97 控制器供应所有培训的非有效位的位置。

（5）插槽 4：PCM 重放右声道

插槽 3 是音频输出帧，是综合数字音频右流。如果采样的分辨率少于 16 位，在插槽为 0 中，AC97 控制器供应所有训练的非有效位的位置。如图 35-5（a）所示，显示 AC97 控制器输出结构。

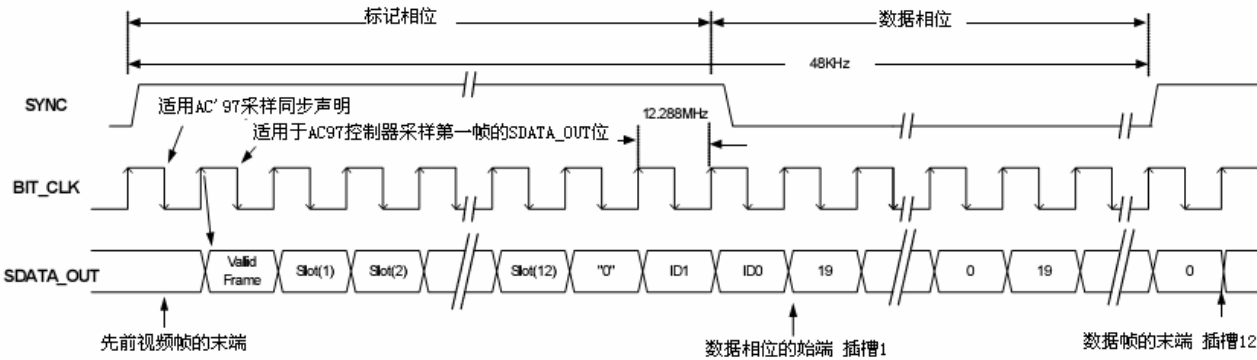


图 35-5（a） AC97 控制器输出结构

## 2. AC-LINK 的输入帧 (SDATA\_IN)

（1）插槽 0：标记相位

在插槽 0 中，第一位（SDATA\_OUT, 位 15）表示在多媒体数字信号编解码器中 AC97 控制器是否准备就绪状态。如果多媒体数字信号编解码器准备位置为 0，这意味着该 AC97 控制器没有准备好正常工作。有效复位后的电源和 AC97 控制器的电压是稳定的，这种情况是正常的。

## （2）插槽 1：状态地址 Port/SLOTREQ 位（端口/插槽）

状态端口监视用于 AC97 控制器功能的状态。它不是限制混音器设置和状态管理。如果控制器标记符插槽 1 和 2 有效期间为插槽 0，音频输入帧插 1 秒流回音控制寄存器指数是为数据返回在插槽 2。如果伴随状态的地址匹配与最后有效的地址发出的命令和最近读取命令之间时，该控制器只接受状态数据。对于多种取样速率输出，编解码器检查它的取样速率控制寄存器、FIFO 的状态，并且在每一个音频输出帧引入 SDATA\_OUT 标志位来确定哪个 SLOTREQ 位设置为有效。在当前音频输入帧指定输出通道请求数据期间，SLOTREQ 位有效。对于固定的 48kHz 的操作，SLOTREQ 位设置位有效，并且样品被传送。对于多种取样速率，每个输入通道的“标签”位指示数据是否有效。如表 35-2 所示显示标记位。

表 35-2 显示标记位

位	描述
19	保留（填充 0）
18-12	控制寄存器指数（填充 0，如果 AC97 标记是无效）
11	插槽 3 请求：左声道的 PCM
10	插槽 4 请求：右声道的 PCM
9	插槽 5 请求：不可用
8	插槽 6 请求：MIC 通道
7	插槽 7 请求：不可用
6	插槽 8 请求：不可用
5	插槽 9 请求：不可用
4	插槽 10 请求：不可用
3	插槽 11 请求：不可用
2	插槽 12 请求：不可用
1、0	保留（填充 0）

## （3）插槽 2：状态数据端口

插槽 2 带有 16 位分辨率的状态数据。([19:4]有效数据)。

## （4）插槽 3：左声道的 PCM

插槽 3 是音频输入结构左声道音频输出的 AC97 编解码器。如果样品的分辨率少于 16 位，AC97 编解码器填充所有的培训（training）的非有效位安置插槽 0 内。



(5) 插槽 4: 右声道的 PCM

插槽 4 是音频输入结构右声道音频输出的 AC97 编解码器。如果样品的分辨率少于 16 位，在插槽 0 中 AC97 编解码器填补了所有的训练非有效位位置。

(6) 插槽 6 : 麦克风记录数据

该 AC97 控制器只支持 16 位分辨率，用于麦克风接口通道。如图 35-5 (b) 所示，显示 AC97 控制器输入结构。

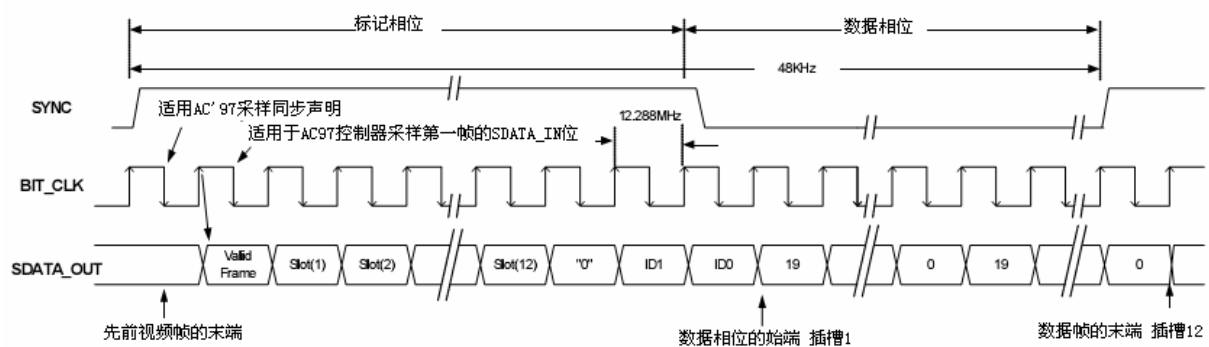


图 35-5 (b) AC97 控制器输入结构

## 35.4 AC97 掉电模式

### 1.AC-link 电源关闭

当 AC97 编解码器的电源关闭寄存器 (0x26) PR4 设置为 1 (写入 0x1000), AC-link 信号进入一个低功耗模式。主要的编解码器驱动 BITCLK 和 SDATA\_IN 入逻辑低电平的水平。序列时序图，如图 35-6 所示。

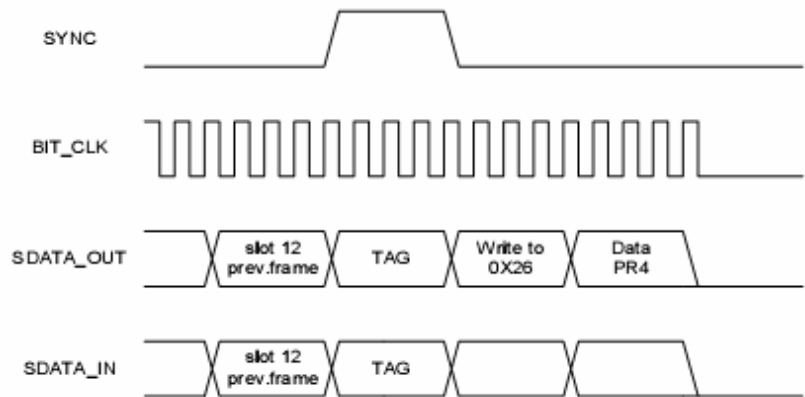


图 35-6 AC97 电源关闭时间

该 AC97 控制器的传输越过 AC-link 写入电源关闭寄存器 (0x26)。建立 AC97 控制器是为了当它写入关闭电源寄存器位 PR4 (data 0x1000) 时, 它不传送数据到插槽 3-12 的位置, 并当收到关闭电源请求时, 它不需要编解码器处理其它数据。当编解码器处理请求时, 它立即转换 BITCLK 和 SDATA\_IN 为逻辑低水平。设计 AC\_GLBCTRL 寄存器后, 改变 AC97 控制器驱动 SYNC 和 SDATA\_OUT 为逻辑低水平。

## 2. 唤醒 AC-link——通过 AC97 控制器触发唤醒

AC-link 协议提供了一个 AC97 冷复位和一个 AC97 热复位。目前省电状态最终发送 AC97 复位指令。在所有掉电模式下, 寄存器必须停在同一个状态, 除非执行一个 AC97 冷复位。在一个 AC97 冷复位后, AC97 被初始化为它的默认值。关闭电源后, AC—Link 必须等待该帧后的四个音频帧中的一个最小值。该帧被 SYNC 信号恢复前掉电。AC—Link 上电后, 它通过编解码准备位 (输入插槽 0, 位 15) 指示读准备就绪。如图 35-7 所示, 显示 AC97 关闭/启动电源流。

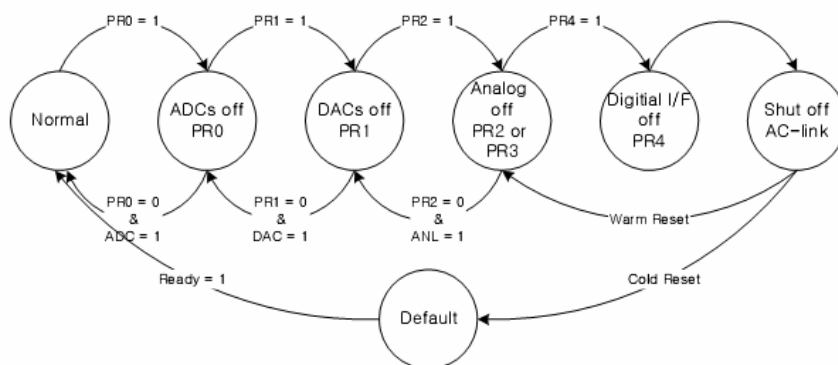


图 35-7 AC97 关闭/启动电源流

## 3. AC97 冷复位

当通过 AC\_CLBCTRL, nRESET 引脚是有效时, 产生一个冷复位。有效和无效 nRESET 启动 BITCLK 和 SDATA\_OUT。所有 AC97 控制寄存器都被初始化为它们默认启动复位值。nRESET 是一个异步 AC97 输入。

## 4. AC97 热复位

AC97 热复位重新启动 AC-link, 不需要改变当前 AC97 寄存器的值。当缺少 BITCLK 和高驱动 SYNC 时会生成一个热复位。在正规的音频结构中, SYNC 是一个同步的 AC97 输入。当 BITCLK 不存在, SYNC 被视为一个异步输入, 用来产生 AC97 热复位。该 AC97 控制器不能启动 BITCLK, 直到采样 SYNC 再次为低电平。

## 35.5 控制 AC97 的特殊寄存器

### 35.5.1. AC97 特殊功能寄存器概述

寄存器	地址	读/写	描述	复位值
AC_GLBCTRL	0x7F001000	读/写	AC97 通用控制寄存器。	0x00000000
AC_GLBSTAT	0x7F001004	读	AC97 全球状态寄存器。	0x00000001
AC_CODEC_CMD	0x7F001008	读/写	AC97 编解码器命令寄存器。	0x00000000
AC_CODEC_STAT	0x7F00100C	读	AC97 编解码器状态寄存器。	0x00000000
AC_PCMADDR	0x7F001010	读	AC97 的 PCM 输出/输入 通道 FIFO 地址寄存器。	0x00000000
AC_MICADDR	0x7F001014	读	AC97 的 MIC 输入通道 FIFO 地址寄存器。	0x00000000
AC_PCMDATA	0x7F001018	读/写	AC97 的 PCM 输出/输入 通道 FIFO 数据寄存器。	0x00000000
AC_MICDATA	0x7F00101C	读/写	AC97 的 MIC 输入通道 FIFO 数据寄存器。	0x00000000

### 35.5.2. AC97 通用控制寄存器（AC\_GLBCTRL）

这是 AC97 通用控制寄存器。有中断控制寄存器，DMA 的控制寄存器，AC-Link 控制寄存器，数据传输控制寄存器和相关的重置控制寄存器组成。

寄存器	地址	读/写	描述	复位值
AC_GLBCTRL	0x7F001000	读/写	AC97 通用控制寄存器。	0x00000000

AC_GLBCTRL	位	描述	初始状态
-	[31]	保留。	0
编解码器准备就绪中断清除	[30]	中断清除（只写）	0
PCM 输出通道欠载中断清除	[29]	中断清除（只写）	0

PCM 输入通道超限中断清除	[28]	中断清除（只写）	0
MIC 输入通道超限中断清除	[27]	中断清除（只写）	0
PCM 输出通道极限中断清除	[26]	中断清除（只写）	0
PCM 输入通道极限中断清除	[25]	中断清除（只写）	0
MIC 输入通道极限中断清除	[24]	中断清除（只写）	0
—	[23]	保留	0
编解码器准备就绪中断启动	[22]	0：禁止 1：启动。	0
PCM 输出通道超限中断启动	[21]	0：禁止 1：启动（FIFO 为空）	0
PCM 输入通道超限中断启动	[20]	0：禁止 1：启动（FIFO 为满）	0
MIC 输入通道超限中断启动	[19]	0：禁止 1：启动（FIFO 为满）	0
PCM 输出通道极限中断启动	[18]	0：禁止 1：启动（FIFO 为半空）	0
PCM 输入通道极限中断启动	[17]	0：禁止 1：启动（FIFO 为半满）	0
MIC 输入通道极限中断启动	[16]	0：禁止 1：启动（FIFO 为半满）	0
—	[15:14]	00：关闭    01：PIO    10：DMA    11：保留	00
PCM 输出通道传输模式	[13:12]	00：关闭    01：PIO    10：DMA    11：保留	00
PCM 输入通道传输模式	[11:10]	00：关闭    01：PIO    10：DMA    11：保留	00
麦克风在频道传输模式	[9:8]	00：关闭    01：PIO    10：DMA    11：保留	00
—	[7:4]	保留。	0000
使用 AC-link 传输数据启动	[3]	0：禁止 1：启动	0
AC-link 开关	[2]	0：关闭 1：SNYC 信号转移到编解码器	0

热复位	[1]	0: 正常 1: 从关闭电源唤醒编解码器	0
冷复位	[0]	0: 正常 1: 复位编解码器和控制器的逻辑	0

### 35. 5. 3. AC97 通用状态寄存器(AC\_GLBSTAT)

这是状态寄存器。当中断发生时，可以检查中断源是什么。

寄存器	地址	读/写	描述	复位值
AC_GLBSTAT	0x7F001004	读	AC97 通用状态寄存器	0x00000001

AC_GLBSTAT	位	描述	初始状态
–	[31:23]	保留。	0x00
编解码器准备就绪中断	[22]	0: 不请求 1: 请求	0
PCM 输出通道欠载中断	[21]	0: 不请求 1: 请求	0
PCM 输入通道超限中断	[20]	0: 不请求 1: 请求	0
MIC 输入通道超限中断	[19]	0: 不请求 1: 请求	0
PCM 输出通道极限中断	[18]	0: 不请求 1: 请求	0
PCM 输入通道极限中断	[17]	0: 不请求 1: 请求	0
MIC 输入通道极限中断	[16]	0: 不请求 1: 请求	0
–	[15:3]	保留。	0x000
控制主要状态	[2:0]	000: 闲置 001: 初始 010: 准备 011: 有效 100: LP 101: 热的	001

### 35. 5. 4. AC97 编解码器命令寄存器(AC\_CODEC\_CMD)

当控制写或读时，必须设定读取的启动位。如果想要将数据写入 AC97 编解码器中，可以设定该 AC97 编解码器和数据的指数（或地址）。

寄存器	地址	读/写	描述	复位值
-----	----	-----	----	-----

AC_CODEC_CMD	0x7F001008	读/写	AC97 编解码器命令寄存器	0x00000000
--------------	------------	-----	----------------	------------

AC_CODEC_CMD	位	描述	初始状态
–	[31:24]	保留。	0x00
读取启动	[23]	0：写命令 1：读取状态	0
地址	[22:16]	编解码器的命令地址	0x00
数据	[15:0]	编解码器的命令数据	0x0000

注：当命令被写在 AC\_CODEC\_CMD 寄存器上，一个命令到下一个命令之间有延时多于 1 / 48KHz。

### 35. 5. 5.AC97 编解码器的状态寄存器（AC\_CODEC\_STAT）

如果读取启动位是 1 和编解码器的命令地址是有效的，编解码器的状态数据也是有效的。

寄存器	地址	读/写	描述	复位值
AC_CODEC_STAT	0x7F00100C	读	AC97 编解码器的状态寄存器	0x00000000

AC_CODEC_CMD	位	描述	初始状态
–	[31:23]	保留	0x00
地址	[22:16]	编解码器的状态地址	0x00
数据	[15:0]	编解码器的状态数据	0x0000

注：如果想从 AC97 编解码器通过 AC\_CODEC\_STAT 寄存器读取数据，则按照以下步骤进行：

- （1）在 AC\_CODEC\_CMD 寄存器上，写入关于位[23]=1 的命令地址和命令数据。
- （2）有一定的延时时间。
- （3）在 AC\_CODEC\_STAT 寄存器中，读取命令地址和命令数据。

### 35.5.6. AC97 的 PCM 输出/输入通道 FIFO 数据寄存器(AC\_PCMADDR)

内部 PCM FIFO 地址索引。

寄存器	地址	读/写	描述	复位值
-----	----	-----	----	-----

AC_PCMADDR	0x7F001010	读	AC97 的 PCM 输出/输入通道 FIFO 数据寄存器。	0x00000000
------------	------------	---	--------------------------------	------------

AC_PCMADDR	位	描述	初始状态
–	[31:28]	保留。	0000
输出读取地址	[27:24]	PCM 输出通道 FIFO 读取地址。	0000
–	[23:20]	保留。	0000
输入读取地址	[19:16]	PCM 输入通道 FIFO 读取地址。	0000
–	[15:12]	保留。	0000
输出写入地址	[11:8]	PCM 输出通道 FIFO 写入地址。	0000
–	[7:4]	保留。	0000
输入写入地址	[3:0]	PCM 输入通道 FIFO 写入地址。	0000

### 35.5.7.AC97 的 MIC 输入通道 FIFO 地址寄存器

寄存器	地址	读/写	描述	复位值
AC_MICADDR	0x7F001014	读	AC97 的 MIC 输入通道 FIFO 数据寄存器。	0x00000000

AC_MICADDR	位	描述	初始状态
–	[31:20]	保留。	
读取地址	[19:16]	MIC 输入通道 FIFO 读取地址。	
–	[15:4]	保留。	
写入地址	[3:0]	MIC 输入通道 FIFO 写入地址。	

### 35.5.8.AC97 的 PCM 输出/输入通道 FIFO 数据寄存器(AC\_PCMDATA)

这是 PCM 输出/输入通道 FIFO 数据寄存器

寄存器	地址	读/写	描述	复位值
AC_PCMDATA	0x7F001018	读/写	AC97 的 PCM 输出/输入通道 FIFO 数据寄存器。	0x00000000

AC_PCMDATA	位	描述	初始状态
右数据	[31:16]	PCM 输出输入右通道 FIFO 数据。 读取：PCM 输入右通道。 写入：PCM 输出右通道。	0x0000
左数据	[15:0]	PCM 输出/输入左通道 FIFO 数据。 读取：PCM 输入左通道。 写入：PCM 输出左通道。	0x0000

**35.5.9.AC97 的 MIC 输入通道 FIFO 数据寄存器(AC\_MICDATA)**

这是 MIC 输入通道 FIFO 数据寄存器

寄存器	地址	读/写	描述	复位值
AC_MICDATA	0x7F00101C	读/写	AC97 的 MIC 输入通道 FIFO 数据寄存器。	0x00000000

AC_MICDATA	位	描述	初始状态
-	[31:16]	保留。	0x0000
单数据	[15:0]	MIC 输入单通道 FIFO 数据。	0x0000



## 36 IIS 总线接口

IIS 是一种常用的数字音频接口。总线只处理音频数据，像编码和控制这样的其它信号被转移分开。尽可能的在两个 IIS 总线之间传输数据。一个 3 线串行总线被用于组成一个为多元数据通道的路线、一个选择路线和一个时钟路线，以尽量减少必要的插槽和保持简单的配线。

IIS 接口传输或接受声音数据来自于外部立体声音频编解码器。用于传输和接收数据，包括两个  $32 \times 16$  FIFO 数据结构。DMA 传输模式能支持传输或接受样本。从内部系统时钟控制器通过 IIS 时钟分频器或直接时钟状态中提供 IIS 的特定时钟。

### 36.1 IIS 总线特征

在 IIS 总线接口包括以下功能：

- 2 通道 IIS 总线用于 DMA 装置的音频接口运作。
- 串行，8/16 位经通道数据传输。
- 支持 IIS，MSB-justified 和 LSB-justified 数据格式。

### 36.2 结构框图

对 IIS 总线结构框图的描述，如图 30-1 所示。

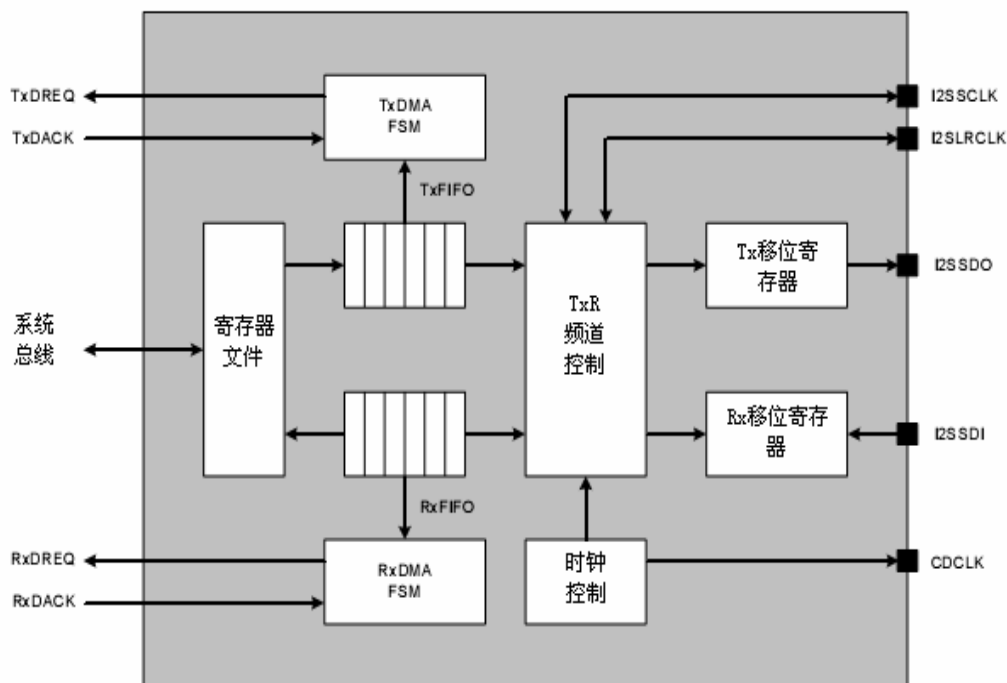


图 30-1 IIS 总线结构框图

### 36.3 功能说明

IIS 接口由寄存器层、FIFO、移位寄存器、时钟控制、DMA 有限状态设计和通道控制模块组成。如图 6-66 所示。每个 FIFO 有 32 位的宽度和 16 位深度构成，其中包括左/右通道数据。因此 FIFO 的访问和数据传输以左/右成对的单元进行操作。

#### 1. 主/从模式

可以通过 IISMOD 寄存器设置 IMS 位，来选择主/从模式。在主模式中，内部产生 I2SSCLK 和 I2SLRCLK 并提供给外部设备。因此区分产生 I2SSCLK 和 I2SLRCLK 需要一个启动时钟。IIS 预分频器以内部系统时钟分出的频率产生一个启动时钟。在外部主模式下，从 IIS 外部直接反馈启动时钟。在从模式中从引脚提供 I2SSCLK 和 I2SLRCLK。

主/从模式不同与发射/接收。主/从模式主要介绍 I2SLRCLK 和 I2SSCLK 的用法。I2SLRCLK（这个只是辅助）的用法并不重要。如果 IIS 总线接口传送 3 时钟信号到 IIS 编解码器，则 IIS 总线在主模式中。但如果 IIS 总线接口传输数据到 IIS 编解码器，这是 TX 模式。反过来说，IIS 总线接口从 IIS 编解码器接收

时钟信号，这是 RX 模式。发射/接收模式会显示数据流的方向。

如图 30-2 所示，显示在 IIS 时钟控制模块和系统控制器设置内部或外部主模式的启动路线。RCLK 表明，在外部的主模式下启动时钟能被提供给外部 IIS 编解码器。

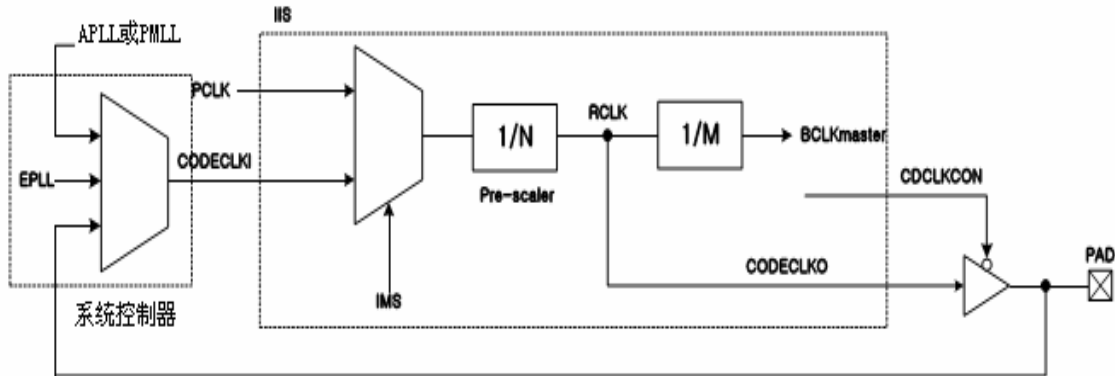


图 30-2 IIS 的时钟控制框图

## 2. DMA 传输

在 DMA 传输模式中，发送器或接收器 FIFO 能通过 DMA 控制器被访问。DMA 的服务请求由发送器或接收器 FIFO 的状态被激活。I2SCON 寄存器的 FTXEMPT、FRXEMPT、FTXFULL 和 FRXFULL 位代表发送器或接收器 FIFO 数据状态。特别是 FTXEMPT 和 FRXFULL 位，它们是对 DMA 服务请求准备好的标记；当发送 FIFO 不为空时，发送 DMA 服务请求有效；当接收 FIFO 为空时，接收 DMA 服务请求有效。对于单一数据，DMA 传输只用“握手”的方法。确认 DMA 被激活，则数据读取或写入操作必须执行。

\*参考：DMA 请求点

- 发送模式：（FIFO 不满）并且（TXDMACTIVE 有效）
- 接收模式：（FIFO 非空）并且（RXDMACTIVE 有效）

# 36.4 音频串行数据格式

## 1. IIS 的总线格式

IIS 总线有四线，包括串行数据输入 I2SSDI、串行数据输出 I2SSDO，左右声道选项时钟 I2SLRCLK 和串行位时钟 I2SDCLK；设备产生 I2SLRCLK 和 I2SBCLK 是主模式。

串行数据以 2 的补码形式传输，其中 MSB 有固定的位置，而 LSB 的位置要依据字长。发送器在一个时钟周期中发送下一个 MSB 后，I2SLRCLK 被改变。I2SLRCLK 改变后，发送器发送下一个字的 MSB。然而在串

行时钟信号的最主要优势上，串行数据必须被锁存到接收器里。因此数据传输对同步的最主要优势有一定的限制。

LR 通道选择线显示，该通道被传送。在任一尾随或领先的串行时钟上 I2SLRCLK 可能被改变，但它不是强制性的被对称。在从属模式内，该信号锁存时钟信号第一位的边。在 MSB 传输前，I2SLRCLK 线改变一个时钟周期。这允许从属发送器来驱动一个同步时速。进而，它使接收器来存储先前的字并为下一个字清除输入

## 2. MSB（左）的对齐

MSB 对齐（左对齐）格式与 IIS 总线格式类似，除了在 MSB 对齐格式里面，每当 I2SLRCLK 被改变，发送机始终在同一时间发送 MSB 的下一个消息。

## 3. LSB（右）的对齐

LSB 的对齐（右对齐）格式是相对 MSB 对齐的格式。换句话说，传输串行数据以 I2SLRCLK 移位的结束点对齐。

如图 30-3 所示，显示 IIS 的串行音频格式，MSB 对齐和 LSB 对齐。注意，在这个形式内，该该字节长度为 16 位和 I2SBCLK 的每 24 周期内 I2SLRCLK 产生传输（BFS 是  $48f_s$ ,  $f_s$  采样频率；I2SLRCLK 频率）。



	4. 0960	5. 6448	8. 1920	11. 2900	16. 3840	22. 5790	24. 5790	32. 7680	45. 1580	49. 1520
	768fs									
	6. 1440	8. 4672	12. 2880	16. 9340	24. 5760	33. 8690	36. 8640	49. 1520	67. 7380	73. 7280

### 5. IIS 的时钟映射表

I2SMOD 寄存器选择 BFS, RFS, 和 BLC 位, 必须参考下表。如表 30-2 所示, 显示允许时钟频率的映射关系。

表 30-2 IIS 的时钟映射表

时钟频率		RFS			
		256fs (00B)	512fs (01B)	384fs (10B)	768fs (11B)
BFS	16fs (10B)	(a)	(a)	(a)	(a)
	24fs (11B)	.	.	(a)	(a)
	32fs (00B)	(a) (b)	(a) (b)	(a) (b)	(a) (b)
	48fs (01B)	.	.	(a) (b)	(a) (b)
描述		(a) 允许 BLC 为 8 位 (b) 允许 BLC 为 16 位			

## 36.5 编程指南

在 IIS 总线接口可以被访问由处理器的使用程序的 I / O 的指示, 或由 DMA 的控制器。

### 1. 初始化

(1) 首先, 使用 IIS 总线接口, 必须配置 GPIO 到 IIS 模块并检验信号的方向。I2SLRCLK, I2SSCLK 和 I2SCDCLK 的输入输出型。每个 I2SSDI 和 I2SSDO 是一个输入和输出。

(2) 必须选择一个时钟源。S3C6410 有三个时钟源。PCLK, EPLL 和外部编解码器

### 2. DMA 的播放模式 (TX 模式)

(1) TXFIFO 直接操作。如果从发射/接受模式不区分主/从模式, 则必须考虑主/从模式和发射/接收模式。

(2) 正确地配置 I2SMOD 寄存器和 I2SPSR (IIS 预定标器)。

- (3) 为了操作系统稳定，DMA 操作前，内部 **RXFIFO** 必须至少有一个数据。
- (4) 通过轮询访问 SFR 检查 **RXFIFO** 的状态。
- (5) 如果 **RXFIFO** 不为空，启动 **RXDMACTIVE**（直接存储器存取通道）

36.6 IIS 总线接口的特殊寄存器

36.6.1. IIS 总线接口的特殊寄存器

寄存器	地址	读/写	描述	复位值
IISCON	0x7F002000 0x7F003000	读/写	IIS 的接口控制寄存器。	0xE00
IISMOD	0x7F002004 0x7F003004	读/写	IIS 的接口模式寄存器。	0x0
IISFIC	0x7F002008 0x7F003008	读/写	IIS 的接口 FIFO 控制寄存器。	0x0
IISPSR	0x7F00200c 0x7F00300c	读/写	IIS 的接口时钟分配器控制寄存器。	0x0
IISTXD	0x7F002010 0x7F003010	写	IIS 的接口传输数据寄存器。	0x0
IISRXD	0x7F002014 0x7F003014	读	IIS 的接口接收数据寄存器。	0x0

注：所有 IIS 接口的寄存器是易受影响的通过带有 STR/LDR 指令的字单位。

36.6.2. IISCON 寄存器

寄存器	地址	描述	初始值
IISCON	0x7F002000	IIS 的接口控制寄存器。	0x0000_0E00

	0x7F003000		
--	------------	--	--

IISCON	位	读/写	描述
	[31:12]	读/写	保留，置 0。
LRI	[11]	读	左/右通道时钟显示。注意：LRI 依赖 I2SMOD 寄存器的 LRP 位的值。  0：左（当 LRP 的位是低）或右（当 LRP 的位为高电平）。 1：右的（当 LRP 的位是低）或左（当 LRP 的位为高电平）。
FTXEMPT	[10]	读	Tx FIFO 空状态指示。  0：FIFO 不为空的（ 1：FIFO 为空
FRXEMPT	[9]	读	Rx FIFO 的空状态指示。  0：FIFO 不为空 1：FIFO 为空
FTXFULL	[8]	读	Tx FIFO 的充分状态指示。  0: FIFO 为不充分 1: FIFO 为充分
FRXFULL	[7]	读	Rx FIFO 的充分状态指示。  0: FIFO 为不充分（从通道准备接收数据） 1: FIFO 为充分（从通道不准备接收数据）
TXDMAPAUSE	[6]	读/写	Tx DMA 运行停止命令。注意：当该位在任何时间都有效时，当前运行的 DMA 传输完成后 DMA 请求将被挂起。  0: 不停止 DMA 操作 1: 停止 DMA 操作
RXDMAPAUSE	[5]	读/写	Rx DMA 运行停止命令。注意：当该位在任何时间都有效时，当前运行的 DMA 传输完成后 DMA 请求将被挂起。  0: 不停止 DMA 操作 1: 停止 DMA 操作



TXCHPAUSE	[4]	读/写	Tx 通道运行停止命令。注意：当该位在任何时间都有效时，当左右通道数据传输完成后通道操作将被挂起。 0：不停止运作 1：停止运作
RXCHPAUSE	[3]	读/写	Rx 通道运行停止命令。注意：当该位在任何时间都有效时，当左右通道数据传输完成后通道操作将被挂起。 0：不停止运作 1：停止运作
TXDMACTIVE	[2]	读/写	Tx DMA 的有效（启动 DMA 请求）。注：当该位被从高位设置为低位时，DMA 操作将立即被强制停止。 0：无效， 1：有效
RXDMACTIVE	[1]	读/写	Rx DMA 的有效（启动 DMA 请求）。注：当该位被从高位设置为低位时，DMA 操作将立即被强制停止。 0：无效， 1：有效
I2SACTIVE	[0]	读/写	IIS 接口有效（启动运作）。 0：无效， 1：有效

### 36. 6. 3. IISMOD 寄存器

寄存器	地址	描述	初始值
IISMOD	0x7F002004 0x7F003004	IIS 的接口模块寄存器。	0x0000_0000

IISCON	位	读/写	描述
	[31:13]	读/写	保留，置 0。
CDCLKCON	[12]	读/写	确定编解码器的时钟源。 0：使用内部编解码器的时钟源 1：从外部编解码芯片获得编解码器的时钟源

IMS	[11:10]	读/写	<p>IIS 主或（内部/外部）从模式选择。</p> <p>00：主模式（分模式，使用 PCLK）</p> <p>01：主模式（旁路模式，使用 I2SCLK）</p> <p>10：从模式（分模式，使用 PCLK）</p> <p>11：从模式（旁路模式，使用 I2SCLK）</p>
TXR	[9:8]	读/写	<p>发送或接收模式选择。</p> <p>00 ： 发送模式</p> <p>01 ： 只得到模式</p> <p>10 ： 发送和接收同步模式</p> <p>11 ： 保留</p>
LRP	[7]	读/写	<p>左/右通道时钟极性选择。</p> <p>0 ： 低为左声道和高为右声道</p> <p>1 ： 高为左声道和低为右声道</p>
SDF	[6:5]	读/写	<p>串行数据格式。</p> <p>00 ： IIS 的格式</p> <p>01 ： MSB 对齐（左对齐）格式</p> <p>10 ： LSB 的对齐（右对齐）格式</p> <p>11 ： 保留</p>
RFS	[4:3]	读/写	<p>IIS 根目录时钟（编解码器时钟）频率的选择。 00 ： 256 fs，fs 是采样频率</p> <p>01 ： 512 fs</p> <p>10 ： 384 fs</p> <p>11 ： 768 fs</p>
BFS	[2:1]	读/写	<p>位时钟频率的选择。</p> <p>00 ： 32 fs，fs 是采样频率</p> <p>01 ： 48 fs</p> <p>10 ： 16 fs</p> <p>11 ： 24 fs</p>
BLC	[1]	读/写	<p>为长度分组差错率通道。</p> <p>0: 16 位，1: 8 位</p>

36. 6. 4. IISFIC 寄存器

寄存器	地址	描述	初始值
IISFIC	0x7F002008 0x7F003008	IIS 的接口 FIFO 控制寄存器。	0x0000_0000

IISFIC	位	读/写	描述
	[31:16]	读/写	保留，置 0。
TFLUSH	[15]	读/写	TX FIFO 刷新命令。 0：不刷新 1：刷新
	[14:13]	读/写	保留，置 0。
FTXCNT	[12:8]	读	TX FIFO 数据计数。数值范围是 0~16。 N：FIFO 的数据计数 n
RFLUSH	[7]	读/写	RX FIFO 刷新命令。 0：不刷新 1：刷新
	[6:5]	读/写	保留，置 0。
FRXCNT	[4:0]	读	RX FIFO 数据计数。数值范围是 0~16。 N：FIFO 的数据计数 n

36. 6. 5. IISPSR 寄存器

寄存器	地址	描述	初始值
IISPSR	0x7F00200C 0x7F00300C	IIS 的接口时钟分配器控制寄存器。	0x0000_0000

IISPSR	位	读/写	描述
	[31:16]	读/写	保留，置 0。
PSRAEN	[15]	读/写	预定标器（时钟分频器）有效。

			0：无效 1：有效
	[14]	读/写	保留，置 0
PSVALA	[13:8]	读/写	预定标器（时钟分频器）一个分配的值 N：分配因数为 N+1。
	[7:]	读/写	保留，置 0。

36. 6. 6. IISTXD 寄存器

寄存器	地址	描述	初始值
IISTXD	0x7F002010 0x7F003010	IIS 的接口传输数据寄存器。	0x0000_0000

IISTXD	位	读/写	描述
ISSTXD	[31:0]	写	TX FIFO 写入数据。注意：左/右通道数据是作为以下位字段分配的： R[31:16]， L[15:0] 16 位 BLC R[23:16]， L[7:0] 8 位 BLC

36. 6. 7. IISRXD 寄存器

寄存器	地址	描述	初始值
IISRXD	0x7F002014 0x7F003014	IIS 的接口接收数据寄存器。	0x0000_0000

IISRXD	位	读/写	描述
ISSRXD	[31:0]	读	RX FIFO 写入数据。注意：左/右通道数据是作为以下位字段分配的： R[31:16]， L[15:0] 16 位 BLC R[23:16]， L[7:0] 8 位 BLC

## 37 PCM 音频接口

这节主要介绍 PCM 音频接口在 S3C6410X RISC 微处理器上的功能及使用。PCM 音频接口模块提供的 PCM 双向串行接口到一个外部编解码器。

该 PCM 音频接口包括以下特性：

- (1) 主模式：这个模块源于主移位时钟。
- (2) 所有 PCM 连续定时，选通脉冲和主要移位时钟是基于一个外部 PCM 音频时钟输入的。
- (3) 基于内部 APB PCLK 的可选时间。
- (4) 输入和输出 FIFO 到缓冲数据。
- (5) 可选的 DMA 接口为 Tx 和/或 Rx。

### 37.1 PCM 音频接口

PCM 音频接口提供了一个串行接口到外部编解码器。PCM 模块收到一个输入 PCMCODEC\_CLK，用来产生串行移位时间。PCM 接口输出一个串行数据，一个串行移位时间以及同步信号。通过一个串行输入线从外部接收数据。串行数据输入，串行数据输出和同步信号同步于串行的转变时钟。

串行移位时钟，PCMSCLK，它是由 PCMCODEC\_CLK 产生。同步信号 PCMSYNC 的产生是基于串行时钟的可以进行设计的数，并且是一个串行时钟的宽度。

PCM 数据字为 16 位宽，并且每 PCMSCLK 串行输出一位。每个 PCMSYNC，只有一个 16 位的字被移出。所有 16 位已被转移出来后 PCMSCLK 将继续切换。16 位字被完成后 PCMSOUT 数据将未被定义。下一个 PCMSYNC 将发信号给下一个 PCM 数据字的优先位。

Tx FIFO 提供 16 位数据字被串行转移出来。首先该数据首先被连续转移出 MSB，每 PCMSCLK 一位。PCM 连续输出数据 PCMSYNC 通过上升边缘连续被计时。有关的 MSB 位的位置是可编程的，其通过同步的 PCMSYNC 或者延后的 PCMCLK。16 位被移出后，一个中断产生，以指示传输结束。数据被移出，PCMSIN 输入用于从外部的编解码器连续移动数据。被接受的数据首先是 MSB，并且在 PCMSCLK 的下降边缘被锁定。第一位的位置可以和 PCMSYNC 同步或者 PCMSCLK 延后来被编程。

首 16 位是串行转移到 PCM\_DATAIN 寄存器中，随后它加载到 RX FIFO 中。后来的位被忽略直到下一个 PCMSYNC。

各种中断可以显示 RX 和 TX FIFO 的状态。当 CPU 需要服务 FIFO 时，每个 FIFO 的有一个可编程的编辑显示。为 RX FIFO，当 FIFO 超越一个可编程 `almost_full` 深度时，有一个中断将被提出。为 TX FIFO，同样有一个可编程 `almost_empty` 中断。

### 37.2 PCM 时序

以下显示，PCM 传输的时序关系。注意所有情况下，PCM 转移时序通过区分输入时钟被导出，PCMCODEC\_CLK。当时序是基于在 PCMCODEC\_CLK 上时，没有尝试重整输出 PCMSCLK 的上升边缘和原来的 PCMCODEC\_CLK 输入时钟。通过位置和方位测定系统，内部延迟将扭曲这些边缘，与分隔器的逻辑是一样的。这不代表一个问题，因为实际的转移时钟，PCMSCLK 是输出数据。如果 PCMSCLK 输出没有使用，歪曲率比 PCMCODEC\_CLK 的周期是不重要的。自从大部分的 PCM 接口在时钟下降边缘上夺取数据以来，它不能代表一个问题。

如图 37-1 所示，显示的 PCM 传输与 MSB 配置是符合 PCMSYNC。在 PCMCTL 寄存器置低位，MSB 配置符合设置 MSB\_POS\_WR 和 MSB\_POS\_RD 位。

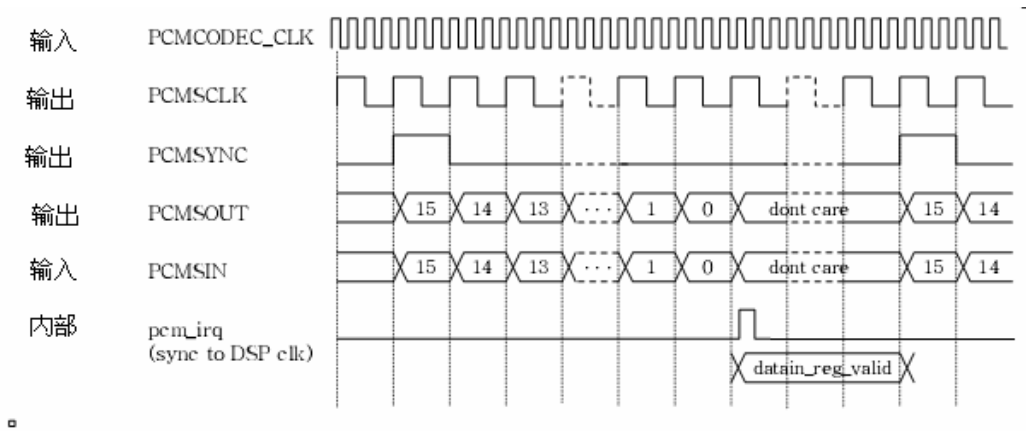


图 37-1 PCM 的时序，POS\_MSB\_WR/RD = 0

如图 37-2 所示，显示关于 MSB 配置的 PCM 传输，PCMSYNC 向后移位一个时钟。在 PCMCTL 寄存器置高位，MSB 配置符合设置 MSB\_POS\_WR 和 MSB\_POS\_RD 位。

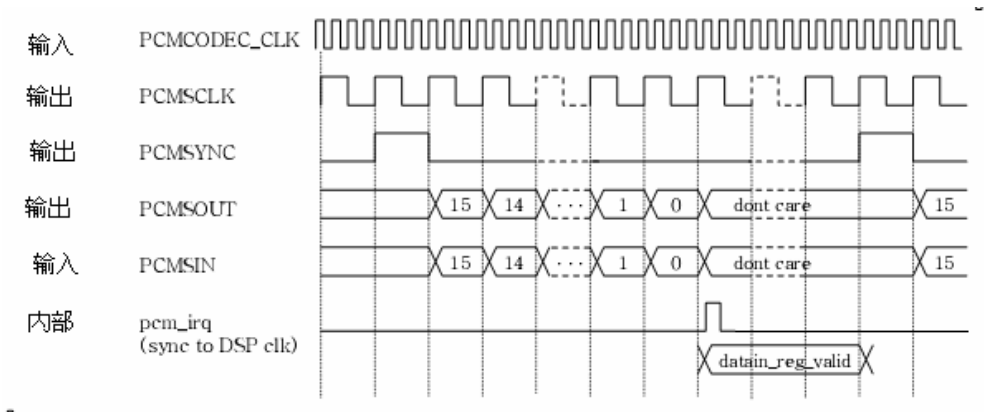


图 37-2 PCM 的时序, POS\_MSB\_WR/RD = 1

S3C6410X 可以提供多种时钟的 PCM。PCM 能选择两个时钟 PCLK 或 AUDIO（来自系统控制器）。我们还可以选择音频时钟里的 PLL 或外部输入时钟。如图 37-3 所示。

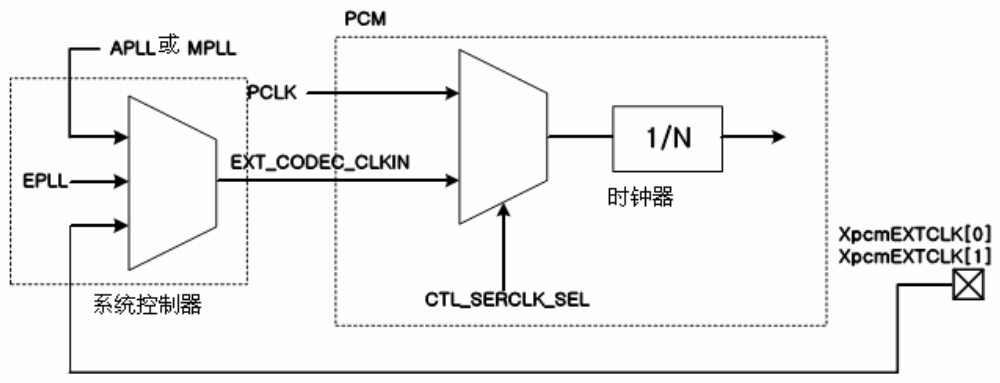


图 37-3 PCM 输入时钟图

### 37.3 PCM 寄存器

#### 37.3.1. PCM 寄存器概要

寄存器	地址	读/写	描述	复位值
PCM_CTL	0x7F009000 0x7F00A000	读/写	PCM 主控制器。	0x00000000

PCM_CLKCTL	0x7F009004 0x7F00A004	读/写	PCM 时钟和移位控制。	0x00000000
PCM_TXFIFO	0x7F009008 0x7F00A008	读/写	PCM Tx FIFO 写入端口。	0x00000000
PCM_RXFIFO	0x7F00900C 0x7F00A00C	读/写	PCM Rx FIFO 读取端口。	0x00000000
PCM_IRQ_CTL	0x7F009010 0x7F00A010	读/写	PCM 中断控制器。	0x00000000
PCM_IRQ_STAT	0x7F009014 0x7F00A014	读	PCM 中断状态。	0x00000000
PCM_FIFO_STAT	0x7F009018 0x7F00A018	读	PCM Tx 默认值。	0x00000000
PCM_CLRINT	0x7F009020 0x7F00A020	写	PCM 中断清除。	0x00000000

### 37.3.2. PCM 控制寄存器

PCM\_CTL 寄存器主要用于控制各个方面的 PCM 模块。它也提供一个状态位，该位用于提供用轮询代替基于中断的控制的选项。

寄存器	地址	读/写	描述	复位值
PCM_CTL	0x7F009000 0x7F00A000	读/写	PCM 主控制器	0x00000000

为 PCM\_CTL 控制寄存器的位定义说明如下：

PCM_CTL	位	描述	初始状态
Reserved	[31:19]	保留。	
TXFIFO_DIPSTICK	[18:13]	当 Almost_full, Almost_empty 的标记为 TXFIFO 启动时，使确定。  Almost_empty: <code>fifo_depth &lt; fifo_dipstick</code> 。	0



		<p>Almost_full:        fifo_depth    &gt;    (32    - fifo_dipstick)。</p> <p>注意: 如果 fifo_dipstick == 0 Almost_empty ,           Almost_full 是无效的。</p> <p>注: 对 TX FIFO 的 DMA 装载, Txfifo_dipstick &gt;=2。 PCM_TXDMA 几乎都用作 DMA 请求 (保持请求数据, 直到 FIFO 差不多满了为止), 该寄存器被请求。在 一些条件下, DMA 请求完成后, DMA 写入更多的字。 因此, FIFO 几乎为满的标志在只剩最少的空间用于 额外的字时有效。</p>	
RXFIFO_DIPSTICK	[12:7]	<p>当 almost_full, almost_empty 的标记为 RXFIFO 启动有效时, 使确定。</p> <p>Almost_empty:    fifo_depth &lt; fifo_dipstick 。</p> <p>Almost_full:        fifo_depth    &gt;    (32    - fifo_dipstick)。</p> <p>注意: 如果 fifo_dipstick == 0 Almost_empty ,           Almost_full 是无效的</p> <p>注: 对 DMA, RXFIFO_DIPSTICK 是不注意 RX FIFO 的 DMA 卸载的, 使用 rx_fifo_empty 标记作为 DMA 请求。</p>	0
PCM_TX_DMA_EN	[6]	<p>启动 DMA 接口为 TXFIFO</p> <p>DMA 必须在需求模式下操作。只要 TXFIFO 不是几 乎充分的, DMA_TX 将请求发生。</p>	0
PCM_RX_DMA_EN	[5]	<p>启动 DMA 接口为 RXFIFO</p> <p>DMA 必须在需求模式下操作。只要 RXFIFO 不是几 乎充分的, DMA_RX 将请求发生。</p>	0
TX_MSB_POS	[4]	<p>在串行输出流中, 控制 MSB 位的位置, 相对的 PCMSYNC 信号。</p> <p>0: MSB 发送在同一时钟, PCMSYNC 为高位</p>	0

		1: MSB 发送在下一个 PCMSCLK 周期后, PCMSYNC 为高位	
RX_MSB_POS	[3]	在串行输入流中, 控制 MSB 位的位置, 相对的 PCMSYNC 信号。  0: 在同一周期, MSB 被夺取在 PCMSCLK 的下降边缘上, PCMSYNC 为高位。  1: 在期间周期后, MSB 被夺取在 PCMSCLK 的下降边缘上, PCMSYNC 为高位。	0
PCM_TXFIFO_EN	[2]	启动 TXFIFO。	0
PCM_RXFIFO_EN	[1]	启动 RXFIFO。	0
PCM_PCM_ENABLE	[0]	PCM 启动信号。启动串行转移状态机。  启动必须设置为高位, 以便 PCM 的操作。  当启动为低, PCMSOUT 将不会切换。  此外, 当启动为低, 内部分隔计数器将被保存重置。  当启动为低, 内部 FIFO 清除和初始化。	0

37.3.3. PCMCLK 控制寄存器

寄存器	地址	读/写	描述	复位值
PCM_CLKCTL	0x7F009004 0x7F00A004	读/写	PCM 时钟和移位控制	0x00000000

定义为 PCM\_CLKCTL 控制寄存器说明如下:

PCM_CLKCTL	位	描述	初始状态
Reserved	[31:20]	保留。	
CTL_SERCLK_EN	[19]	启动串行时钟分工逻辑。  为 PCM 操作必须为高位 (如果是高位, 操作 SCLK 和 FSYNC)。	0
CTL_SERCLK_SEL	[18]	选择串行时钟的初始  0: 外部编解码器时钟输入	0

		1: PCLK	
SCLK_DIV	[17:9]	控制分频器来创建基于 PCMCODEC_CLK 的 PCMSCLK。时钟是 $\text{source\_clk} / 2 \times (\text{sclk\_div} + 1)$ 。	000
SYNC_DIV	[8:0]	控制 PCMSYNC 信号的频率基于 PCMSCLK 上。 PCMSYNC 频率 = PCMSCLK 频率 / (SYNC_DIV + 1)。	000

### 37.3.4. PCM TxFIFO 寄存器

寄存器	地址	读/写	描述	复位值
PCM_TXFIFO	0x7F009008 0x7F00A008	读/写	PCM Tx FIFO 写入端口	0x00000000

为 PCM\_TXFIFO 控制寄存器的定义位说明如下：

PCM_TXFIFO	位	描述	初始状态
Reserved	[31:17]	保留。	
TXFIFO_DVALID	[16]	TXFIFO 数据是有效的。 写：无效。 读：TXFIFO 读取数据有效。 1：有效 0：无效（可能读一个空的 FIFO）	0
TXFIFO_DATA	[15:0]	TXFIFO 数据。 写：TXFIFO 数据被写入 TXFIFO。 读：TXFIFO 被用于读 APB 界面。	0

### 37.3.5. PCM RxFIFO 寄存器

寄存器	地址	读/写	描述	复位值
PCM_RXFIFO	0x7F00900C 0x7F00A00C	读/写	PCM Rx FIFO 读取端口。	0x00000000

为 PCM\_RXFIFO 控制寄存器的定义位说明如下：

PCM_RXFIFO	位	描述	初始状态
Reserved	[31:17]	保留。	
RXFIFO_DVALID	[16]	RXFIFO 数据是有效的。 写：无效。 读：TXFIFO 读取数据有效。 1：有效 0：无效（可能读一个空的 FIFO）	0
RXFIFO_DATA	[15:0]	RXFIFO 数据 写：RXFIFO 数据被写入 TXFIFO。 读：RXFIFO 被用于读 APB 界面。 注：写 RXFIFO 意思是支持调试。	0

**37.3.6. PCM 中断控制寄存器**

寄存器	地址	读/写	描述	复位值
PCM_IRQ_CTL	0x7F009010 0x7F00A010	读/写	PCM 中断控制器。	0x00000000

为 PCM\_IRQ\_CTL 控制寄存器的定义位说明如下：

PCM_IRQ_CTL	位	描述	初始状态
Reserved	[31:15]	保留。	
EN_IRQ_TO_ARM	[14]	控制是否 PCM 中断发送到 ARM。 1：PCM IRQ 转交 ARM 子系统 0：PCM IRQ 不转交 ARM 子系统	0
Reserved	[13]	保留	0
TRANSFER_DONE	[12]	每次，一个字的串行移位完成，中断产生。	0

		1: IRQ 初始启动 0: IRQ 初始禁止	
TXFIFO_EMPTY	[11]	只要 TxFIFO 是空，产生中断。 1: IRQ 初始启动 0: IRQ 初始禁止	0
TXFIFO_ALMOST_EMPTY	[10]	只要 TxFIFO 几乎为空，产生中断。定义几乎为空，类似 FIXME 字保留。 1: IRQ 初始启动 0: IRQ 初始禁止	0
TXFIFO_FULL	[9]	TxFIFO 为充分，产生中断。 1: IRQ 初始启动 0: IRQ 初始禁止	0
TXFIFO_ALMOST_FULL	[8]	只要 TxFIFO 几乎为充分，产生中断。定义几乎为充分，类似 FIXME 字保留。 1: IRQ 初始启动 0: IRQ 初始禁止	0
TXFIFO_ERROR_STARVE	[7]	由于 TxFIFO starve 错误，中断产生。 当它仍是空白时，只要发生 TXFIFO 被读取就发生这个情况。 这被视为一种错误，并会有意外的结果。 1: IRQ 初始启动 0: IRQ 初始禁止	0
TXFIFO_ERROR_OVERFLOW	[6]	由于 TXFIFO 溢出错误，产生中断。 当它已经完整时，只要发生 TXFIFO 写入就发生这种情况。 这被视为一种错误，并会有意外的结果。 1: IRQ 初始启动	0

		0: IRQ 初始禁止	
RXFIFO_EMPTY	[5]	只要 RXFIFO 是空，产生中断。  1: IRQ 初始启动  0: IRQ 初始禁止	0
RXFIFO_ALMOST_EMPTY	[4]	只要 RxFIFO 几乎为空产生中断。定义 几乎为空类似 FIXME 字保留。  1: IRQ 初始启动  0: IRQ 初始禁止	0
RX_FIFO_FULL	[3]	只要 RxFIFO 为充分产生中断。  1: IRQ 初始启动  0: IRQ 初始禁止	0
RX_FIFO_ALMOST_FULL	[2]	只要 RxFIFO 几乎为充分 中断产生。 定义几乎为充分类似 FIXME 字保留  1: IRQ 初始启动  0: IRQ 初始禁止	0
RXFIFO_ERROR_STARVE	[1]	由于 TxFIFO STARVE 错误，中断产生。 当它仍是空白时，只要发生 RXFIFO 被读 取就发生这个情况。  这被视为一种错误，并会有意外的结 果。  1: IRQ 初始启动  0: IRQ 初始禁止	0
RXFIFO_ERROR_OVERFLOW	[0]	因为 RXFIFO 溢出错误产生中断。  当它已经完整时，只要发生 RXFIFO 写 入就发生这种情况。  这被视为一种错误，并会有意外的结 果。  1: IRQ 初始启动  0: IRQ 初始禁止	0

37.3.7. PCM 中断状态寄存器

PCM\_IRQ\_STAT 寄存器用于报告 IRQ 状态。

寄存器	地址	读/写	描述	复位值
PCM_IRQ_STAT	0x7F009014 0x7F00A014	读	PCM 中断状态。	0x00000000

为 PCM\_IRQ\_STAT 控制寄存器的定义位说明如下：

PCM_IRQ_STAT	位	描述	初始状态
Reserved	[31:14]	保留	
IRQ_PENDING	[13]	控制是否将 PCM 中断发送到 ARM 上。 1: PCM IRQ 被转发到 ARM 0: PCM IRQ 不被转发到 ARM	0
TRANSFER_DONE	[12]	每次串行移位的一个字完成，发生中断。 1: IRQ 初始启动 0: IRQ 初始禁止	0
TXFIFO_EMPTY	[11]	只要 RxFIFO 几乎为空，产生中断。 1: IRQ 初始启动 0: IRQ 初始禁止	0
TXFIFO_ALMOST_EMPTY	[10]	只要 TxFIFO 几乎为空，中断产生 1: IRQ 初始启动 0: IRQ 初始禁止	0
TXFIFO_FULL	[9]	只要 TxFIFO 为充分，产生中断。 1: IRQ 初始启动 0: IRQ 初始禁止	0
TXFIFO_ALMOST_FULL	[8]	只要 TxFIFO 几乎为充分产生中断。	0

		定义几乎为充分类似 FIXME 字保留。 1: IRQ 初始启动 0: IRQ 初始禁止	
TXFIFO_ERROR_STARVE	[7]	由于 TxFIFO STARVE 错误, 中断产生。 当它是仍是空白时, 只要发生 TxFIFO 被读取就发生这个情况。 这被视为一种错误, 并会有意外的结果。 1: IRQ 初始启动 0: IRQ 初始禁止	0
TXFIFO_ERROR_OVERFLOW	[6]	因为 TxFIFO 溢出错误产生中断。 当它已经完整时, 只要发生 TxFIFO 写入就发生这种情况。 这被视为一种错误, 并会有意外的结果。 1: IRQ 初始启动 0: IRQ 初始禁止	0
RXFIFO_EMPTY	[5]	只要 RXFIFO 是空, 产生中断。 1: IRQ 初始启动 0: IRQ 初始禁止	0
RXFIFO_ALMOST_EMPTY	[4]	只要 RxFIFO 几乎为空, 产生中断。定义几乎为空类似 FIXME 字保留。 1: IRQ 初始启动 0: IRQ 初始禁止	0
RX_FIFO_FULL	[3]	只要 RxFIFO 为充分, 产生中断。 1: IRQ 初始启动 0: IRQ 初始禁止	0
RX_FIFO_ALMOST_FULL	[2]	只要 RxFIFO 几乎为充分, 产生中断。 定义几乎为充分类似 FIXME 字保留	0



		1: IRQ 初始启动 0: IRQ 初始禁止	
RXFIFO_ERROR_STARVE	[1]	由于 TxFIFO STARVE 错误，产生中断。 当它仍是空白时，发生随时 RxFIFO。 这个被视为一个错误，并将会有想象不到的结果。  1: IRQ 初始启动 0: IRQ 初始禁止	0
RXFIFO_ERROR_OVERFLOW	[0]	因为 RXFIFO 溢出错误，产生中断。 当它已经完整时，只要发生 RXFIFO 写入就发生这种情况。 这被视为一种错误，并会有意外的结果。  1: IRQ 初始启动 0: IRQ 初始禁止	0

### 37.3.8. PCM FIFO 状态寄存器

PCM\_FIFO\_STAT 寄存器用于报告 FIFO 状态。

寄存器	地址	读/写	描述	复位值
PCM_FIFO_STAT	0x7F009018 0x7F00A018	读	PCM Tx 默认值。	0x00000000

为 PCM\_FIFO\_STAT 控制寄存器的定义位说明如下：

PCM_FIFO_STAT	位	描述	初始状态
Reserved	[31:20]	保留。	
TXFIFO_COUNT	[19:14]	显示 TXFIFO 的用法。	0
TXFIFO_EMPTY	[13]	显示 TXFIFO 是否为空。	0

TXFIFO_ALMOST_EMPTY	[12]	显示 TXFIFO 是否几乎为空。	0
TXFIFO_FULL	[11]	显示 TXFIFO 是否为充分。	0
TXFIFO_ALMOST_FULL	[10]	显示 TXFIFO 是否为几乎充分。	0
RXFIFO_COUNT	[9:4]	显示 RXFIFO 的用法。	
RXFIFO_EMPTY	[3]	显示 RXFIFO 是否为空。	0
RXFIFO_ALMOST_EMPTY	[2]	显示 RXFIFO 是否几乎为空。	0
RX_FIFO_FULL	[1]	显示 RXFIFO 是否为充分。	0
RX_FIFO_ALMOST_FULL	[0]	显示 RXFIFO 是否为几乎充分。	0

### 37.3.9. PCM 中断清除寄存器

PCM\_CLRINT 寄存器用于清除中断。中断服务程序负责清除中断访问。在这个寄存器上，可以写如任一清除中断。读取这个寄存器是不允许的。清除中断必须预先解决中断条件。另外，当这个中断可能被忽略后，其它中断将发生。

寄存器	地址	读/写	描述	复位值
PCM_CLRINT	0x7F009020	写	PCM 中断清除。	0x00000000
	0x7F00A020			

为 PCM\_CLRINT 控制寄存器的定义位说明如下：

PCM_CLRINT	位	描述	初始状态
Reserved	[31:1]	保留。	0
CLRINT	[0]	中断清除寄存器。	0

## 38 红外控制器

本节主要描述 S3C6410X RSIC 微处理器内的红外控制器的功能和用法。

### 38.1 概述

三星红外核心是无限系列通信控制器。三星红外核心支持两种不同类型的红外速度。此核心可以转换红外脉冲高达 4Mbps。它包括可配置的 FIFO 功能，用于减少 CPU 的负担。这使调整内部 FIFO 尺寸变得简单。

可以通过访问 16 个内部寄存器运行核心。当接收到红外脉冲时，核心可以检测到三种线性错误，如 CRC 错误，PHY 错误和有效和在长度错误。

#### 1.性能

红外线控制器支持以下性能：

- (1) 红外规格兼容
- (2) 红外 1.1 物理延迟规格
- (3) 在 MIR 和 FIR 模式下的 FIFO 操作（4Mbps，1.152Mbps 和 0.576Mbps）
- (4) 64 字节的 FIFO 尺寸
- (5) Back-to-Back 交易
- (6) 选择 Temic-IBM 和 HP 收发器的软件。

#### 2. 模块图

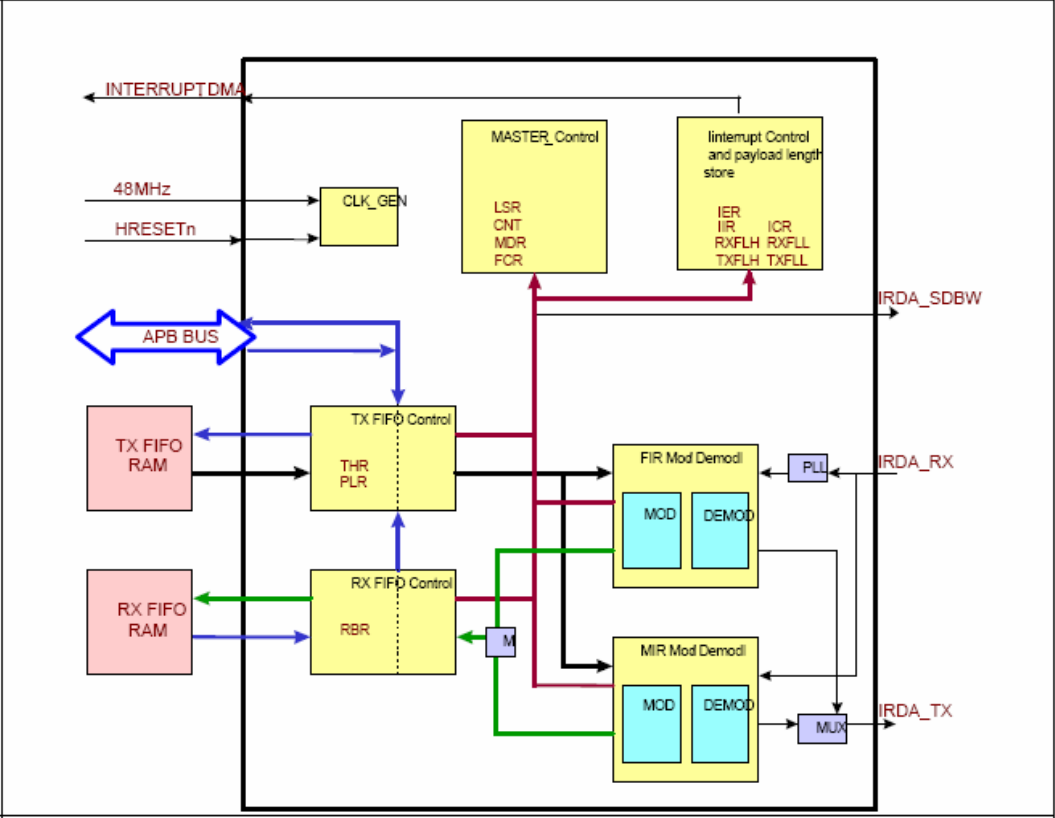


图 38-1 模块图

### 3. 外部接口信号

- IRDA\_TX : 红外 Tx 信号（输出）
- IRDA\_RX : 红外 Rx 信号（输入）
- IRDA\_SDBW : 红外收发器控制（输出）
- MCLK : 红外运行时钟；必须设置 SYSCON 内的红外时钟为 48MHz

组	名称	位	描述	源/目的地
红外特殊信号	MCLK	1	输入	SYSCON 源时钟：USB 时钟，PLL 时钟
	IrDA_Rx	1	输入	PAD
	IrDA_Tx	1	输出	PAD
	IrDA_SDBW	1	输出	PAD

## 38.2. 功能描述

### 1. 快速红外(FIR)模式（IRDA 1.1）

在快速红外模式（FIR）下，红外以 4Mbps 的波特速率传输。在数据传输模式下，核心将有效荷载数据解码为 4PPM 格式，并且将前同步信号、开始标志、CRC-32 和停止标志附加在解码的有效数据上，并成串的输出。在数据接收模式下，核心反向工作。首先，当检测到红外脉冲时，核心覆盖接收时钟并且移动前同步信号和停止标志。核心检测三个不同种类的错误，这些错误发生在传输的中间位置。这些错误是 PHY 错误，帧长度错误和 CRC 错误。当接收到整体有效荷载数据时可以检测到 CRC 错误。微控制器可以通过读取接收帧末尾的线性状态寄存器镜像接收帧的错误状态。

下面表格显示出 fir 数据帧的帧结构。

前同步信号	开始标志	连接层帧(有效荷载数据)	CRC32	停止标志
-------	------	--------------	-------	------

前同步信号：1000, 0000, 1010, 1000

开始标志：0000, 1100, 0000, 1100, 0110, 0000, 0110, 0000

停止标志：0000, 1100, 0000, 1100, 0000, 0110, 0000, 0110

前同步信号的数量是 10.

注：4PPM 编码

数据位对(DBP)	4PPM 数据象征 (DD)
00	1000
01	0100
10	0010
11	0001

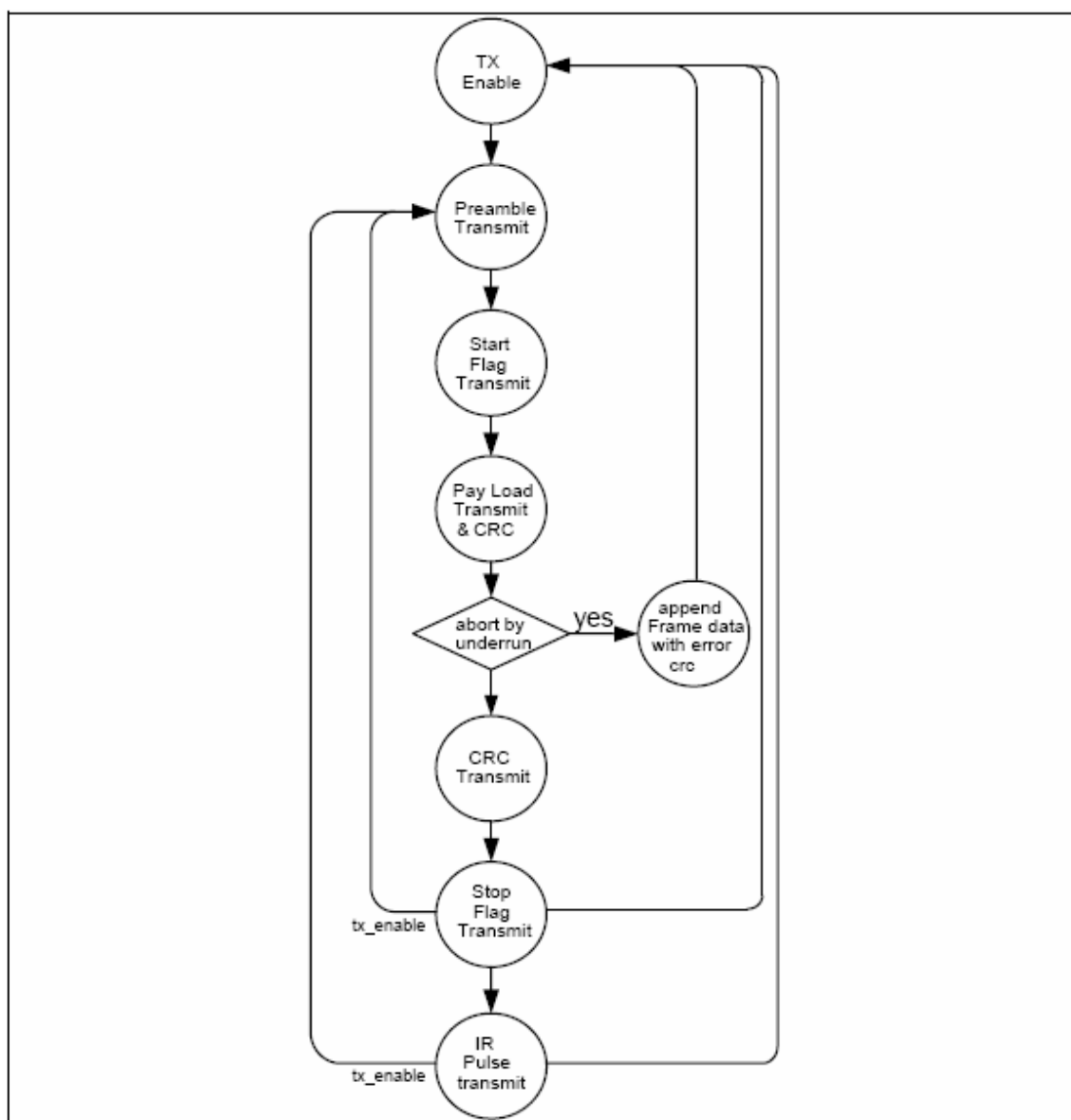


图 38-2 FIR 调制过程

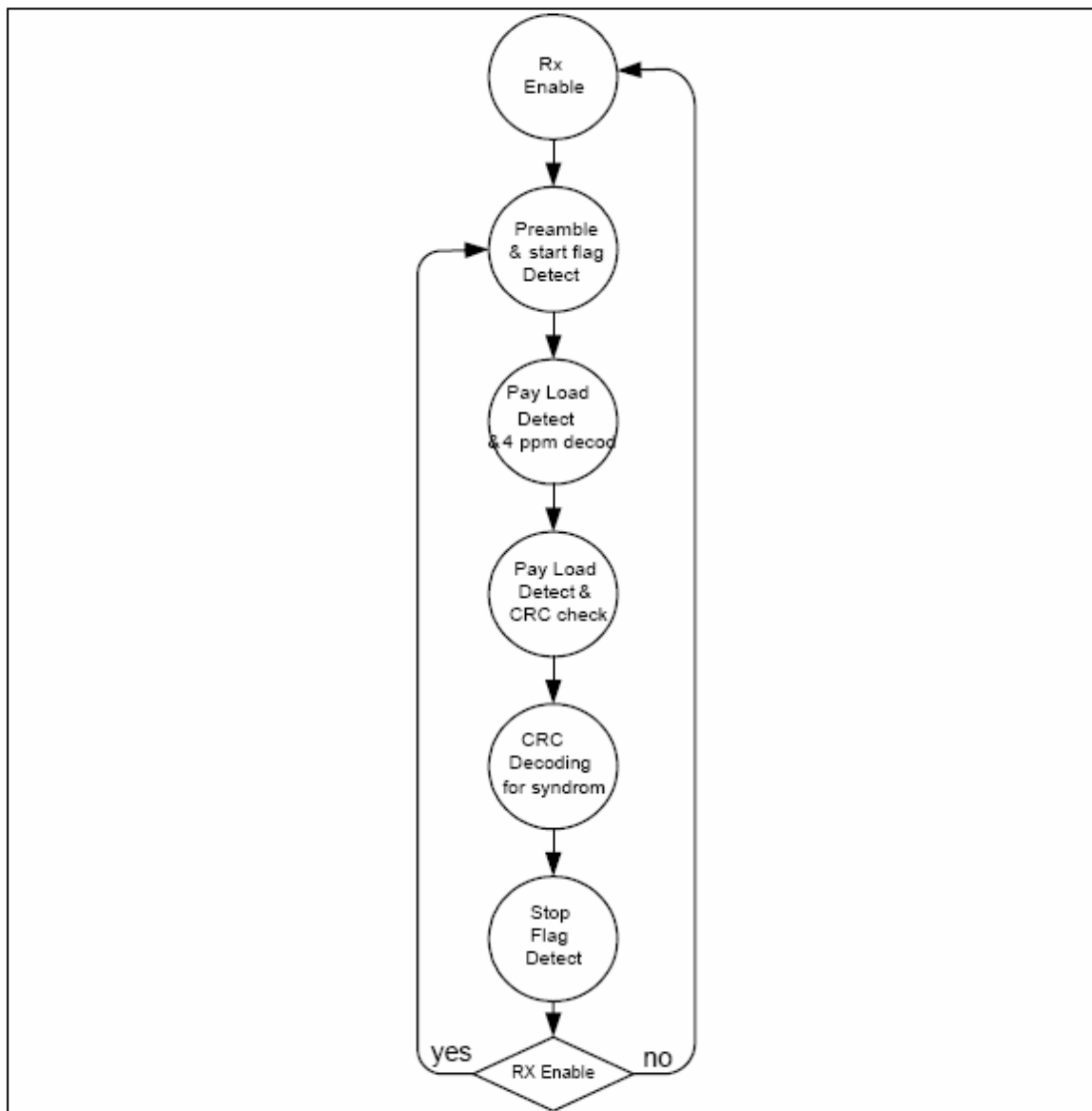


图 38-3 FIR 解调过程

上图显示了 FIR 解调状态机。当 IRD\_CNT 寄存器位 6 设置为逻辑高电平时，状态机开始。

## 2.中速红外(MIR) 模式 (IRDA 1.1)

在中速红外 (MIR) 模式下，红外以 1.152Mbps 和 0.576Mbps 的速率传输。有效荷载数据有开始标志、CRC16 和停止标志包裹。开始标志最小为 3 个字节。在传输和接收过程中，基本包和解包进程与 FIR 模式相同。中速模式需要一个位填充进程。MIR 模式下的位填充有核心插入 0 位。在接收模式下，必须转移

填充位。如 **FIR** 模式情况，可以通过读取 **IRDA\_LSR** 寄存器向接收模式内的微控制器报告三种不同的错误。

下面显示了 **MIR** 帧的数据结构

STA	STA	有效荷载数据	CRC16	STO
-----	-----	--------	-------	-----

**STA:** 开始标志，01111110 二进制

**CRC16:** CCITT 16 位 CRC

**STO:** 结束标志，01111110 二进制

**MIR** 脉冲通过 1/4 脉冲格式调制。图 38-4 显示脉冲产生。

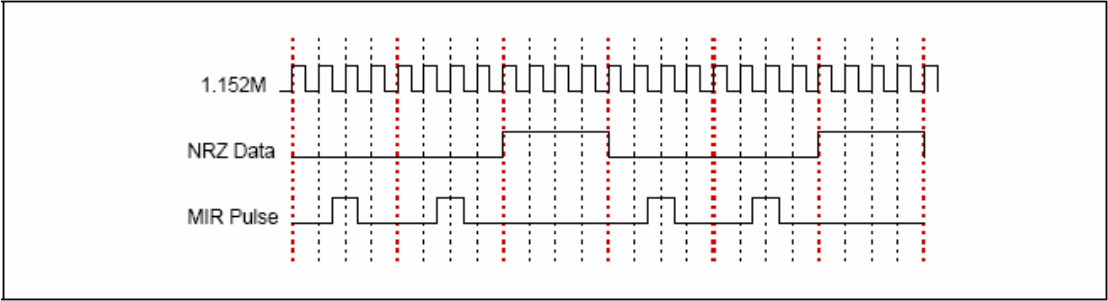


图 38-4 **MIR** 模式下的脉冲调制



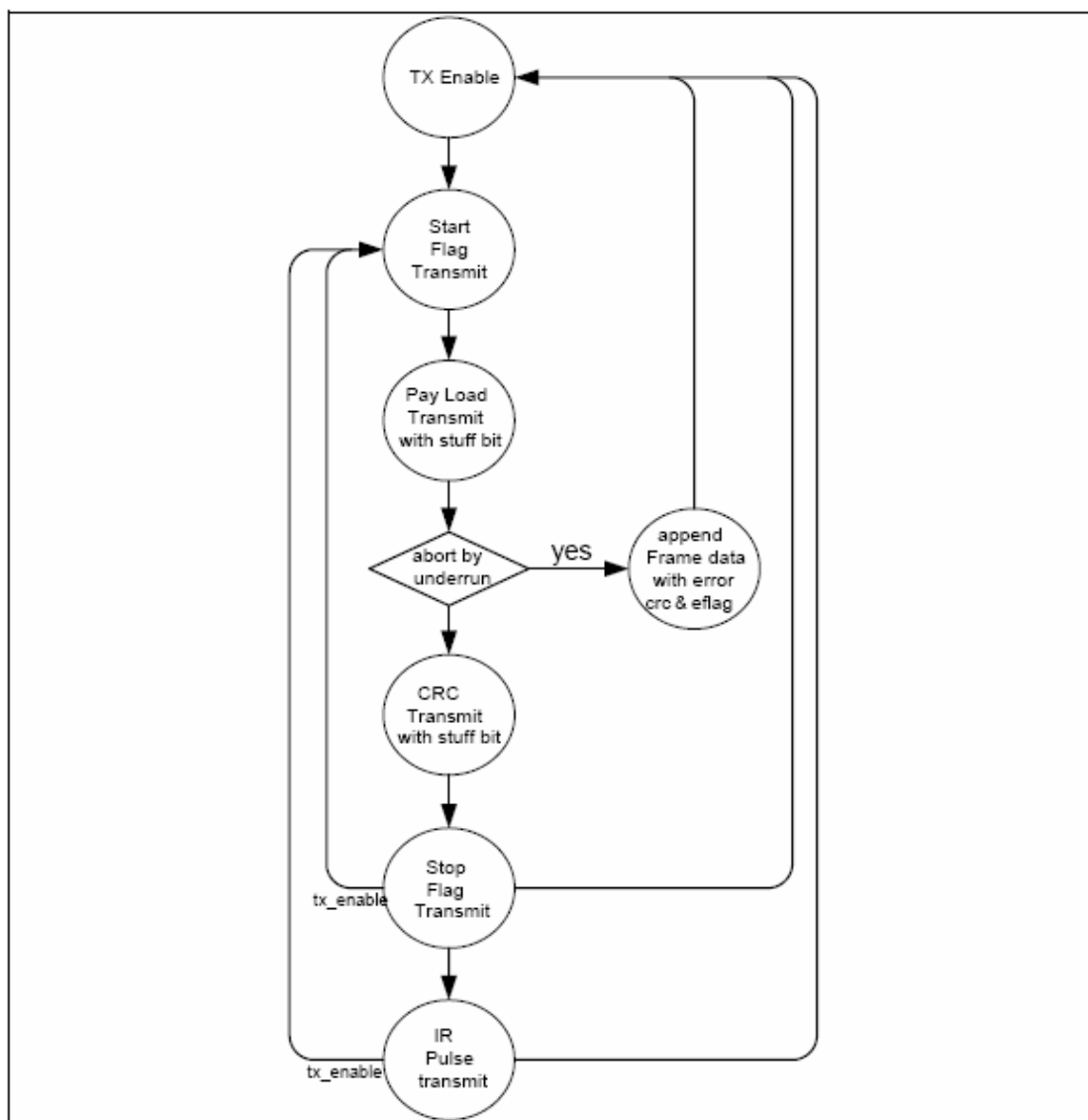


图 38-5 MIR 调制进程

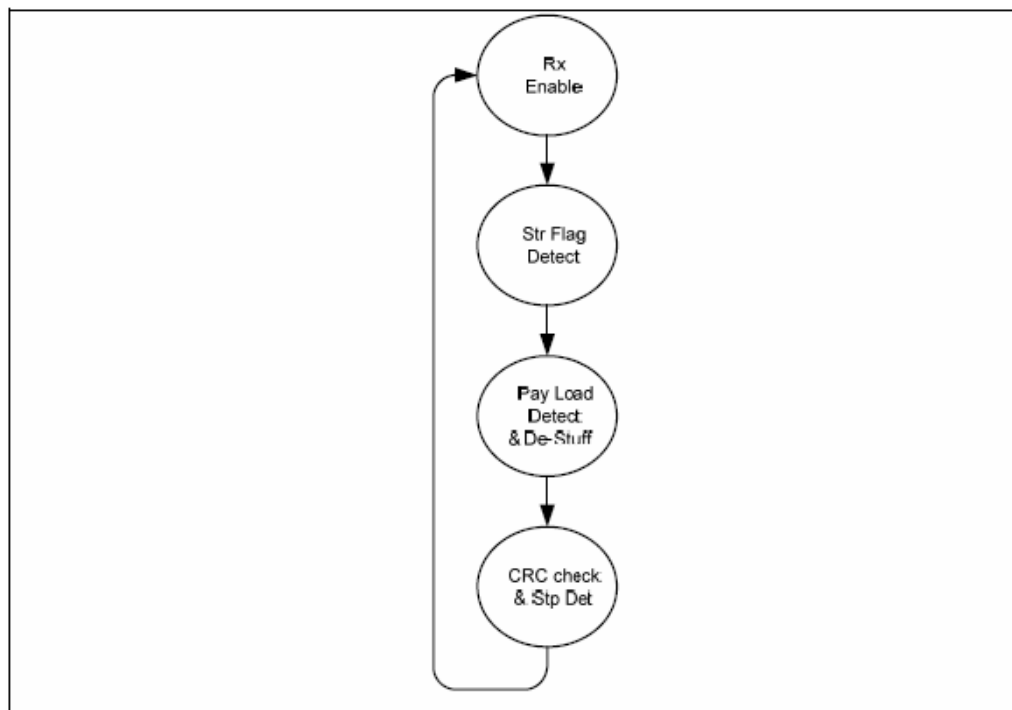


图 38-6 MIR 解调过程

### 3. 核心初始化程序

MIR/FIR 模式初始化操作

- (1) 运行 IRDA\_MDR 寄存器，选择 MIR/FIR 模式
- (2) 运行 IRDA\_CNT 寄存器，选择收发机类型  
对于 Temic-IBM 类型收发机，运行两次 IRDA\_CNT[0]=1'b0 和 IRDA\_CNT[0]=1'b1  
对于 HP 类型收发机，只运行一次 IRDA\_CNT[0]=1'b0。
- (3) 运行 IRDA\_PLR 寄存器，选择卡标志的数量和 TX 阈值的级别
- (4) 运行 IRDA\_RXFLL 和 IRDA\_RXFLH 寄存器（帧内的最大可用接收字节）
- (5) 运行 ERDA\_TXFLL 和 IRDA\_TXLH 寄存器（传输帧内的转换字节）
- (6) 运行 IRDA\_FCR 寄存器（FIFO 尺寸和 RX 阈值级别）
- (7) 运行 IRDA\_IER 寄存器（中断类型）
- (8) 运行 IRDA\_CNTjicunqi（TX 使能或 RX 使能）
- (9) 运行 IRDA\_IER 寄存器（中断使能）
- (10) 核心的服务中断信号

### 38.3 特殊功能寄存器

#### 38.3.1. 红外控制寄存器（IRDA\_CNT）

寄存器	地址	读/写	描述	复位值
IRDA_CNT	0x7F00_7000	读/写	红外控制寄存器	0x00

IRDA_CNT	位	描述	初始状态
TX enable	[7]	Tx 使能。位 7 必须设置为 1 来使能 MIR/FIR 红外模式内的数据传输	0
RX enable	[6]	Rx 使能。位 6 必须设置为 1 来使能所有 MIR/FIR 红外模式内的数据接收	0
Core loop	[5]	Tx 使能。位 7 必须设置为 1 来使能 MIR/FIR 红外模式内的数据传输	0
MIR half mode	[4]	MIR 半模式。当位 4 设置为 1 时，MIR 模式下的操作速度的有 1.152Mbps 转变到 0.576Mbps	0
Send IR pulse	[3]	发送 1.6us IR 脉冲。当 IRDA_MDR[4]位设置为 1 时，CPU 向此位写入 1，传输接口设备在帧结尾发送 1.6 us IR 脉冲。位 3 通过传输接口设备在 1.6us IR 脉冲数据结尾的传输进行自动清除。	0
Reserved	[2]	保留	0
Frame abort	[1]	帧中断。CPU 可以通过向位 1 写入 1 故意中断帧数据传输。接收机可以发现 FIR 模式内的帧的中断形式，和 FIR 模式内的 PHY 错误。在传输下一帧之前，CPU 必须复位 TX FIFO 和复位此位，通过向位 1 写入 0 完成复位。	0
SD/BW	[0]	此信号控制着 IRDA_SDBW 输出信号。用于控制 IRDA 收发机的模式。	0

38.3.2.红外模式定义寄存器(IRDA\_MDR)

寄存器	地址	读/写	描述	复位值
IRDA_MDR	0x7F00_7004	读/写	红外模式定义寄存器	0x00

IRDA_MDR	位	描述	初始状态
Reserved	[7:5]	保留	0
SIP Select	[4]	SIP 选择方法。如果此位设置为 1，同时 IRDA_CNT[3]设置为 1，SIP 脉冲附于 FIR/MIR TX 帧的末尾。同样，当此位设置为 0 是，SIP 产生在每个 FIR/MIR 帧的末尾。如果 IRDA_CNT[3]位设置为 0，设置此位为 1 不会产生 SIP。SIP 产生可以随着 IRDA_CNT[3]位被控制。	0
Temic select	[3]	位 3 是 Temic 收发机选择位。当位 3 清除为 0 时，核心在 temic 收发机模式下自动选择。	0
Reserved	[2:1]	保留	00
Mode select	[0]	选择操作模式为：  0： FIR 模式      1： MIR 模式	

38.3.3.红外中断/DMA 配置寄存器（IRDA\_CNF）

寄存器	地址	读/写	描述	复位值
IRDA_CNF	0x7F00_7008	读/写	红外中断/DMA 配置寄存器	0x00

IRDA_CNF	位	描述	初始状态
Reserved	[7:4]	保留	0
DMA Enable	[3]	0： DMA 禁止              1： DMA 使能	0

DMA Mode	[2]	0: Tx DMA                      1:Rx DMA	0
Reserved	[1]	保留	00
Interrupt enable	[0]	位 0 使能中断输出信号 0: 禁止      1: 使能	0

### 38.3.4.红外中断使能寄存器（IRDA\_IER）

寄存器	地址	读/写	描述	复位值
IRDA_IER	0x7F00_700C	读/写	红外中断使能寄存器	0x00

IRDA_IER	位	描述	初始状态
Last byte to Rx FIFO	[7]	使能状态说明中断。当最后字节写入 RX FIFO	0
Error indication	[6]	数据接收模式下的使能错误装填说明中断	0
Tx Underrun	[5]	使能传输欠载运行中断	0
Last byte detect	[4]	检测停止标志中断使能。如果此位设置为 1，当接收的数据帧最后字节进入解调模块时中断信号被激活，然后 CRC 解码完成。	0
Rx overrun	[3]	使能接收机过载中断	0
Last byte read from Rx FIFO	[2]	位 2 从产生的 RX FIFO 中断内使能最后字节，当微控制器重 RX FIFO 内读取最后帧字节时。	0
Tx FIFO below threshold	[1]	位 1 使能 TX FIFO 低于阈值级别中断，当 TX FIFO 内的可用空闲空间超过阈值级别时。	00
Rx FIFO over threshold	[0]	位 0 使能 RX FIFO 接收的数据超过阈值级别中断，当 RX FIFO 等于或大于阈值级别时。	0

38.3.5. 红外中断坚定寄存器（IRDA\_IIR）

寄存器	地址	读/写	描述	复位值
IRDA_IIR	0x7F00_7010	读/写	红外中断鉴定寄存器	0x00

IRDA_IIR	位	描述	初始状态
Last byte to Rx FIFO	[7]	向 RX FIFO 中断悬挂写入的最后字节。当帧的最后有效荷载数据字节下载到 RX FIFO 内时， 位 7 设置为 1.设置位 7 比位 2 优先，因为位 7 读取时清除	0
Error indication	[6]	接收行错误指明。位 6 设置为 1，如过在 RX 进程中三种错误有一种发生。使相应的终端使能位激活，PHY, CRC 和帧长度错误中的一种可以设置此位被激活。当清除错误源时，位 6 被清除。	0
Tx Underrun	[5]	传输欠载运行中断悬挂。当相应中断使能位激活时，如果 TX FIFO 内发生欠载运行，位 5 被设置为 1.服务欠载运行将清除位 5。	0
Last byte detect	[4]	检测帧中断悬挂的最后字节。如果相应中断使能位被激活，当解套块检测到接收帧很的最后字节时，位 4 被设置为 1，同时完成 CRC 解码。当位 4 读取操作时，位 4 被清除。	0
Rx overrun	[3]	RX FIFO 过载中断。当相应中断使能位被设置,位 3 被激活,当 RX FIFO 内发生过载时，位 3 被设置为 1.服务过载可以清楚位 3.	0
Last byte read from Rx FIFO	[2]	RX FIFO 最后字节读取中断。当相应中断使能位被激活，CPU 从 RX FIFO 内读取帧的最后字节时 位 2 被设置为 1.	0
Tx FIFO below threshold	[1]	TX FIFO 低于阈值中断悬挂。当传输 FIFO 级别低于阈值级别时，位 1 被设置为 1.	0
Rx FIFO over threshold	[0]	RX FIFO 大于阈值中断悬挂。当接收 FIFO 级别等于或高于阈值级别时，位 0 被设置为 1.	0

38.3.6. 红外行状态寄存器（IRDA\_LSR）

寄存器	地址	读/写	描述	复位值
IRDA_LSR	0x7F00_7014	读/写	红外行状态寄存器	0x83

IRDA_LSR	位	描述	初始状态
Tx empty	[7]	发射机空。当 TX FIFO 为空或者发射机前端闲置时设置此位为 1.	0
Reserved	[6]	保留	0
Received last byte from RX FIFO	[5]	从 RX FIFO 内接收的最后字节。当为控制器从 RX FIFO 内读取帧的最后字节时设置此位为 1，当 MCU 读取 IRDA_LSR 寄存器时清除此位。	0
Frame lengtherror	[4]	帧长度错误。当怎草果 IRDA_RXFLL 预先定义的最大帧长度时，此位设置为 1，IRDA_RXFLH 寄存器被接收。当位控制器读取 IRDA_LSR 寄存器时，此位被清除。当检测到此错误时，当前帧接收被终止。数据接收停止，直到检测到下一个 BOF。当 IRDA_LSR 寄存器由微控制器读取时，清除位 4.	0
PHY error	[3]	PHY 错误。在 FIR 模式下，当接收到非法的 4PPM 样本时此位设置为 1.在 IRDA_MIR 模式下，如果在接收期间接收到一个中止模式，设置此位为 1.当微控制器读取 IRDA_LSR 寄存器时，此位被清除。	0
CRC error	[2]	CRC 错误。 当在数据接收过程中发现一个坏的红外 CRC 时，此位设置为 1.当微控制器读取 LSR 寄存器时，清除此位为 0.	0
Reserved	[1]	保留	1
Rx FIFO empty	[0]	RX FIFO 空。指明 RX FIFO 为空。当 RX FIFO 状态变为空时，设置此位为 1，当 RX FIFO 为非空时设置此位为 0.	0

38.3.7. 红外 FIFO 控制寄存器（IRDA\_FCR）

寄存器	地址	读/写	描述	复位值
IRDA_FCR	0x7F00_7018	读/写	红外 FIFO 控制寄存器	0x00

IRDA_FCR	位	描述	初始状态															
RX FIFO Trigger level select	[7:6]	<div>接收机 FIFO 触发电平选择</div> <table><tr><td>位 7</td><td>位 6</td><td>64 字节 RX FIFO</td></tr><tr><td>0</td><td>0</td><td>01</td></tr><tr><td>0</td><td>1</td><td>16</td></tr><tr><td>1</td><td>0</td><td>32</td></tr><tr><td>1</td><td>1</td><td>56</td></tr></table>	位 7	位 6	64 字节 RX FIFO	0	0	01	0	1	16	1	0	32	1	1	56	00
位 7	位 6	64 字节 RX FIFO																
0	0	01																
0	1	16																
1	0	32																
1	1	56																
Reserved	[5]	必须设置为 1	0															
TX FIFO Clear Notification	[4]	当 FIFO 清除超过时，此位被激活。通过 CPU 读取此寄存器将清除此位。	0															
RX FIFO Clear Notification	[3]	当 FIFO 清除超过时，此位被激活。通过 CPU 读取此寄存器将清除此位。	0															
TX FIFO reset	[2]	TX FIFO 复位。当此位设置为 1，位 2 清除转换机 FIFO 内的所有字节，并且复位它的计数为 0.向位 2 写入 1 将自动清除。	0															
RX FIFO reset	[3]	RX FIFO 复位。当设置此位为 1 时，位 1 清除转换机 FIFO 内的所有字节，并且复位它的计数为 0.向位 2 写入 1 将自动清除。	0															
FIFO enable	[0]	FIFO 使能。当此位设置为 1，位 0 使能转换器和接收机 FIFO。当设置其他 IRDA_FCR 位时，位 0 必须设置为 1.改变位 0 将清除 FIFO .	0															

### 38.3.8.红外前同步信号长度寄存器（IRDA\_PLR）

寄存器	地址	读/写	描述	复位值
IRDA_PLR	0x7F00_701C	读/写	红外前同步信号长度寄存器	0x12

IRDA_PLR	位	描述	初始状态
Reserved	[7:6]	保留	00



TX FIFO Trigger level select	[5:4]	收发机 FIFO 触发电平选择			01
		位 5	位 6	64 字节 RX FIFO	
		0	0	保留	
		0	1	48	
		1	0	32	
		1	1	08	
Number of start flags in MIR 模式	[3:0]	MIR 模式下的开始标志数。在帧的开始进行传输的开始标志数应该等于 IRDA_PLR[3:0]的值，最小值为 3.			0010

### 38.3.9.红外接收机和发射机缓冲区寄存器（IRDA\_RBR）

寄存器	地址	读/写	描述	复位值
IRDA_PBR IRDA_THR	0x7F00_7020	读/写	红外接收机和发射机缓冲区寄存器	0x00

IRDA_RBR	位	描述	初始状态
Rx/Tx data	[7:0]	接收的数据（读取数据时） 传输的数据（写入数据时）	0x00

### 38.3.10.留在 TX FIFO 内的数据字节的红外总数（IRDA\_TXNO）

寄存器	地址	读/写	描述	复位值
IRDA_TXNO	0x7F00_7024	读	留在 TX FIFO 内数据字节的总数	0x00

IRDA_TXNO	位	描述	初始状态
Tx data total numbetr	[7:0]	留在 TX FIFO 内数据字节的总数	0x00

38.3.11.留在 RX FIFO 内的数据字节的红外总数（IRDA\_RXNO）

寄存器	地址	读/写	描述	复位值
IRDA_RXNO	0x7F00_7028	读	留在 RX FIFO 内数据字节的总数	0x00

IRDA_RXNO	位	描述	初始状态
Rx data total numbetr	[7:0]	留在 RX FIFO 内数据字节的总数	0x00

38.3.12. 红外转换帧 长度寄存器低（IRDA\_TXFLL）

寄存器	地址	读/写	描述	复位值
IRDA_TXFLL	0x7F00_702C	读/写	红外转换帧长度寄存器低	0x00

IRDA_TXFLL	位	描述	初始状态
Tx frame length low	[7:0]	TXFLL 储存被转换帧的字节数的低 8 位。	00

38.3.13. 红外转换帧 长度寄存器高（IRDA\_TXFLH）

寄存器	地址	读/写	描述	复位值
IRDA_TXFLH	0x7F00_7030	读/写	红外转换帧长度寄存器高	0x00

IRDA_TXFLH	位	描述	初始状态
Tx frame length high	[7:0]	TXFLH 储存被转换帧的字节数的高 8 位。	00

38.3.14. 红外接收帧长度寄存器低（IRDA\_RXFLL）

寄存器	地址	读/写	描述	复位值
IRDA_RXFLL	0x7F00_7034	读/写	红外接收帧长度寄存器低	0x00

IRDA_RXFLL	位	描述	初始状态
Rx frame length low	[7:0]	TXFLL 储存接收到的帧的最大字节数的低 8 位。	00

38.3.15. 红外接收帧 长度寄存器高（IRDA\_RXFLH）

寄存器	地址	读/写	描述	复位值
IRDA_RXFLH	0x7F00_7038	读/写	红外接收帧长度寄存器高	0x00

IRDA_RXFLH	位	描述	初始状态
Reserved	[7:6]	保留	00
Rx frame length high	[5:0]	TXFLH 储存被接收帧的最大字节数的高 6 位。	00

38.3.16. 红外中断清除寄存器（IRDA\_INTCLR）

寄存器	地址	读/写	描述	复位值
IRDA_INTCLR	0x7F00_703C	读/写	红外中断清除寄存器	

IRDA_INTCLR	位	描述	初始状态
Interrupt Clear	[31:0]	读未定义。写入任何值将导致红外中断清除。	

## 39 ADC 和触摸屏接口

本小节主要介绍 ADC 及触摸屏在 S3C6410 RISC 微处理器上的功能及其使用。

10位CMOS的ADC（模数转换器）是一种循环类型的装置，具有8位通道模拟输入。它将模拟的输入信号转换成10位二进制数字编码，最大转换率是500KSPS和2.5MHz的ADC时钟。ADC转换器的操作带有片上采样保持功能。电源中断模式的支持。

触摸屏接口控制触摸屏的位置和方位（XP, XM, YP, YM），为 X 坐标转换和 Y 坐标转换选择触摸屏的位置和方位（XP, XM, YP, YM）。触摸屏界面包含了位置和方位控制逻辑、ADC 界面逻辑和中断发生逻辑。

### 39.1 ADC 及触摸屏的特性

ADC 及触摸屏接口包括以下功能：

- 分辨率：10 位。
- 微分线性误差： $\pm 1.0 \text{ LSB}$ 。
- 积分线性误差： $\pm 2.0 \text{ LSB}$ 。
- 最高转换率：500kSPS。
- 低耗电量。
- 供电电压：3.3V。
- 模拟输入范围：0~3.3V。
- 对芯片采样保持功能。
- 正常转换模式。
- 单独的 X / Y 坐标的转换模式。
- 自动（顺序）的 X / Y 坐标的转换模式。
- 等待中断方式。
- 停止模式唤醒源。

## 39.2 ADC 及触摸屏界面操作

显示 ADC 和触摸屏接口的功能结构框图，如图 39-1 所示。ADC 的装置是一个循环的类型。

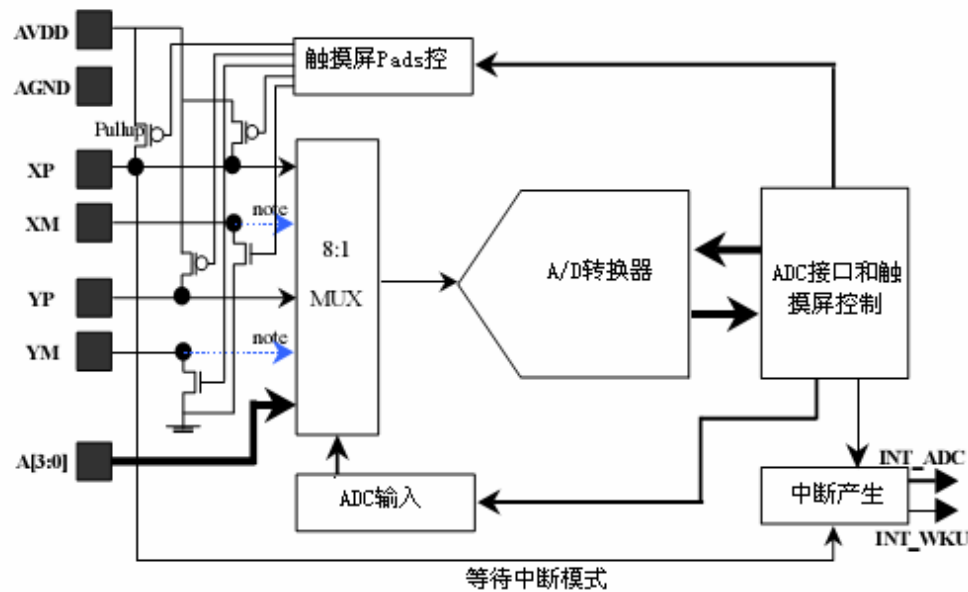


图 39-1 ADC 和触摸屏接口的功能结构框图

当触摸屏装置被使用，触摸屏的 I/F, XM 或 YM 只接地。当触摸屏的装置未被使用，为正常 ADC 转换，XM 或 YM 是连接模拟输入信号的。

## 39.3 功能描述

### 1. A/D 转换时间

当 GCLK 频率是 50 MHz 和分频器值是 49 时，总的 10 位转换时间如下：

$$\text{A/D 转换频率} = 50 \text{ MHz} / (49 + 1) = 1 \text{ MHz}$$

$$\text{转换时间} = 1 / (1 \text{ MHz} / 5 \text{ 周期}) = 1 / 200 \text{ kHz} = 5 \mu\text{s}$$

ADC 可在最高为 2.5MHz 时钟下操作，因此转换率可高达 500kSPS。

### 2. 触摸屏接口方式

#### (1) 正常转换模式

单个转换模式，是最有可能用于通用的 ADC 转换。这种模式可以通过设置 ADCCON (ADC 的控制寄存器) 初始化，并完成读和写存入 ADCDAT0 (ADC 数据寄存器 0)。

#### (2) 单独的 X / Y 坐标转换模式

触摸屏控制器可以使用两个转换模式中的一个转换。单独的 X/Y 坐标转换模式可以在以下方法中转换：

X 坐标模式写 X 坐标的转换数据入 ADCDAT0，因此，触摸屏接口产生中断源到中断控制器。Y 坐标模式写 Y 坐标的转换数据到 ADCDAT1，因此，触摸屏接口生成中断源到中断控制器。

#### (3) 自动（顺序）的 X/Y 坐标转换模式

自动（顺序）的 X/Y 坐标转换模式，在以下方法转换：触摸屏控制器顺序转换 X 坐标和 Y 坐标被触摸。触摸屏写 X 测量数据如 ADCDAT0 和写 Y 测量数据入 ADCDAT1 后，触摸屏接口在自动位置转换模式上，产生中断源到中断控制器。

#### (4) 等待中断方式

当该系统在停止模式（电源中断）时，触摸屏控制器产生唤醒信号（WKU）。在触摸屏接口下，触摸屏控制器等待中断模式必须设置位置和方位状态 (XP, XM, YP, YM)。触摸屏控制器产生唤醒信号（Wake-Up）后，等待中断方式必须清除。（XY\_PST 没有操作模式的设置）

### 3. 待机模式

当 ADCCON[2] 被设定为 '1' 时，待机模式被激活。在此模式下，A / D 转换操作停止，并且 ADCDAT0, ADCDAT1 寄存器包含先前转换的数据。

### 4. 编程记录

(1) 该 A / D 转换的数据通过中断或轮询的方法被访问。中断方法—整个转换时间是从 ADC 开始到数据转换读取，因为中断服务程序的返回时间和数据存取时间，可能会有延时。轮询方法用来检查 ADCCON[15]，交换最后的特征位。该读取时间通过 ADCDAT 寄存器才能确定。

(2) 启动 A/D 转换的另一种方法。ADCCON[1]—A/D 转换的启动读取方式，设置为 1。A/D 转换开始时，同时转换成数据读取。

ADC 和触摸屏操作的信号如图 39-2 所示。

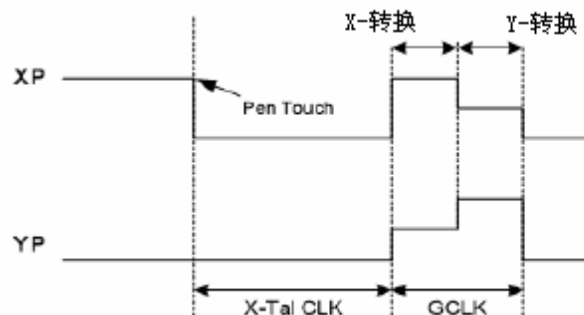


图 39-2 ADC 和触摸屏操作的信号

(3) 如果在 STOP 模式下，唤醒源被使用，XY\_PST 位 (ADCTSC[1:0]) 应设置为等待中断模式 (2' b11)。为了使触摸笔笔尖向上/向下移动有效，使用 UD\_SEN 位。

## 39.4 ADC 和触摸屏界面的特殊寄存器

### 39.4.1. ADC 的控制寄存器 (ADCCON)

寄存器	地址	读/写	描述	复位值
ADCCON	0x7E00B000	读/写	ADC 控制寄存器。	0x3FC4

ADCCON	位	描述	初始状态
ECFLG	[15]	转换的结束标记（只读）。 0= A/D 转换的过程中 1= A/D 转换结束	0
PRSCEN	[14]	ADC 预定标器启动。 0 =禁用 1 =启动	0
PRSCVL	[13:6]	ADC 预定标器值 0xFF。 数据值：5~255	0xFF

SEL_MUX	[5:3]	模拟输入通道选择。  000 = AIN 0 001 = AIN 1 010 = AIN 2 011 = AIN 3 100 = YM 101 = YP 110 = XM 111 = XP	0
STDBM	[2]	待机模式选择。  0 =正常运作模式 1 =待机模式	1
READ_START	[1]	A/D 转换开始读取。  0 =禁用开始读操作 1 =启动开始读操作	0
ENABLE_START	[0]	A/D 转换开始启用。  如果 READ_START 启用，这个值是无效的。 0 =无行动 1 =A/D 转换开始和该位被清理后开启	0

#### 39. 4. 2. ADC 的触摸屏控制寄存器 (ADCTSC)

寄存器	地址	读/写	描述	复位值
ADCTSC	0x7E00B004	读/写	ADC 的触摸屏控制寄存器。	0x58

ADCTSC	位	描述	初始状态
UD_SEN	[8]	检测触摸笔向上向下的位置。  0 =触摸笔向下中断信号	0



		1 =触摸笔向上中断信号	
YM_SEN	[7]	YM 开关启动。 0 =YM 输出驱动器禁用 1 =YM 输出驱动器启动	0
YP_SEN	[6]	YP 开关启动。 0 =YP 输出驱动器禁用 1 =YP 输出驱动器启动	1
XM_SEN	[5]	XM 开关启动。 0 =XM 输出驱动器禁用 1 =XM 输出驱动器启动	0
XP_SEN	[4]	XP 开关启动。 0 =XP 输出驱动器禁用 1 =XP 输出驱动器启动	1
PULL_UP	[3]	上拉开关启动。 0 =XP 上拉启用 1 =XP 上拉禁用	1
AUTO_PST	[2]	X 和 Y 的位置的自动定序转换。 0 =正常的 ADC 转换 1 =X 和 Y 的位置的自动定序测量	0
XY_PST	[1:0]	X 和 Y 坐标的手动测量。 00 =没有运作模式 01 = X 坐标测量 10 = Y 坐标测量 11 =等待中断方式	0

注：

- (1) 等待触摸屏中断，XP\_SEN 位必须设置为 ‘1’，即 ‘XP 输出禁用’ 和 PULL\_UP 位必须设置为 ‘0’，即 ‘XP 上拉启动’。
- (2) 在自动定序的 X/Y 坐标转换下 AUTO\_PST 应设置为 ‘1’。

39. 4. 3. 在 X/Y 坐标转换下触摸屏的 PIN 的条件

	XP	XM	YP	YM	ADC 通道选择
X 坐标	Vref	GUN	Hi-Z	Hi-Z	YP
Y 坐标	Hi-Z	Hi-Z	Vref	GND	XP

39. 4. 4. ADC 开始延迟寄存器（ADCDLY）

寄存器	地址	读/写	描述	复位值
ADCDLY	0x7E00B008	读/写	ADC 启动或时间延迟寄存器。	0x00ff

ADCDLY	位	描述	初始状态
FILCLKsrc	[16]	ADCDLY 时钟初始化。 0 =外部输入时钟 1 =RTC 时钟。	0
DELAY	[15:0]	（1）正常转换模式，模式的 XY 坐标，自动模式。→ ADC 转换启动延迟的值。 （2）等待中断方式。 在 STOP 模式下，触摸笔向下移动发生时，它产生唤醒信号，使间隔时间（数毫秒）用于额外的停止模式。 不要使用非零值（ 0x0000  ）。	00ff

注：在 ADC 的转换，触摸屏使用的 X\_tal 时钟（ 3.68MHz ）。在 ADC 转换使用 GCLK（最大 50 MHz ）。

39. 4. 5. ADC 的数据转换寄存器（ADCDAT0）

寄存器	地址	读/写	描述	复位值
ADCDAT0	0x7E00B00c	读	ADC 的数据转换寄存器。	-

ADCDAT0	位	描述	初始状态
UPDOWNM	[15]	在等待中断模式下，触摸笔向上或向下状态。 0 =触摸笔向下状态 1 =触摸笔向上状态	-
AUTO_PST	[14]	X 和 Y 坐标的自动定序转换。 0 =正常的 ADC 转换 1 =X 和 Y 坐标的定序测量	-
XY_PST	[13:12]	X 和 Y 坐标的手动测量。 00 =没有运作模式 01 =X 的坐标测量 10 =Y 的坐标测量 11 等待中断方式	-
Reserved	[11:10]	保留。	-
XPDATA (Normal ADC)	[9:0]	X 坐标的数据转换（包括正常的 ADC 的转换数据值）。 数据值： 0x0~0x3ff	-

39. 4. 6. ADC 的数据转换寄存器（ADCDAT1）

寄存器	地址	读/写	描述	复位值
ADCDAT1	0x7E00B010	读	ADC 的数据转换寄存器	-

ADCDAT1	位	描述	初始状态
---------	---	----	------

UPDOWNM	[15]	在等待中断模式下触摸笔向上或向下状态。 0 =触摸笔向下状态 1 =触摸笔不是向下状态	—
AUTO_PST	[14]	X 和 Y 坐标的自动定序转换。 0 =正常的 ADC 转换 1 =X 和 Y 坐标的定序测量	—
XY_PST	[13:12]	X 和 Y 坐标的手动测量。 00 =没有操作模式 01 =X 坐标测量 10 =Y 坐标测量 11 等待中断方式	—
Reserved	[11:10]	保留。	—
YPDATA (Normal ADC)	[9:0]	Y 坐标的数据转换。 数据值： 0x0 ~ 0x3ff	—

#### 39. 4. 7. ADC 的触摸屏 UP-DOWN 寄存器（ADCUPDN）

寄存器	地址	读/写	描述	复位值
ADCUPDN	0x7E00B014	读/写	触摸笔向上或向中断寄存器。	0x0

ADC DAT1	位	描述	初始状态
TSC_DN	[1]	触摸笔向下中断。 0 =无触摸笔向下状态 1 =有触摸笔向下状态	0
TSC_UP	[0]	触摸笔向上中断。 0 =无触摸笔向上状态 1 =有触摸笔向上状态	0

39. 4. 8. ADC 触摸屏中断清除寄存器

这些寄存器是用来清除中断的。当中断服务完成后，中断服务程序是负责清理中断的。在这个寄存器上写入任何数据都将清除相关有效的中断。当它被读取时，未定义的值将被返回。

寄存器	地址	读/写	描述	复位值
ADCCLRINT	0x7E00B018	写	清除 ADC 中断。	-

寄存器	地址	读/写	描述	复位值
ADCCLRWK	0x7E00B020	写	清除唤醒中断。	-

## 40 键盘接口

### 40.1 概述

S3C6410X 内的键盘接口模块使与外部键盘设备的通信变的便利。端口多路复用采用 GPIO 端口，提供 8 行 8 列。CPU 通过中断检测键盘按压和键盘释放事件。当行内发生任何中断时，软件用适当的程序浏览列行，检测一个或多个键盘按压及释放事件。

当键盘按压或释放或者两种情况都发生时，提供中断状态寄存器位。为了防止开关噪音，提供内部去抖过滤器。

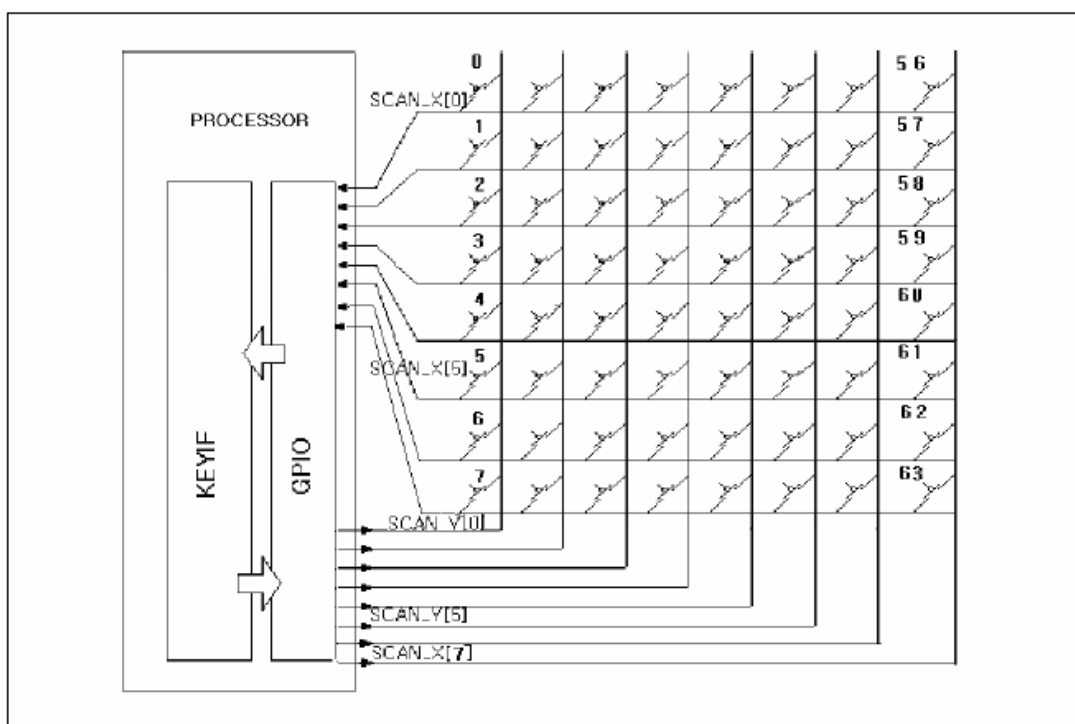


图 40-1 键盘矩阵接口外部链接向导

#### 1. 去抖过滤器

去抖过滤器支持任何键盘输入的键盘中断。过滤的宽度大约为 62.5usec。键盘中断是过

滤以后所有行输入的 ANDed 信号。

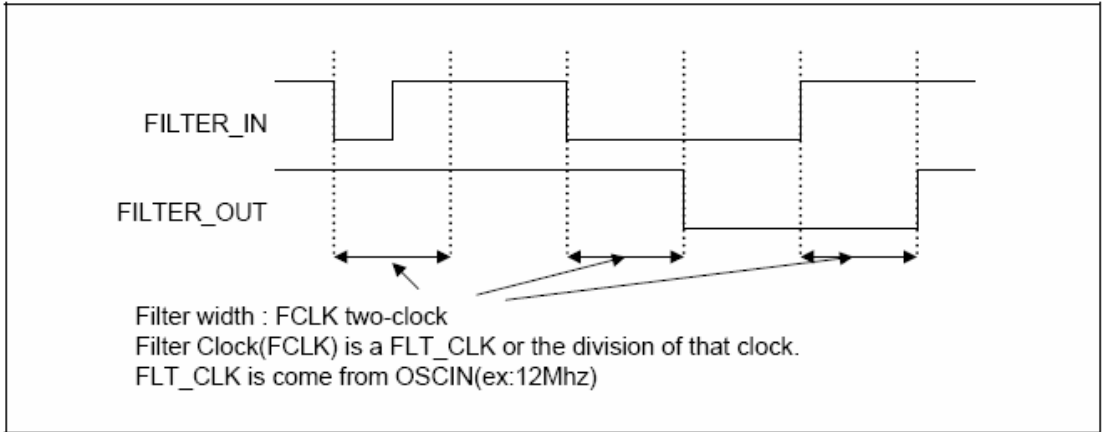


图 40-2 内部去抖过滤器操作

### 2.滤波时钟

键盘接口去抖滤波时钟是 OSC\_IN,由 FLT\_CLK 中划分出来。用户可以设置 10 位加计数器（KEYIFFC）的对比值。当滤波器使能位为高电平时，滤波器时钟分压器处于打开状态。FCLK 的频率是 FLT\_CLK 的功率/（（keyiffc+1）\*2）。相反的，FC\_CN 为低电平时，过滤时钟分压器不划分 FLT\_CLK.

## 40.2 管脚多路复用

在 S3C6410X 中，键盘接口输入、输出端口是 GPIO K 和 GPIO L 端口的多路复用。输入、输出端口计数由位控制，有 8 输入、8 输出。

表 40-1 键盘接口管脚多路复用

GPIO 端口 1	GPIO 端口 2	键盘接口端口	I/O
GPIOK[8]	GPION[0]	ROW_IN[0]	I
GPIOK[9]	GPION[1]	ROW_IN[1]	I
GPIOK[10]	GPION[2]	ROW_IN[2]	I
GPIOK[11]	GPION[3]	ROW_IN[3]	I
GPIOK[12]	GPION[4]	ROW_IN[4]	I
GPIOK[13]	GPION[5]	ROW_IN[5]	I

GPIOK[14]	GPION[6]	ROW_IN[6]	I
GPIOK[15]	GPION[7]	ROW_IN[7]	I
GPIOL[0]	GPIOH[0]	COL_OUT[0]	0
GPIOL[1]	GPIOH[1]	COL_OUT[1]	0
GPIOL[2]	GPIOH[2]	COL_OUT[2]	0
GPIOL[3]	GPIOH[3]	COL_OUT[3]	0
GPIOL[4]	GPIOH[4]	COL_OUT[4]	0
GPIOL[5]	GPIOH[5]	COL_OUT[5]	0
GPIOL[6]	GPIOH[6]	COL_OUT[6]	0
GPIOL[7]	GPIOH[7]	COL_OUT[7]	0

## 40.3 唤醒源

当键盘输入作为停止模式或睡眠模式的唤醒源时，不要求 **KEYPAD I/F** 寄存器的设置。唤醒需要 **KYEPAD I/F** 的 **GPIO** 寄存器设置和用于屏蔽的 **SYSCON** 寄存器(**PWR\_CFG**)。**GPIO**（键盘）端口 1 输入(**GPIOK[15:8]**)可以用来作为唤醒源，但是 **GPIO**(键盘)端口 2(**GPION[7]**)不能用来作为唤醒源。

## 40.4 软件键盘扫描程序

在初始状态，所有列（输出）都处于低电平。当没有键盘按压状态时，所有的行（输入）都是高电平（用于上拉）。当任何键被按压时，相应的行和列连接在一起，低电平划分到相应的行，将产生一个键盘中断。**CPU**（软件）向一行写入低电平，向其他列写入高电平。在每次写入的时候，**CPU** 读取 **KEYIFROW** 寄存器的值，并检测是否在相应的列内有键盘按压。当扫描程序结束时，可以发现被按压的键（一个或多个）。



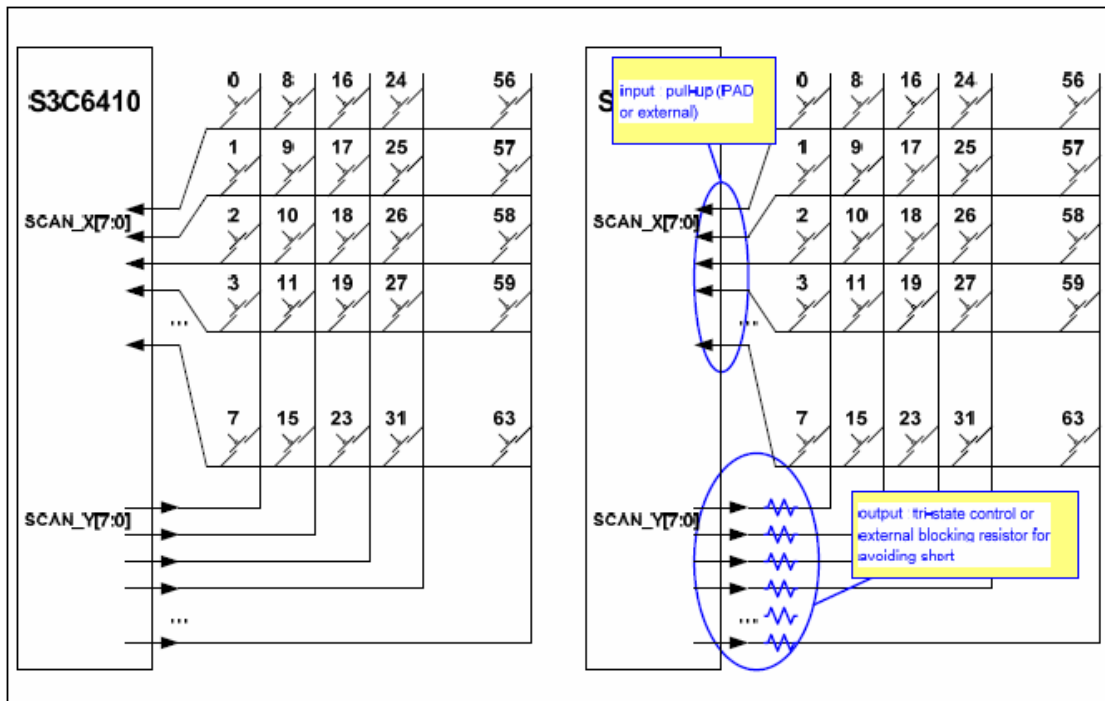


图 40-3 键盘扫描程序 1

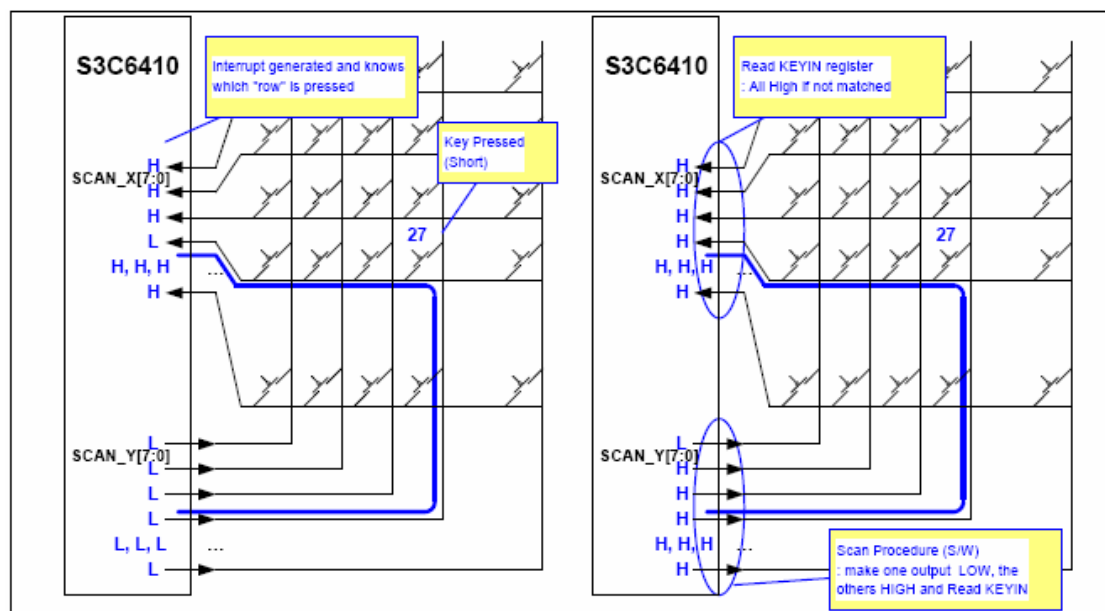


图 40-4 键盘扫描程序 2

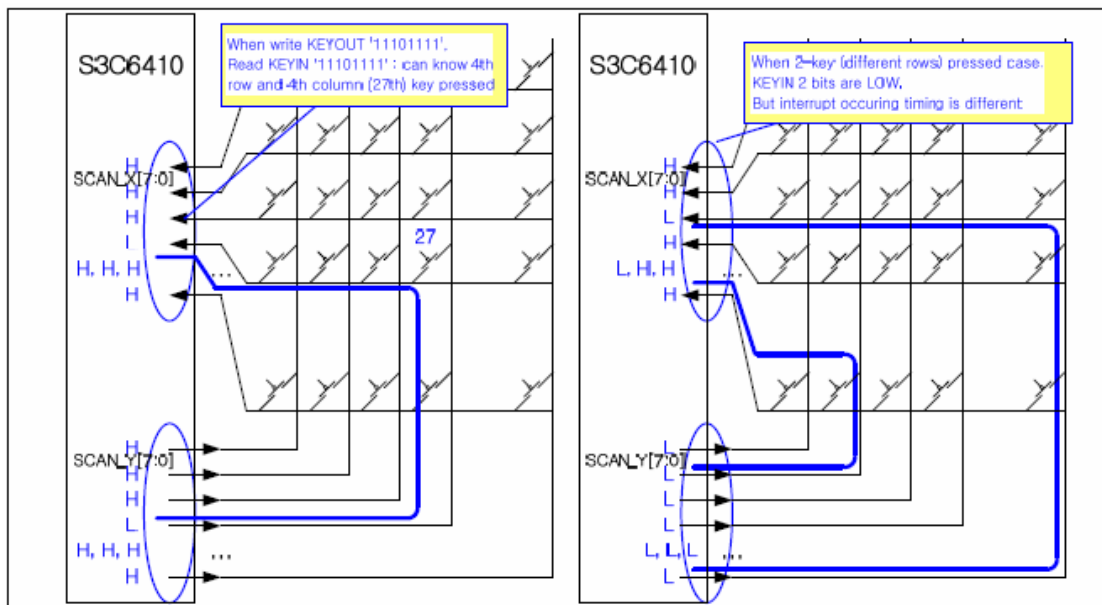


图 40-5 键盘扫描程序 3

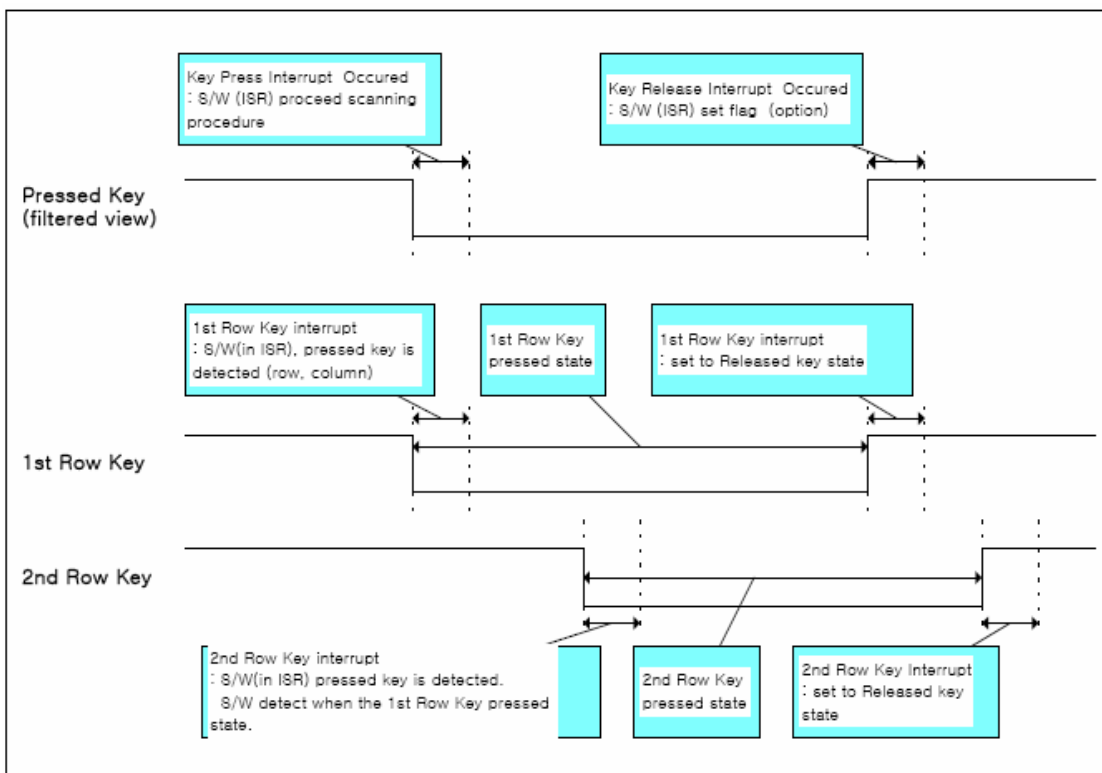


图 40-6 键盘扫描程序当两个键在不同的行上按压

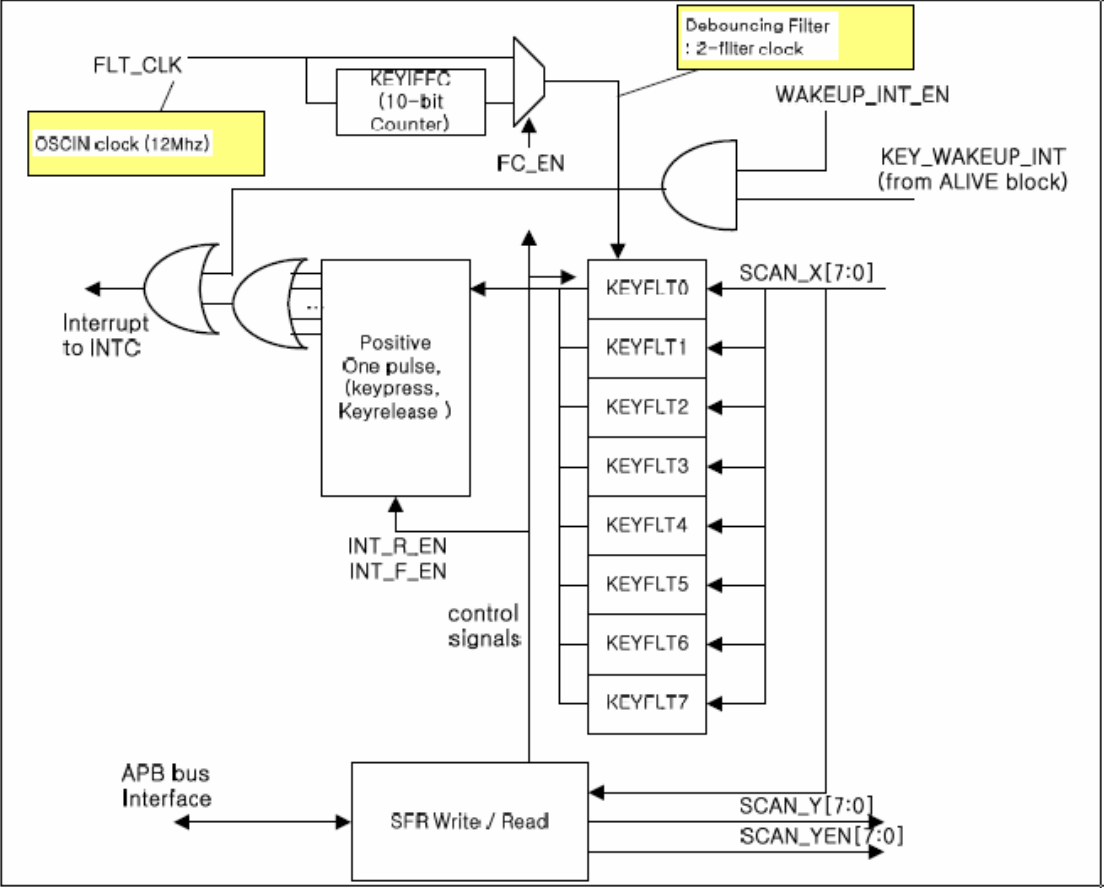


图 40-7 键盘 I/F 模块图

## 40.5 键盘接口寄存器

### 1. 存储器映射

寄存器	地址	读/写	描述	复位值
KEYIFCON	0x7E00A000	读/写	键盘接口控制寄存器	0x00000000
KEYIFSTCLR	0x7E00A004	读/写	键盘接口状态和清除寄存器	0x00000000
KEYIFCOL	0x7E00A008	读/写	键盘接口列数据输出寄存器	0x0000FF00
KEYIFROW	0x7E00A00C	读	键盘接口行数据输入寄存器	反映输入端口
KEYIFFFC	0x7E00A010	读/写	键盘接口去抖滤波时钟划分寄存器	0x00000000

# 40.6 寄存器描述

## 1.键盘接口控制寄存器（KEYIFCON）

寄存器	地址	读/写	描述	复位值
KEYIFCON	0x7E00A000	读/写	键盘接口控制寄存器	0x00000000

KEYIFCON	位	描述
Reserved	[31:5]	-
WAKEUP_INT_EN	[4]	键盘输入 停止/睡眠/闲置模式唤醒中断使能 0： 禁止 1： 键盘输入低电平（按压键盘时）唤醒中断使能
FC_CN	[3]	10 位计数器（用于去抖数字滤波时钟）使能 0： 禁止 1： 使能
DF_CN	[2]	键盘输入端口去抖滤波使能 0： 禁止 1： 使能
INT_R_CN	[1]	键盘输入端口上升沿中断 0： 禁止 1： 使能
INT_F_CN	[1]	键盘输入端口下降沿中断 0： 禁止 1： 使能

注：当 INT\_F\_IN 和 INT\_R\_IN 都被设置时可以选择上升沿和下降沿中断。

## 2.键盘中断状态和清除寄存器

寄存器	地址	读/写	描述	复位值
KEYIFSTCLR	0x7E00A004	读/写	键盘接口状态和清除寄存器	0x00000000

KEYIFSTSCLR	位	描述
P_IN	[7:0]	<p>键盘输入“按压”中断（下降沿）状态（读）和清除（写）</p> <p>读：1=发生按压的中断 0=不发生</p> <p>写：当写入数据 1 时，清除按压中断</p> <p>P_INT[7:0]表明 0 到 7 的每个按压键都有专用的中断。</p>
P_INT	[15:8]	<p>键盘输入“释放”中断（上升沿）状态（读）和清除（写）</p> <p>读：1=发生释放中断 0=不发生</p> <p>写：当写入数据 1 时，清除释放中断</p> <p>P_INT[15:8]表明 0 到 7 的每个释放键都有专用的中断，分别对应 R_INT[8]到 R_INT[15].</p>
Reserved	[31:16]	-

注：键盘唤醒中断当向 KEYIFSTSCLR 进行写访问时经常被清除。

### 3. 键盘中断列数据输出寄存器

寄存器	地址	读/写	描述	复位值
KEYIFCOL	0x7E00A008	读/写	键盘接口列数据输出寄存器	0x0000FF00

KEYIFCOL	位	描述
KEYIFCOL	[7:0]	键盘接口列数据输出寄存器
KEYIFCOLEN	[15:8]	<p>键盘接口列数据输出三态使能寄存器</p> <p>每个位对应着 KEYIFCOL 位</p> <p>0=输出板三状态缓冲区使能（常规输出）</p> <p>1=输出板三状态缓冲区禁止（高-Z 输出）</p>
Reserved	[31:16]	-

4. 键盘接口行数据输入寄存器

寄存器	地址	读/写	描述	复位值
KEYIFROW	0x7E00A00C	读	键盘接口行数据输入寄存器	反映输入端口

KEYIFROW	位	描述
KEYIFROW	[7:0]	键盘接口行数据输入寄存器
Reserved	[31:8]	-

5. 键盘接口去抖滤波时钟划分寄存器

寄存器	地址	读/写	描述	复位值
KEYIFFFC	0x7E00A010	读/写	键盘接口去抖滤波时钟划分寄存器	0x00000000

KEYIFFFC	位	描述
KEYIFFFC	[9:0]	键盘接口去抖滤波时钟划分寄存器 用于科研设置 10 位加计数相应的值。 寄存器值表示 FC_EC 位高电平的时间。 $FCLK \text{ 频率} = FLT\_CLK \text{ 频率} / ((KEYIFFFC[9:0] + 1) * 2)$ (FLT_CLK 是 OSC_IN(10~12MHZ))

# 41 IIS 多音频接口

## 41.1 概述

IIS(Inter-IC Sound)是一个流行的数字音频接口。总线只控制音频数据，其他信号如子编码和控制信号都分开转换。可以在两个 IIS 总线中传输数据。为了使要求的管脚数目最少，保持接线简单，用三线串连总线组成两个时间复用数据通道的线，一个是字选择线一个是时钟线。

IIS 接口从外部立体声音频编解码器上传输或接收声音数据。为了传输和接收数据，包括了两个 16×32 位的 FIFO 数据结构。可以支持 DMA 转换模式的传输或接收样本。

### 1.性能

ISS-BUS 接口包括以下性能：

- (1) DMA 基本操作的音频接口可以达到 5.1ch IIS-bus。
- (2) 每个通道可以转换 8/16/24 位数据
- (3) 支持 8kHz 到 192kHz 的速率。
- (4) 支持 IIS, MSB 校验和 LSB 校验数据格式
- (5) 64 字节 TX FIFO/64 字节 Rx FIFO 端口。

### 2. 信号描述

ISS 外部板由其他 IP 如 PCM，AC97 等等共用。IIS 为了使用这些外部板，必须在 IIS 开始之前设置 GPIO。参考本手册的 GPIO 章节可以得到更多的信息。

名称	类型	源/目的地	描述
XmmcDATA1[4]	输入/输出	Pad	IIS 多音频串行时钟
XmmcDATA1[5]	输出	Pad	IIS 多音频编解码系统时钟
XmmcDATA1[6]	输入/输出	Pad	IIS 多音频通道选择时钟
XmmcDATA [7]	输入	Pad	IIS 多音频串行数据输入
XspiMISO[1]	输出	Pad	IIS 多音频串行数据输出 0

XspiCLK[1]	输出	Pad	IIS 多音频串行数据输出 1
XspiCSn[1]	输出	Pad	IIS 多音频串行数据输出 2

3. 模块图

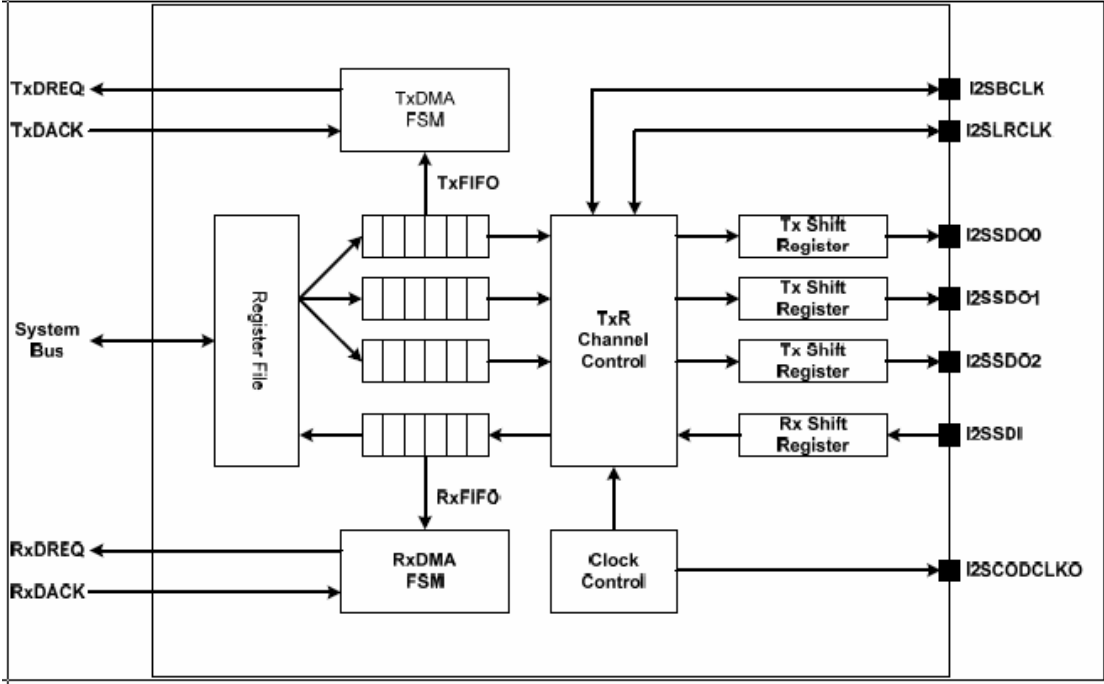


图 41-1 IIS 总线模块图

41.2 功能描述

IIS 接口包括寄存器区间、FIFO、移位寄存器、时钟控制、DMA 有限状态机以及如上图所示的通道控制模块。需要注意的是每个 FIFO 宽 32 位，深 16 位，包含了左右通道数据。因此 FIFO 访问和数据转换由左右成对单元处理。

1. 主/从模式

可以通过设置 IISMOD 寄存器的 IMS 位选择主模式或从模式。在主模式下，产生 I2SSCLK 和 I2SLRCLK，并且支持外部设备。因此在产生 I2SSCLK 和 I2SLRCLK 时需要 I2SCDCLK。使用 IIS 预先定标器从内部系统时钟内产生不同频率的 I2SCDCLK。在外部主模式下，I2SCDCLK 可以从 IIS 外部反馈回来。从模式下的管脚（GPIO）支持 I2SSCLK 和 I2SLRCLK。



主/从模式与 TX/RX 不同。主/从模式表示 I2SLRCLK 和 I2SSCLK 的方向。I2SCDCLK 的方向不重要。如果 IIS 总线接口向 IIS 编解码器传输时钟信号，IIS 总线处于主模式。如果 IIS 总线接口从 IIS 编解码器接收时钟信号，IIS 总线处于从模式。TX/RX 模式指明数据流的方向。如果 IIS 总线接口向 IIS 编解码器传输数据时，为 TX 模式。相反的，IIS 总线接口从 IIS 编解码器接收数据时为 RX 模式。需要将主/从模式和 TX/RX 模式区分开。

图 41-2 指出 I2SCDCLK 的路径，是设置在 IIS 时钟控制模块和系统控制器内的内部主模式或外部主模式。需要注意的是 RCLK 指出了路线时钟，这个时钟可以在外部主模式下向外部 IIS 编解码器芯片提供。

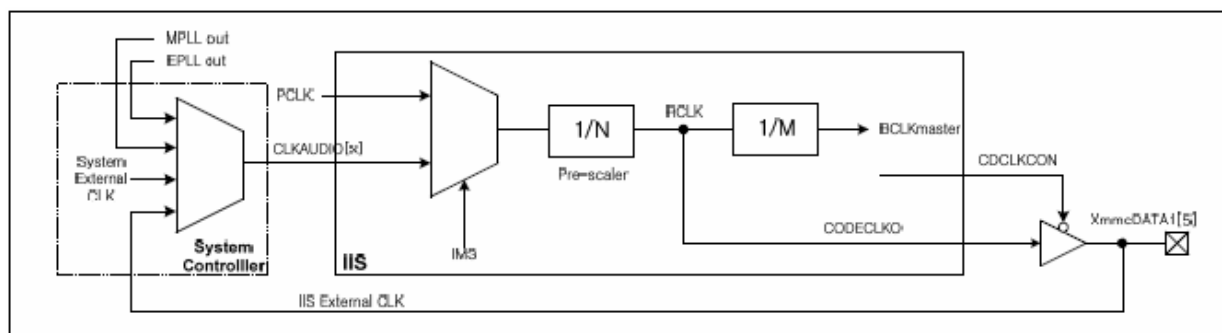


图 41-2 IIS 时钟控制模块图

## 2.DMA 转换

在 DMA 转换模式下，可以通过 DMA 控制器访问发射器或接收器 FIFO。通过传输器或接收器 FIFO 状态可以激活 DMA 服务请求。I2SCON 寄存器 FTXEMPT、FRXEMPT、FTXFULL 和 FRXFULL 位代表发射器或接收器 FIFO 数据状态。FTXEMPT 和 FRXFULL 位是 DMA 服务请求的准备标志；当 TXFIFO 非空时，传输 DMA 服务请求被激活，当 RXFIFO 未空时，接收机 DMA 服务请求被激活。

DMA 转换对单个数据只有交易方法。需要注意的是，在 DMA 承认激活期间，必须运行数据读或写操作。

DMA 请求点：

Tx 模式：(FIFO 未空) & (TXDMACTIVE 激活)

Rx 模式：(FIFO 非空) & (RXDMACTIVE 激活)

注意：在 DMA 模式下只支持单通道传输。

## 41.3 音频串行数据格式

### 1. IIS 总线格式

IIS 总线有四条线，包括串行数据输入 I2SSDI，串行数据输出 I2SSDO，左/右通道选择时钟 I2SLRCLK 和串行位时钟 I2SBCLK。产生 I2SRCLK 和 SI2SBCLK 的是主机。

串行数据在两个组成部分内传输，MSB 有固定的位置，根据字长度可决定 LSB 得位置。I2SLRCLK 变化以后，发射器在一个时钟期间发送最后字的 MSB。通过发射器发送的串行数据可以同时钟信号的尾沿和前沿同步。然而，串行信号必须在被锁在接收器内串行时钟信号的前沿上。因此，传输数据与前沿同步有一定得约束条件。

LR 通道选择行指明通道正在传输数据。在串行时钟的后沿或前沿可以改变 I2SLRCLK，但是 I2SLRCLK 不能调整到对称状态。在从模式下，这个信号被锁在时钟信号的前沿上。在传输 MSB 之前，I2SLRCLK 行改变时钟周期。这就允许子发射器得到传输信号的同步时序可以被设置用于传输。更进一步的说，使接收器储存先前的字，清除输入接收下一字的行为可行。

### 2. MSB(左) 校验

MSB 校验（左校验）格式与 IIS 总线格式相似，不同的是在 MSB 校验格式下，发射器经常发送下一字节的 MSB，同时改变 IWSLRCLK.

### 3.LSB(右)校验

LSB 校验（右校验）格式与 MSB 校验格式相反。在其他字内，正在传输的串行数据与 I2SLRCLK 转变的终点对齐。

图 41-3 显示出 IIS，MSB 校验，LSB 校验的音频格式。需要注意的是，在此图中，字的长度是 16 位的。

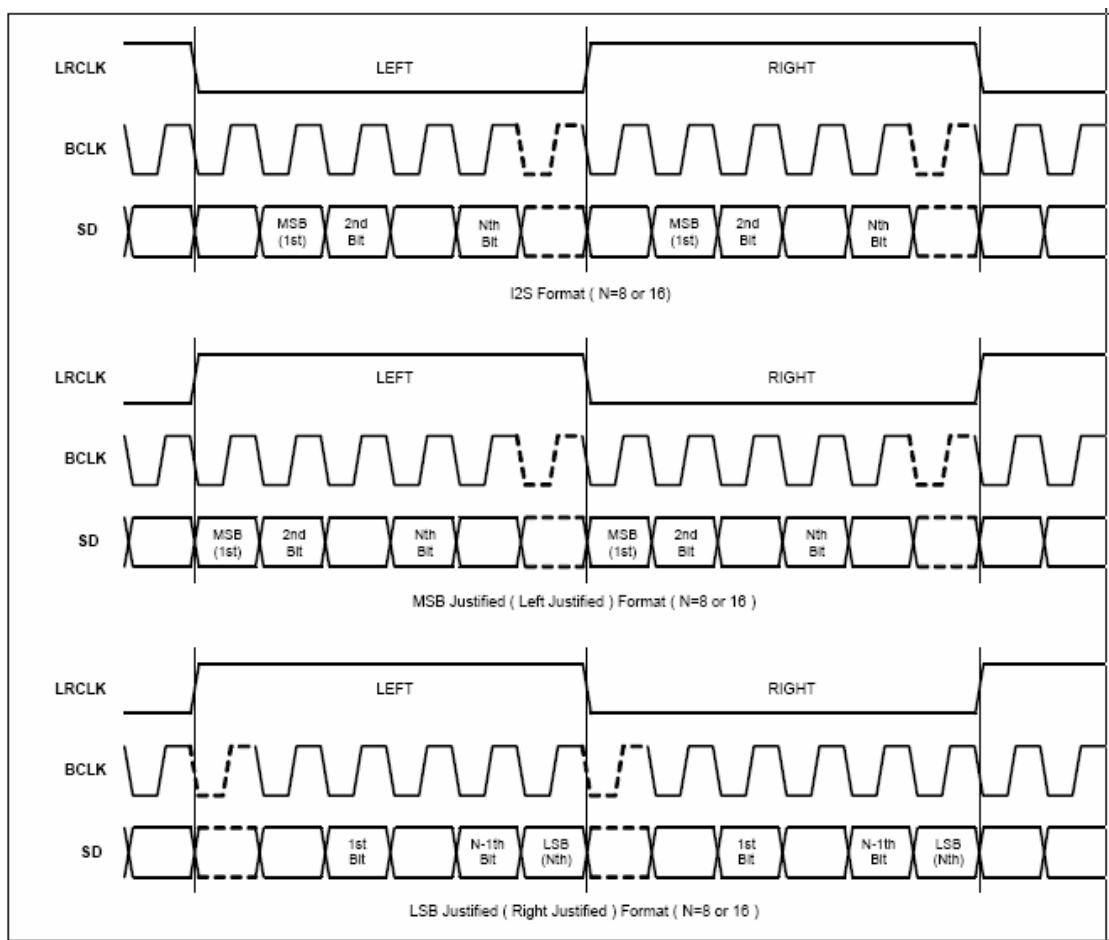


图 41-3 IIS 音频串行数据格式

## 41.4 采样频率和主时钟

可以通过如表 41-1 所示的采样频率选择主时钟频率 (RCLK)。因为 RCLK 是由 IIS 预先扫描产生的，因此预先扫描值和 RCLK 类型必须正确的定义。

表 41-1 编解码器时钟 (CODECLK=256fs, 384fs, 512fs, 768fs)

IISLRCK (fs)	8.000 kHz	11.025 kHz	16.000 kHz	22.050 kHz	32.000 kHz	44.100 kHz	48.000 kHz	64.000 kHz	88.200 kHz	96.000 kHz
	256fs									
	2.0480	2.8224	4.0960	5.6448	8.1920	11.2896	12.2880	16.3840	22.5972	24.5760

CODECLK (MHz)	384fs									
	3.0720	4.2336	6.1440	8.4672	12.2880	16.9344	18.4320	24.5760	33.8688	36.8640
	512fs									
	4.0960	5.6448	8.1920	11.2900	16.3840	22.5790	24.5760	32.7680	45.1580	49.1520
	768fs									
	6.1440	8.4672	12.2880	16.9340	24.5760	33.8690	36.8640	49.1520	-	-

注：fs 表示采样频率。CODEC 时钟是 fs（256、384、512、768）

## 41.5 IIS 时钟映射表

在 I2SMOD 寄存器的选择 BFS,RFS,和 BLC 位上，可以参考下面的表格。表 41-2 表示允许的时钟频率映射关系。

表 41-2 IIS 时钟映射表

时钟频率		RFS			
		256fs（00B）	512fs（01B）	384fs（10B）	768fs（11B）
BFS	16fs（10B）	(a)	(a)	(a)	(a)
	24fs（11B）	-	-	(a)	(a)
	32fs（00B）	(a) (b)	(a) (b)	(a) (b)	(a) (b)
	448fs（01B）	-	-	(a) (b) (c)	(a) (b) (c)
描述		(a)：当 BLC 为 8 位时允许 (b)：当 BLC 位 16 位时允许 (c) 当 BLC 位 24 位时允许			

注：位时钟频率≥fs\*（位长度\*2）。编解码时钟是位时钟的倍数。

## 41.6 编程指南

### 1. 初始化

(1) 在使用 IIS 总线接口之前,必须配置 GPIO 为 IIS 模式,并且检查信号的方向。I2SLRCLK,I2SSCLK 和 I2SCDCLK 是输入输出类型。I2SSDI 是输入,I2SSDO 是输出。

(2) 需要选择一个时钟源。S3C6410 有五个时钟源。这些时钟源是 MPLL, EPLL, PCLK, 系统外部时钟和 IIS 外部时钟。

### 2. 播放模式下的 DMA

(1) 在操作之前刷新 TXFIFO。如果没有将主/从模式和 TX/RX 模式区别好,必须先学习主/从模式和 TX/RX 模式。

(2) 正确的配置 I2SMOD 寄存器和 I2SPRS 寄存器。

(3) 为了稳定的运行系统,在传输之前 TXFIFO 应该几乎为满的。

(4) IIS 总线不支持中断,可以通过访问 SFR 调查监测 TXFIFO 状态。

(5) 如果 TXFIFO 满了,可以声明 I2SACTIVE.

### 3. 录音模式下的 DMA

(1) 在操作之前刷新 TXFIFO。如果没有将主/从模式和 TX/RX 模式区别好,必须先学习主/从模式和 TX/RX 模式。

(2) 正确的配置 I2SMOD 寄存器和 I2SPRS 寄存器。

(3) 为了稳定的运行系统,在 DMA 运行之前,内部 RXFIFO 最少有一个数据。

(4) 通过访问 SFR 调查监测 RXFIFO 状态。

(5) 如果 RXFIFO 非空,开始 RXDMACTIVE。

### 4.实例代码

#### (1) TX 通道

I2S TX 通道提供立体声兼容输出。传输通道可以在主模式或从模式下操作。在处理器和 I2S 控制器之间通过 APB 访问或 DMA 访问转换数据。

处理器必须写入多个字或两个字。关于音频 bitclk、BCLK 和字选择时钟、LRCLK 定时的字成串的定

时转换。

TX 通道有 16×32 位宽度的 FIFO，处理器或 DMA 可以在使能传输通道后向此区域写入高达 16 的 左/右数据采样。

(2) RX 通道

I2S RX 通道提供单个立体声兼容输出。接收通道可以操作在主模式或从模式下。数据从输入行接收，然后转换到 RX FIFO 内。处理器可以通过 APB 读操作读取此数据，或通过 DMA 访问此数据。

RX 通道有 16×32 位宽度的 RX FIFO，处理器或 DMA 可以在使能接收通道后向此区域写入高达 16 的 左/右数据采样。

41.7 IIS 总线接口特殊寄存器

表 41-3 IIS 接口寄存器摘要

寄存器	地址	读/写	描述	复位值
IISCON	0x7F00D000	读/写	IIS 接口控制寄存器	0xE00
IISMOD	0x7F00D004	读/写	IIS 接口模式寄存器	0x0
IISFIC	0x7F00D008	读/写	IIS 接口 FIFO 控制寄存器	0x0
IISPSR	0x7F00D00C	读/写	IIS 接口时钟划分控制寄存器	0x0
IISTXD	0x7F00D010	写	IIS 接口传输数据寄存器	0x0
IISRXD	0x7F00D014	读	IIS 接口接收数据寄存器	0x0

41.7.1.IISCON

寄存器	地址	读/写	描述	复位值
IISCON	0x7F00D000	读/写	IIS 接口控制寄存器	0xE00

IISCON	位	读/写	描述
Reserved	[31:18]	读/写	保留。运行到 0.

FTXURSTATUS	[17]	读/写	<p>TX FIFO 欠载运行中断状态。</p> <p>此位用来中断清除位。当此位为高电平时，可以通过写入 1 进行中断清除。</p> <p>0：中断不发生</p> <p>1：中断发生</p>
FTXURINTEN	[16]	读/写	<p>TX FIFO 欠载中断使能。</p> <p>0：TX FIFO 欠载中断禁止</p> <p>1：TX FIFO 欠载中断使能</p>
FTX2EMPTY	[15]	读	<p>TX FIFO2 空状态指明</p> <p>0：TX FIFO 为非空状态(准备传输数据)</p> <p>1：TX FIFO 为空状态（不准备传输数据）</p>
FTX1EMPTY	[14]	读	<p>TX FIFO1 空状态指明</p> <p>0：TX FIFO 为非空状态(准备传输数据)</p> <p>1：TX FIFO 为空状态（不准备传输数据）</p>
FTX2FULL	[13]	读	<p>TX FIFO2 满状态指明</p> <p>0：TX FIFO 未满</p> <p>1：TX FIFO 已满</p>
FTX1FULL	[12]	读	<p>TX FIFO1 满状态指明</p> <p>0：TX FIFO 未满</p> <p>1：TX FIFO 已满</p>
LRI	[11]	读	<p>左/右通道时钟指明。需要注意的是，LRI 的意思是根据 I2SMOD 寄存器的 LRP 为的值。</p> <p>0：左（当 LR 位为低电平）或右（当 LRP 位为高电平）</p> <p>1：右（当 LR 位为低电平）或左（当 LRP 位为高电平）</p>
FTX0EMPTY	[10]	读	<p>TX FIFO0 空状态指明</p> <p>0：FIFO 为非空状态(准备向通道传输数据)</p> <p>1：FIFO 为空状态（不准备向通道传输数据）</p>
FRXEMPTY	[9]	读	<p>RX FIFO 空状态指明</p> <p>0：FIFO 为非空状态</p>

			1: FIFO 为空状态
FTX0FULL	[8]	读	TX FIFO 满状态指明 0: TX FIFO 未满 1: TX FIFO 已满
FRXFULL	[7]	读	RX FIFO1 满状态指明 0: FIFO 未满(准备从通道接收数据) 1: FIFO 已满(未准备从通道接收数据)
TXDMA_PAUSE	[6]	读/写	TX DMA 操作暂停命令。需要注意的是当此位在任何时间被激活时，DMA 请求在当前运行的 DMA 转换完成以后被停止。 0: 不暂停 DMA 操作 1: 暂停 DMA 操作
RXDMA_PAUSE	[5]	读/写	RX DMA 操作暂停命令。需要注意的是当此位在任何时间被激活时，DMA 请求在当前运行的 DMA 转换完成以后被停止。 0: 不暂停 DMA 操作 1: 暂停 DMA 操作
TXCH_PAUSE	[4]	读/写	TX 通道操作暂停命令。需要注意的是当此位在任何时间被激活时，通道操作在当左-右通道数据转换完成以后被停止。 0: 不暂停操作 1: 暂停操作
RXCH_PAUSE	[3]	读/写	RX 通道操作暂停命令。需要注意的是当此位在任何时间被激活时，通道操作在当左-右通道数据转换完成以后被停止。 0: 不暂停操作 1: 暂停操作
TXDMACTIVE	[2]	读/写	TX DMA 激活(开始 DMA 请求)。需要注意的是当此位设置由高电平变为低电平时，DMA 操作被迫立即停止。 0: 未被激活 1: 被激活
RXDMACTIVE	[1]	读/写	RX DMA 激活(开始 DMA 请求)。需要注意的是当此位设置由高电平变为低电平时，DMA 操作被迫立即停止。



			0: 未被激活 1: 被激活
I2SACTIVE	[0]	读/写	IIS 接口激活（开始操作）  0: 未被激活 1: 被激活

### 41.7.2. IISMOD

寄存器	地址	读/写	描述	复位值
IISMOD	0x7F00D004	读/写	IIS 接口模式寄存器	0x0000_0000

IISMOD	位	读/写	描述
Reserved	[31:22]	读/写	保留。运行到 0.
CDD2	[21:20]	读/写	通道 2 数据丢弃。丢弃是指零填充。只支持 8/16 位模式。  00: 没有丢弃 01: I2STXD[15:0] 丢弃 10: I2STXD[31:16] 丢弃 11: 保留
CDD1	[19:18]	读/写	通道 1 数据丢弃。丢弃是指零填充。只支持 8/16 位模式。  00: 没有丢弃 01: I2STXD[15:0] 丢弃 10: I2STXD[31:16] 丢弃 11: 保留
DCE	[17:16]	读/写	数据通道使能  [17]:SD2 通道使能 [16]:SD1 通道使能
Reserved	[15]	读/写	保留。运行到 0.
BLC	[14:13]	读/写	位长度控制位，决定每个音频通道传输 8 位或 16 位。  00: 每个通道传输 16 位

			01: 每个通道传输 8 位 10: 每个通道传输 24 位 11: 保留
CDCLKCON	[12]	读/写	确定编解码器时钟源 0: 用内部编解码器时钟源 1: 从外部编解码器芯片内得到编解码器时钟源。
IMS	[11:10]	读/写	IIS 主模式或从模式选择 00: 主模式（用 PCLK） 01: 主模式（用 CLKAUDIO[X]） 10: 从模式（划分模式，用 PCLK） 11: 从模式（旁路模式，用 I2SCLK）
TXR	[9:8]	读/写	传输或接收模式选择 00: 传输模式 01: 接收模式 10: 传输和接收交替模式 11: 保留
LRP	[7]	读/写	左/右通道时钟极性选择 0: 左通道为低电平，右通道位高电平 1: 左通道为高电平，右通道为低电平
SDF	[6:5]	读/写	串行数据格式 00: IIS 格式 01: MSB 校验（左校验）格式 10: LSB 校验（右校验）格式 11: 保留
RFS	[4:3]	读/写	IIS 基础音时钟频率选择 00: 256fs, fs 是采样频率 01: 512fs 10: 384fs 11: 768fs

FFS	[2:1]	读/写	位时钟频率选择  00: 32fs, fs 是采样频率  01: 48fs  10: 16fs  11: 24fs
Reserved	[0]	读/写	保留。运行到 0.

### 41.7.3. IISFIC

寄存器	地址	读/写	描述	复位值
IISFIC	0x7F00D008	读/写	IIS 接口 FIFO 控制寄存器	0x0000_0000

IISFIC	位	读/写	描述
Reserved	[31:29]	读/写	保留。运行到 0.
FTX2CNT	[28:24]	读	TX FIFO2 数据计数。FIFO 的深度为 16, 所以值的变化范围是 0 到 15.  N: FIFO 的数据计数 N
Reserved	[23:21]	读/写	保留。运行到 0.
FTX1CNT	[20:16]	读	TX FIFO1 数据计数。FIFO 的深度为 16, 所以值的变化范围是 0 到 15.  N: FIFO 的数据计数 N
TFLUSH	[15]	读/写	TX FIFO 刷新命令  0: 不刷新    1: 刷新
Reserved	[14:13]	读/写	保留。运行到 0.
FTX0CNT	[12:8]	读	TX FIFO0 数据计数。FIFO 的深度为 16, 所以值的变化范围是 0 到 15.  N: FIFO 的数据计数 N
RFLUSH	[7]	读/写	RX FIFO 刷新命令  0: 不刷新    1: 刷新

Reserved	[6:5]	读/写	保留。运行到 0.
FTXCNT	[4:0]	读	RX FIFO 数据计数。FIFO 的深度为 16，所以值的变化范围是 0 到 15。 N: FIFO 的数据计数 N

## 41.7.4. IISPSR

寄存器	地址	读/写	描述	复位值
IISPSR	0x7F00D00C	读/写	IIS 接口时钟划分控制寄存器	0x0000_0000

IISPSR	位	读/写	描述
Reserved	[31:16]	读/写	保留。运行到 0.
PSRAEN	[15]	读/写	预先扫描激活 0: 未激活    1: 激活
Reserved	[14]	读/写	保留。SBZ
PAVALA	[13:8]	读/写	预先扫描划分值 N: 划分因子为 N+1
Reserved	[7:0]	读/写	保留。运行到 0.

## 41.7.5. IISTXD

寄存器	地址	读/写	描述	复位值
IISTXD	0x7F00D010	读/写	IIS 接口传输数据寄存器	0x0000_0000

IISTXD	位	读/写	描述
IISTXD	[31:0]	写	TX FIFO 写数据。注意：左/右通道数据分配为下面位区域： 当 16 位 BLC 时，R[31:16], L[15:0] 当 8 位 BLC 时，R[23:16], L[7:0]

41.7.6.IISRXD

寄存器	地址	读/写	描述	复位值
IISRXD	0x7F002014 0x7F003014	读/写	IIS 接口接收数据寄存器	0x0000_0000

IISRXD	位	读/写	描述
IISRXD	[31:0]	读	RX FIFO 读数据。注意：左/右通道数据分配为下面位区域： 当 16 位 BLC 时，R[31:16],L[15:0] 当 8 位 BLC 时，R[23:16],L[7:0]

## 42 3D 图形

### 42.1 概述

3D 图形是一个 3D 的图形的硬件加速器，可以加速 OpenGL ES 1.1&2.0 的描绘过程。3D 引擎主要定位于移动手机上，它的关键性能在下面有讲述。3D 引擎包括两个可编程着色器，一个是顶点着色器，另一个是像素着色器。在单描绘通道上最多可以支持 8 种属性。另外，因为 3D 引擎是采用 32 位浮点管道设计的，可以获得高质量的图形。等级纹理捕捉和纹理压缩用于低存储带宽环境。3D 引擎的主机接口使用一个 AHB 通道，帧缓冲区使用两个 AXI 通道。

#### 1.性能

- (1) 支持可编程着色器模型 3.0
- (2) 支持 128 位的浮点顶点着色器和几何纹理缓存。
- (3) 最大 4K×4K 帧缓冲区
- (4) 32 位深度缓冲区
- (5) 纹理格式：1/2/4/8/16/32 bpp RGB，YUV 422，S3TC 压缩
- (6) 最多可支持 8 个表面层
- (7) API 支持：OpenGL ES 1.1&2.0，D3D 移动
- (8) 智能主机接口，15 个输入数据类型，顶点缓冲区，顶点缓存
- (9) 8 个阶段，5 个线程着色器架构
- (10) 原始部件&有线硬件三角形设置引擎

模块图

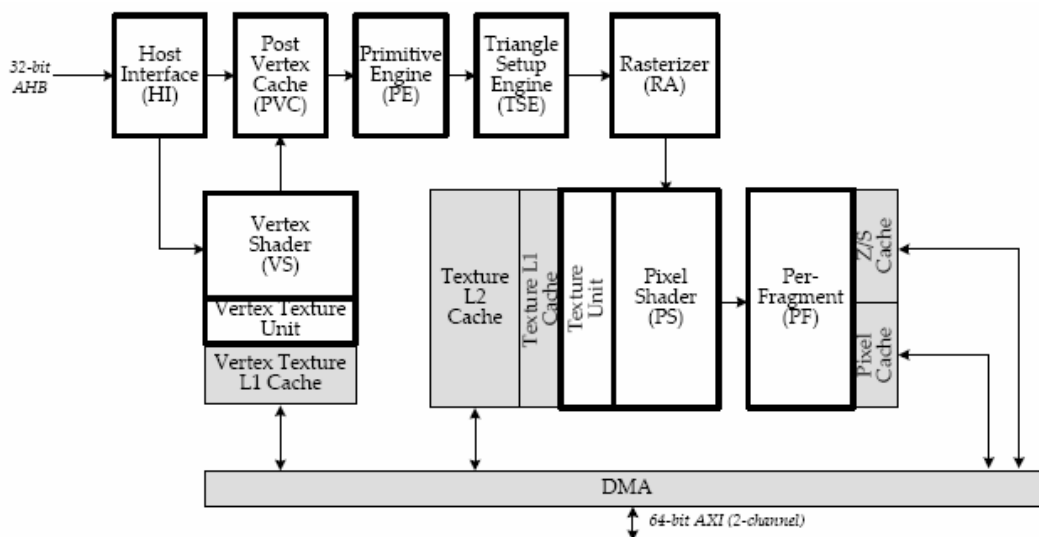


图 42-1 模块图

## 42.2. 全局寄存器

全局寄存器是 3D 图形中整体状态的设置。

### 42.2.1. FGGB\_PIPESTATE SFRS 的角色

FGGB\_PIPESTATE 中的每个位表示模块是否运行几何数据。如果位值是 1，表示几何数据在相应的模块运行。位值是 0 值表示相应的模块等待几何数据。FGGB\_PIPESTATE 用于确定每个模块状态更新的时间。例如，在 CPU 发送几何数据后，CPU 希望设置预片元单元的下一个状态。如果当先前几何数据在顶点着色器内，CPU 更新预片元新的状态，预片元单元新的状态将会影响剩下的数据，将会产生错误的结果。这种情况下，CPU 检测 FGGB\_PIPESTATE，确定处理几何数据的地点。在预片元单元之后，CPU 等待被转换的几何数据。只有当所有的时钟在预片元单元之前空闲时，预片元单元的状态可以被更新。

所有的几何数据都可以被处理和发送到帧缓冲区。此刻，CPU 可以安全的更新预片元单元的状态。如果 CPU 知道更新状态的正确时间，可以增加运行。这是 FGGB\_PIPESTATE 存在的原因。

### 42.2.2. 用 3D 图形中断的数据转换

数据转换包括 SFR 值的修改和几何数据的转换，中断可以用来改变 SFR 值和发送几何数据。

从 3D 图形管道状态发出的中断可用来知道何时改变 3D 图形的 SFR 值。只有当先前模块为空时，可以改变模块的 SFR 值。否则，3D 图形管道内遗留的几何数据可以被新的 SFR 值影响，并替代为试图应用于几何数据的先前的 SFR 值。CPU 可以重复读取管道状态，被称为投票站，可以知道更新 SFR 的时间。CPU 可以做另一项工作，替代调查管道状态和耗费的周期。这种情况下，CPU 可以设置中断条件。如果满足中断条件并发生中断时，CPU 可以改变 SFR 的值。

中断可以用来转换几何数据。当 3D 图形主机接口的主机 FIFO 内有空余空间时，CPU 转换几何数据。为了知道何时向 3D 图形主机接口的主机 FIFO 转换几何数据，CPU 可以一直检测管道状态。在一束几何数据发送之后，CPU 可以设置下一个几何数据的中断条件，并且运行另一个待定的工作。当发生一个中断时，CPU 可以像 3D 图形转换余下的几何数据。

图 42-2 说明如何转换几何数据。

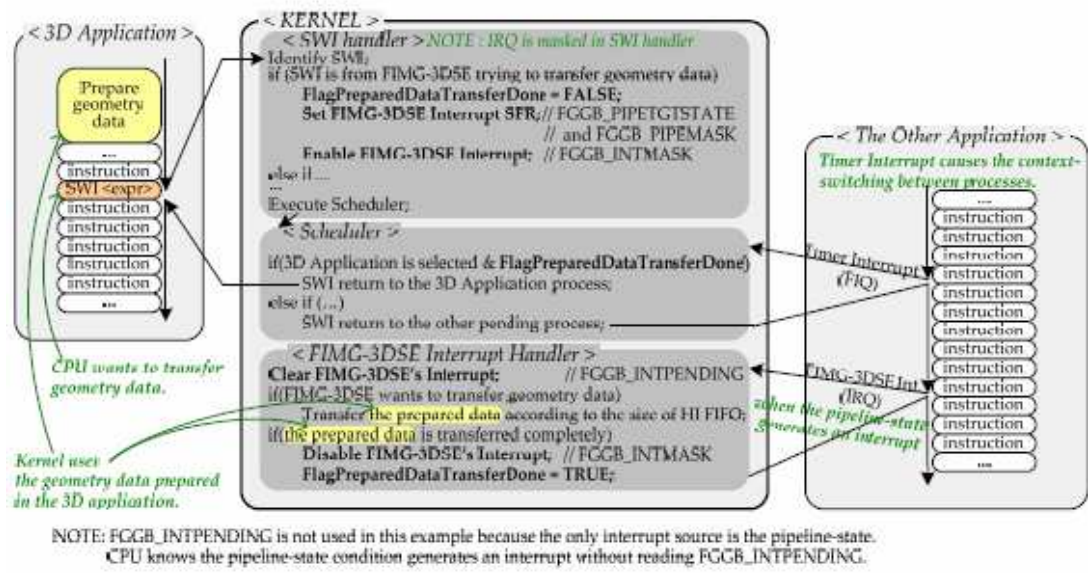


图 42-2 几何数据转换说明图

需要注意的是，当 3D 图形为空时，在 3D 图形中断使能后，管道状态产生一个中断。然而，当 CPU 执行其他用户模式应用程序时，将执行 3D 图形中断处理机。

图 42-3 说明如何改变 SFR 值。当所有的管道状态变为空时可以改变 SFR 值。



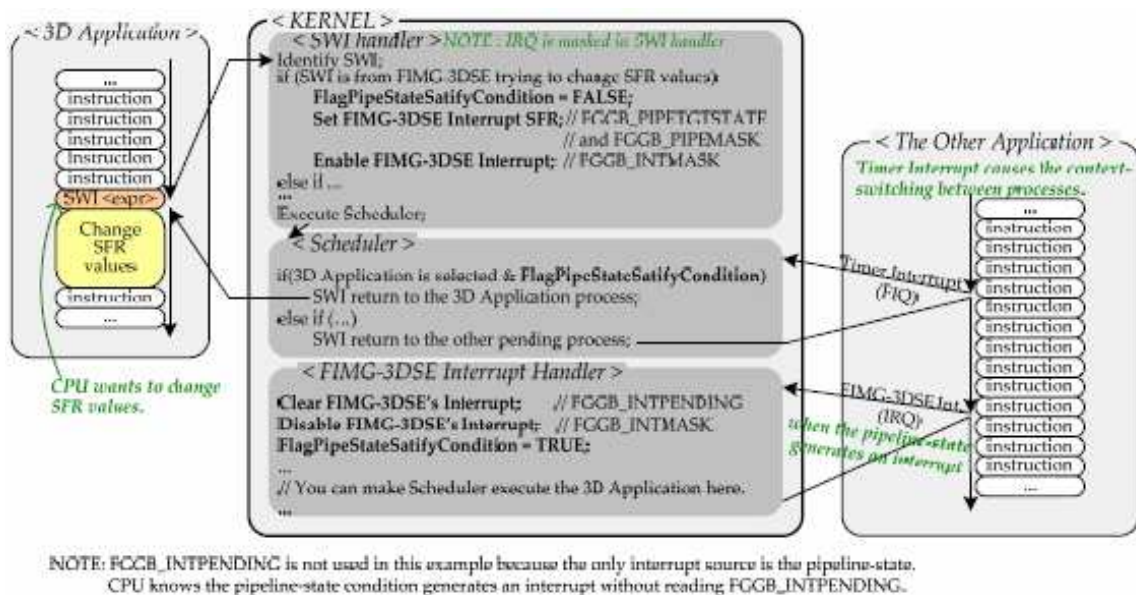


图 42-3 改变 SFR 值

注意上面的图表可以用来转换几何数据。

## 42.2.3. 全局特殊寄存器

### 42.2.3. 1. 管道状态寄存器（FGGB\_PIPESTATE）

寄存器	地址	读/写	描述	复位值
FGGB_PIPESTATE	0x72000000	读	管道状态	0x00000000

FGGB_PIPESTATE	位	描述	初始状态
Reserved	[31:19]	保留	0
CCache0	[18]	0b=颜色缓存 0 为空 1b=颜色缓存 0 为非空（繁忙）	0b
Reserved	[17]	保留	0

PF0	[16]	0b=预片元单元 0 为空 1b=预片元单元 0 为非空（繁忙）	0b
Reserved	[15:13]	保留	0
PS0	[12]	0b=像素着色器单元 0 为空 1b=像素着色器单元 0 为非空（繁忙）	0b
Reserved	[11]	保留	0
PA	[10]	0b=光栅化引擎为空 1b=光栅化引擎为非空（繁忙）	0b
TSE	[9]	0b=三角形设置引擎为空 1b=三角形设置引擎为非空（繁忙）	0b
PE	[8]	0b=图元引擎为空 1b=图元引擎为非空（繁忙）	0b
Reserved	[7:5]	保留	0
VS	[4]	0b=顶点着色器为空 1b=顶点着色器为非空（繁忙）	0b
VC	[3]	0b=顶点缓存为空 1b=顶点缓存为非空（繁忙）	0b
HVF	[2]	0b=主机接口和顶点着色器之间的 FIFO 为空 1b=主机接口和顶点着色器之间的 FIFO 为非空（繁忙）	0b
HI	[1]	0b=主机接口为空 1b=主机接口为非空（繁忙）	0b
HOST-FIFO	[0]	0b=主机接口的主机 FIFO 为空 1b=主机接口的主机 FIFO 为非空 （繁忙）	0b

42.2.3..2 缓存控制寄存器（FGGB\_CACHECTL）

如果设置 VTCCLEAR 为 1，一个周期以后 VTCCLEAR 自动变为 0.TCCLEAR 可以使纹理缓存 0 和纹理缓存 1 之间的连接无效。可以设置 TCCLEAR 为 01，10 或 11。一个周期以后，TCCLEAR 变为 00.FGGB\_CACHECTL 内的 CCFLUSH 和 ZCFLUSH 领域可以将缓存数据冲入原始和 Z 缓冲区。如果设置 CCFLUSH 为 11，当冲入操作完成以后，CCFULSH 自动变为 00。ZCFLUSH 的操作与 CCFULSH 的操作相同。

寄存器	地址	读/写	描述	复位值
FGGB_CACHECTL	0x72000004	读/写	缓存控制寄存器	0x00000000

FGGB_CACHECTL	位	描述	初始状态
Reserved	[31:13]	保留	0
VTCCLEAR	[12]	顶点纹理缓存清除(一个周期以后自动设置为 0b)  0：默认状态；顶点纹理缓存无效 操作不改变  1：顶点纹理缓存开始无效	0b
Reserved	[11:10]	保留	0
TCCLEAR	[9:8]	纹理缓存清除(一个周期以后自动设置为 0b)  00：默认状态；纹理缓存不改变 11：纹理缓存开始无效	00b
Reserved	[7:6]	保留	0
CCFLUSH	[5:4]	颜色缓存刷新（刷新之后，自动设置为 00b）  00：颜色缓存刷新结束 11：颜色缓存刷新开始	00b
Reserved	[3:2]	保留	0

ZCFLUSH	[1:0]	Z 缓存刷新（刷新之后，自动设置为 00b）  00b=Z 缓存 0 刷新结束  11b=Z 缓存 0 刷新开始	00b
---------	-------	--	-----

### 42.2.3 .3 软件复位寄存器（FGGB\_RST）

可以用 FGGB\_RST 寄存器复位 3D 图形的核心。然而，通过 FGGB\_RST 不影响 SFR 值。FGGB\_RST 的复位位不会自动覆盖为 0.可以设置 3D 图形操作的 FGGB\_RST 为 0.

寄存器	地址	读/写	描述	复位值
FGGB_RST	0x72000008	写	软件复位控制	0xbad1ff00

FGGB_RST	位	描述	初始状态
Reserved	[31:8]	保留	0
Reset	[0]	3D 图形核心的复位信号（内部逻辑存储器）  1=复位     0=工作	0b

### 42.2.3.4 版本信息寄存器（FGGB\_VERSION）

通过读 FGGB\_INFO 寄存器，可以识别系统内在执行那个 3D 图形。

寄存器	地址	读/写	描述	复位值
FGGB_VERSION	0x72000010	读	版本信息	0x01050000

FGGB_VERSION	位	描述	初始状态
Major	[31:24]	主要版本	0x01
Reset	[23:0]	次要版本  1=复位     0=工作	0x050000

42.2.3.5 中断悬挂寄存器（FGGB\_INTPENDING）

当 CPU 从 3D 图形内接收一个中断时，CPU 必须调查 3D 图形内的哪个功能块产生中断。CPU 可以通过读取 FGGB\_INTPENDING 找出产生中断的模块。

在从 3D 图形清除中断的服务惯例内，必须向 FGGB\_INTPENDING 写入任何值。通过向 FGGB\_INTPENDING 写入值，FGGB\_INTPENDING 将会自动清零，3D 图形可以产生另一个中断。向 FGGB\_INTPENDING 写入的数值是什么并不重要，凡是写入 FGGB\_INTPENDING 的写操作都会清除此值。

当前，FGGB\_PIPESTATE 在 HI 内可以产生中断。一旦 3D 图形产生一个中断，在不读取 FGGB\_INTPENDING 的情况下，CPU 会知道 FGGB\_PIPESTATE 是中断源。

寄存器	地址	读/写	描述	复位值
FGGB_INTPENDING	0x72000040	读/写	中断悬挂寄存器	0x00000000

FGGB_INTPENDING	位	描述	初始状态
Major	[31:1]	主要版本	0
Pipeline-State	[0]	读：产生管道状态中断 1= 发生中断， 0=没有中断 写：清除值到 0. 写入的数值并不重要	0b

42.2.3. 6 中断屏蔽寄存器（FGGB\_INTMASK）

FGGB\_INTMASK 可以使 3D 中断的信号使能或禁止。当前，只有通过 HI 才能产生中断。因此，FGGB\_INTMASK 的 LSB 用于使中断使能或禁止。

注：有另一种从管道状态禁止中断的方法。当 FGGB\_PIPEMASK 是位运算控制时，FGGB\_INTMASK 是全局控制。

寄存器	地址	读/写	描述	复位值
FGGB_INTMASK	0x72000044	读/写	使中断能活不能	0x00000000

FGGB_INTMASK	位	描述	初始状态
Reserved	[31:1]	保留	0
Pipeline-State	[0]	管道状态产生中断 1=使能中断 0=使中断不能	0b

### 42.2.3. 7 管道屏蔽寄存器（FGGB\_PIPEMASK）

FGGB\_PIPEMASK 指定产生中断的 3D 图形模块。

寄存器	地址	读/写	描述	复位值
FGGB_PIPEMASK	0x72000048	读/写	指定 3D 图形内用于产生中断的块。每个块的位位置与 FGGB_PIPESTATE 相同	0x00000000

FGGB_PIPEMASK	位	描述	初始状态
Reserved	[31:17]	保留	0
PF0	[16]	0b=不考虑 1b=用于产生中断	0b
Reserved	[15:13]	保留	0
PS0	[12]	0b=不考虑 1b=用于产生中断	0b
Reserved	[11]	保留	0
RA	[10]	0b=不考虑 1b=用于产生中断	0b
TSE	[9]	0b=不考虑 1b=用于产生中断	0b
PE	[8]	0b=不考虑 1b=用于产生中断	0b
Reserved	[7:5]	保留	0
VS	[4]	0b=不考虑 1b=用于产生中断	0b

VC	[3]	0b=不考虑 1b=用于产生中断	0b
HVF	[2]	0b=不考虑 1b=用于产生中断	0b
HI	[1]	0b=不考虑 1b=用于产生中断	0b
HOSTFIFO	[0]	0b=不考虑 1b=用于产生中断	0b

42.2.3. 8 管道目标状态寄存器（FGGB\_PIPETGTSTATE）

如前所述，FGGB\_PIPEMASK 定义产生中断的 3D 图形模块。FGGB\_PIPETGTSTATE 定义中断发生时的管道状态值。需要注意的是 FGGB\_PIPEMASK 内的 0 值模块的 FGGB\_PIEGTSTATE 值可忽略。

寄存器	地址	读/写	描述	复位值
FGGB_PIPETGTSTATE	0x7200004C	读/写	指定发生中断时的管道状态值	0x00000000

FGGB_PIPETGTSTATE	位	描述	初始状态
Reserved	[31:17]	保留	0
PF0	[16]	0b=当 PF0 不工作时中断（空） 1b=当 PF0 工作时中断（非空）	0b
Reserved	[15:13]	保留	0
PS0	[12]	0b=当 PS0 不工作时中断（空） 1b=当 PS0 工作时中断（非空）	0b
Reserved	[11]	保留	0
RA	[10]	0b=当 RA 不工作时中断（空） 1b=当 RA 工作时中断（非空）	0b
TSE	[9]	0b=当 TSE 不工作时中断（空） 1b=当 TSE 工作时中断（非空）	0b
PE	[8]	0b=当 PE 不工作时中断（空）	0b

		1b=当 PE 工作时中断（非空）	
Reserved	[7:5]	保留	0
VS	[4]	0b=当 VS 不工作时中断（空） 1b=当 VS 工作时中断（非空）	0b
VC	[3]	0b=当 VC 不工作时中断（空） 1b=当 VC 工作时中断（非空）	0b
HVF	[2]	0b=当 HI 和 VS 间的 FIFO 为空时中断 1b=当 HI 和 VS 间的 FIFO 为非空时中断	0b
HI	[1]	0b=当 HI 不工作时中断（空） 1b=当 HI 工作时中断（非空）	0b
HOSTFIFO	[0]	0b=当主机 FIFO 不工作时中断（空） 1b=当主机 FIFO 工作时中断（非空）	0b

### 42.2.3. 9 管道中断状态寄存器（FGGB\_PIPEINTSTATE）

当发生中断时，FGGB\_PIPEINTSTATE 捕捉管道状态。当同时发生几个中断时，FGGB\_PIPEINTSTATE 掌握第一个管道状态。

需要注意的是 FGGB\_PIPEINTSTATE 取决于 FGGB\_PIPEMASKE。

寄存器	地址	读/写	描述	复位值
FGGB_PIPEINTSTATE	0x72000050	读	发生几个中断时，掌握第一个管道状态	0x00000000

FGGB_PIPEINTSTATE	位	描述	初始状态
Reserved	[31:17]	保留	0
PF0	[16]	0b=发生中断时，PF0 为空 1b=发生中断时，PF0 非空	0b
Reserved	[15:13]	保留	0
PS0	[12]	0b=发生中断时，PS0 为空 1b=发生中断时，PS0 非空	0b
Reserved	[11]	保留	0



RA	[10]	0b=发生中断时，RA 为空 1b=发生中断时，RA 非空	0b
TSE	[9]	0b=发生中断时，TSE 为空 1b=发生中断时，TSE 非空	0b
PE	[8]	0b=发生中断时，PE 为空 1b=发生中断时，PE 非空	0b
Reserved	[7:5]	保留	0
VS	[4]	0b=发生中断时，VS 为空 1b=发生中断时，VS 非空	0b
VC	[3]	0b=发生中断时，VC 为空 1b=发生中断时，VC 非空	0b
HVF	[2]	0b=发生中断时， HI 和 VS 间的 FIFO 为空 1b=发生中断时， HI 和 VS 间的 FIFO 为非空	0b
HI	[1]	0b=发生中断时，HI 为空 1b=发生中断时，HI 非空	0b
HOSTFIFO	[0]	0b=发生中断时，主机 FIFO 为空 1b=发生中断时，主机 FIFO 非空	0b

## 42.3.主机接口

### 42.3.1.概述

主机接口单元的主要功能是接收 CPU 内转换为浮点数据格式的数据。主机接口可以向 CPU 转换状态数据（SFR 值， VS isnt-memory 等等）。

根据数据的特点，对从 CPU 向 3D 图形转换的数据进行分类，主要可分为状态数据和几何数据。CPU 首先设置状态数据，然后向 3D 图形转换几何数据。3D 图形用状态数据返回转换的几何数据。对于被返回

的下一个几何数据，相应的状态数据必须转换给 3D 图形。新的状态数据可以影响到先前的几何数据。因此，只有当状态数据不影响先前几何数据时才可以向 3D 图形转换状态数据。管道状态寄存器

(FGGB\_PIPESTATE) 在前几章内有介绍，用 FGGB\_PIPESTATE 询问 3D 图形内处理几何数据的地点。假设 3D 图形内有 A, B, C 和 D 四个模块，C 块的状态数据已被更新。如果 B 块处理几何数据，CPU 不更新 C 块的数据，因为 C 块新的状态可以影响到 B 块的几何数据。CPU 必须等到 B 块的几何数据穿过 C 块且到达 D 块时，CPU 才可以更新 C 块的状态，因为 A, B 和 C 块都为空。

如果帧缓冲区的内容应用于纹理，CPU 必须向纹理复制帧缓冲区的数据。下面部分描述了如何向主机接口提供几何数据。

### 42.3.2.操作模式

主机接口可以用两种方法决定如何转换几何数据：索引方法和非索引方法。

索引方法：CPU 向主机接口内的顶点缓冲区储存几何数据后，CPU 向储存的几何数据转换索引。这个方案消耗低总线的带宽。

非索引方法：CPU 直接转换几何数据。当顶点缓冲区内没有空间时这种方案非常有效。很少使用此种方法向主机接口转换几何数据。

### 42.3.3.索引模式

主机接口内索引模式使用顶点缓冲区。索引模式有两种操作方式：自动增加方式和索引转换方式。注意：所有的几何数据必须在顶点缓冲区内。索引范围与 VB 的尺寸有关：索引计算的 VB 地址必须是有效的 VB 地址。因此，应用程序中必须小心的控制索引。

自动增加模式 (FGHI\_CONTROL.EnVB=1,FGHI\_CONTROL.AutoInc=1)：CPU 发送两个 DWORD，一个表示计数和一个表示索引。主机接口使用顶点缓冲区内被转换的索引，然后下一个索引将自动计算；FGHI\_IDXOFFSET.VALUE 增加先前的索引数值(通常情况下设置为 0)，这个进程重复计数。每对 DWORD 表示索引的设置。因此，这种方法在转换几何数据中可有效的提高性能。自动增加模式如图 42-4 所示。



图 42-4 自动增加模式

索引转换模式 (FGHI\_CONTROL.EnVB=1,FGHI\_CONTROL.AutoInc=0): 在索引转换模式下, CPU 发送个人索引。在 CPU 发送索引计数的数目后, 将转换一个随机索引设置。这些索引用于索引顶点缓冲区。索引转换模式如图 42-5 所示。



图 42-5 索引转换模式

非索引模式 (FGHI\_CONTROL.EnVB=0,FGHI\_CONTROL.AutoInc=1): 在非索引模式下, 不使用顶点缓冲区。CPU 发送所有的几何数据。与索引模式一样, 必须首先转换顶点数和索引。在这种情况下, 索引使用模拟数值 (0xFFFFFFFF) .非索引模式如图 42-6 所示。

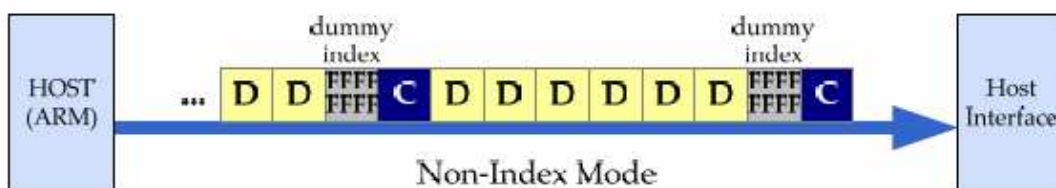


图 42-6 非索引模式

#### 42.3.4.如何设置主机接口的特殊寄存器

主机接口的特殊寄存器设置以后, 可以从 CPU 或顶点缓冲区内转换几何数据。几何数据是顶点着色器输入属性的一个设置。换句话说, 顶点的属性序号可以从 CPU 或从顶点缓冲区内转换。在

FGHI\_ATTRIB0~FGHI\_ATTRIB9 内定义一个顶点属性的构成设置。可以通过 Fghi\_Attribn.NumComp 决定每个属性组成部分的数量。FGHI\_ATTRIBn 内的最后属性位表示 Fghi\_Attribn 是否被使用。例如，FGHI\_ATTRIB0.LastAttr= Fghi\_Attrib1.LastAttr = Fghi\_Attrib2.LastAttr=0，FGHI\_ATTRIB3.LastAttr=1 的意思是顶点由四个属性组成。

只有 CPU 可以转换顶点的数量。（计数写入主机接口的 Fghi\_Dwentry）。CPU 可以向同一个 Fghi\_Dwentry 内转换索引或新的几何数据。顶点缓冲区只能向主机接口提供几何数据。顶点缓冲区的用途由 Fghi\_Control.EnVB 决定。如果使用了顶点缓冲区，将需要索引来索引查找顶点缓冲区内的几何数据。需要的索引可以通过 CPU 发送，或在主机接口内产生。在 CUP 的 DWORD 内有一些索引取决于 Fghi\_Control.IdxType(索引数据类型)。在下列途径中应用转换的或产生的索引。

```
1.  Get a count from CPU
2.  Get an index from CPU or previous index; // depending on Fghi_Control.AutoInc. In Non-Index mode, 0xFFFFFFFF is used.
3.  Add Fghi_IdxOffset.VALUE to the index for Index mode // In Non-Index mode, this step is skipped.
4.  for each n from 0x0 to 0xF
5.      if(Fghi_Control.EnVB == 1 && Fghi_Attrib[n].VBCTRL.Range != 0 && index < Fghi_Attrib[n].VBCTRL.Range)
6.          Use DWORDs fetched from
              VertexBuffer[Fghi_Attrib[n].VBBASE.Addr + index*Fghi_Attrib[n].VBCTRL.Stride]; // Index mode
7.      else
8.          Fetch DWORDs from CPU and use them as the geometry data; // Non-Index mode
9.          Transform DWORDs into floating point using Fghi_Attrib[n].Dt and send them to vertex shader.
10.     if (Fghi_Attrib[n].LastAttr == 1) break;
11.  end for
12.  repeat step 2~11 (count) times. // the (count) value in step 1 is used.
```

在第八步和第九步中，DWDORD 的数目由 Fghi\_Attribn.Dt 和 Fghi\_Attribn.NumComp 决定。在第八步，主机接口不能正确的认识 DWORD。主机接口从 CPU 只能得到 DWORD 的需要的数据。这就是 CPU 必须正确传输索引和几何数据的原因。

在第九步，Fghi\_Attrib[n].Dt 决定如何用浮点数格式传输转换的 DWORD。同时，在相同的步骤内，每个属性组成部分的顺序可以用每个 Fghi\_Attribn 内的 SrcX~SrcW 交换。SrcX~SrcW 的初始化为 0，与转换到 SrcX~SrcW 内的第一个值映射。如果（x，y，z，w）属性按照顺序被转换，SrcX~SrcW 必须设置值为 2'b11，2'b10，2'b01，2'b00。这个构造在颜色值转换为 BGRA 而不是 RGBA 时非常有用。如果每个组成部分的数目不是 4 个，那么 0.0，0.0 和 1.0 将自动附加上。例如，如果(x，y)数据被转换，那么（FLOAT(x),FLOAT(y)，0.0，1.0）被发送到顶点着色器。

需要注意的是，从 CPU 或顶点缓冲区转换的顶点属性的 DWORD 必须为 DWORD aligned。如果转换（8 位 x，8 位 y，8 位 z），将忽略下一个 8 位数据。

### 42.3.5. 如何从 CPU 向主机接口发送 DWORDS

在主机接口内有一个主机 FIFO。CPU 只能向主机接口转换 DWORD 数据。如果 CPU 写入的数据比主机 FIFO 内的数据多，并且能够正确的储存，主机接口使 HREADY，一个 AMBA 总线信号为低电平。这种情况下，AMBA 总线授予 3D 图形，在同一 AMBA 总线上的其他 IP 不能得到使用此总线的权利，这是不希望发生的情况。只读 FGHI\_DWSPACE 寄存器用于解决此种情况。FGHI\_DWSPACE 控制主机 FIFO 内的空闲的 DWORD 空间的数量。主机 FIFO 在主机接口内，并且可以储存 32 个 DWORD。无论 CPU 在什么时候发送计数，索引或几何数据，CPU 必须得到 FGHI\_DWSPACE 值，转换 DWORD 的数量于 FGHI\_DWSPACE 的值一样多。

一般而言，FGHI\_DWSPACE 不告诉 CPU 准确的空闲空间，因为提供给 3D 图形和 AMBA 总线的时钟信号可以不同。（如果时钟信号相同，FGHI\_DWSPACE 有准确的值）。因此，FGHI\_DWSPACE 受 AMBA 总线和内部情况的影响。

如果读取的 FGL\_DWSPACE 值比主机 FIFO 内的实际自由空间小，写操作将会完成，并且不会出现任何问题。另一方面，如果读取的 FGHI\_DWSPACE 值比写入的 DWORD 的数量多，主机接口使 HREADY 信号变为低电平，并扩展转换。然而，读取的 FGHI\_DWSPACE 值于实际值之间的不同通常很小。

CPU 读取 FGHI\_DWSPACE 和转换 DWORD 与读取值一样多以后，CPU 可以做其他工作和处理，或者继续发送几何数据的其他部分，重复相同的处理。CPU 可以使用 3D 图形的中断方案。

当转换几何数据的时候，中断和顶点缓冲是非常有用的方案。

### 42.3.6. CPU 的索引转换类型

FGHI\_CONTROL 内的 IdxType 控制着 CPU 的 DWORD 内存在的索引的数量。如果 IdxType 为无符号的整数类型，在转换的 DWORD 内只有 32 位的索引。如果为无符号的短型，在一个 DWORD 内有两个 16 位的索引。如果为无符号的字节类型，将有 4 个八位的索引。

当所有索引被使用时，DWORD 内剩下的索引被忽略。例如，如果转换了三个无符号字节类型索引的顶点，将用到一个 DWORD 数据。这种情况下，忽略 DWORD 内剩下的无符号字节。

### 42.3.7. 顶点缓冲区的数据转换

在使用顶点缓冲区的内容之前，几何数据必须在顶点缓冲区内。首先，将顶点缓冲区内 16 字节对齐的目标地址设置为 FGHI\_VBADDR。然后，写入 FGHI\_VBDATA 内的一串 DWORD 被储存到顶点缓冲区内。当 4 个 DWORD 写入 FGHI\_VBDATA 内时，FGHI\_VBADDR 的地址自动增加 16。因此，每向 FGHI\_VBDATA 写入 DWORD 时，不需要更新目标地址。需要注意的是，写入 FGHI\_VBDATA 的 DWORD 数量必须是 4 的倍数。只有当从 CPU 内转换 4 个 DWORD 时，这 4 个 DWORD 储存在顶点缓冲区内。

如果几何数据的尺寸不是 4 的倍数，那么向顶点缓冲区发送增加的 DWORD。DWORD-aligned 增加的 DWORD 不影响 FGHI\_ATTRIBn\_VBCTRL.Range 的值。实际的索引范围值必须写入 FGHI\_ATTRIBn\_VBCTRL.Range。

注意：NAN 或无限浮点数或半浮点值必须写入顶点缓冲区，像非索引模式一样。

同时注意顶点缓冲区内几何数据必须是 DWORD-aligned。

### 42.3.8. 如何使用顶点缓冲区作为一个时间缓冲使用中断

主机 FIFO 内有 32 个 DWORD 空间。如果 CPU 发送大量的 FGHI\_DWORDS 轮流检测的 DWORD，CPU 将耗费大量的周期读取 FGHI\_DWSPACE，并没有做任何其他有用的工作，直到所有的 DWORD 转换完毕。这是不希望发生的情况。

顶点缓冲区和中断方案可以用在此种情况下。这种情况下的顶点缓冲区的一次几何应用：顶点缓冲区经常储存几何数据，假设进程中多次使用数据。

CPU 将几何数据的 DWORD 的一部分发送到顶点缓冲区内，而不是发送到主机 FIFO 内。保存 DWORD 到顶点缓冲区内后，当主机 FIFO 的 FGGB\_POPESTATE 值和主机接口值变为 0 的时候，CPU 设置 3D 图形中断使中断单元向 CPU 发送中断。此时，CPU 可以做其他有用的工作，如运行系统或探测相关处理，等待 3D 图形的中断。如果 3D 图形发生中断，将允许 CPU 处理几何发送进程，CPU 用相同的步骤继续发送几何数据的剩下的部分。

当所有 3D 图形管道状态变为空时可以发生中断。可以自己决定什么时候发生中断。这里有一个需要注意的事情：必须发送准确的顶点的数量。例如，当图元引擎，就是顶点缓存的下一个模块，被假设得到

一个三角数据，顶点的数必须是 3 的倍数。如果  $(3n+1)$  订单被发送，中断单元会等待 FGHI\_PIPESTATE 变为 0，在这种情况下，3D 图形的不会发生中断，因为 FGHI\_PIPESTATE 的图元引擎的值是 1，将等待另两个顶点。

如果用这种方式使用顶点缓冲区，建议不使用顶点缓存。因为所有的 DWORDS 只用一次，顶点缓存没有 Hit-Case。

### 42.3.9.顶点缓存控制

FGHI\_CONTROL 内的 EnVE 和 NumOutAttrib 领域控制着顶点缓存工作的方法。如果 EnVC 区域为 0，那么顶点缓存将不可用。NunOutAttrib 区域储存着顶点着色器输出属性的数量。由 NunOutAttrib 决定的输出属性的数量被转换到图元引擎内。

注意当通过 CPU 写入 FGHI\_CONTROL 时，后顶点缓存将自动清零(初始化)。当发送一个几何数据的一串索引时，将发送另一个不同的几何数据。这种情况下，先前几何数据的索引保存在顶点缓存内，当新的几何数据索引发送时可以被隐藏。因此，当用索引模式发送多个几何数据时，必须清除几何数据之间的顶点缓存。当通过 CPU 写入 FGHI\_CONTROL 时，顶点缓存自动清除。尽管 FGHI\_CONTROL 值不改变，FGHI\_CONTROL 可以用相同的值写入来清除顶点缓存的内容。

### 42.3.10.主机接口特殊寄存器

#### 42.3.10.1 主机接口的自由 DWORD 空间寄存器（FGHI\_DWSPACE）

寄存器	地址	读/写	描述	复位值
FGHI_DWSPACE	0x72008000	读	HI 内主机 FIFO 的空插槽数量	0x00000000

FES	位	描述	初始状态
VAL	[31:0]	空插槽数量	0x0

42.3.10.2 主机 FIFO 进入端口寄存器（FGHI\_DWENTRY）

寄存器	地址	读/写	描述	复位值
FGHI_DWENTRY	0x7200C000 ~ 0x7200DFFF	写	HI 内主机 FIFO 的输入端口。  可以突发写入。  写入 0x0000C000 ~ 0x000DFF 的 DWORD 被储存在主机接口内的主机 FIFO 内。	-

FGHI_DWENTRY	位	描述	初始状态
DATA	[31:0]	顶点, 索引的数目和几何数据被转换到此寄存器内	-

42.3.10.3 主机接口控制寄存器（FGHI\_CONTROL）

寄存器	地址	读/写	描述	复位值
FGHI_CONTROL	0x72008008	读/写	主机接口控制寄存器。注意：当 FGHI_CONGROL 被写入时，VC 自动初始化	0x00010000

FGHI_CONTROL	位	描述	初始状态
EnVB	[31]	使能顶点缓冲区	0b
Reserved	[30:26]	保留	0
IdxType	[25:24]	转换的索引类型  00b=无符号整数  01b=无符号短整数  10b=保留	00b



		11b=无符号字节	
Reserved	[23:17]	保留	0
AutoInc	[16]	自动增加模式	1b
Reserved	[15:5]	保留	0
EnVC	[4]	使能顶点缓存	0b
NumOutAttrib	[3:0]	顶点着色器输出属性的数量 当 point-sprite 被使用时，这个数必须是（着色器输出的数目+1）。	0000b

42.3.10.4 索引 补偿寄存器（FGHI\_IDXOFFSET）

寄存器	地址	读/写	描述	复位值
FGHI_IDXOFFSET	0x7200800C	读/写	索引补偿寄存器	0x00000001

FGHI_IDXOFFSET	位	描述	初始状态
VAL	[31:0]	索引补偿值  当索引自动增加时，VAS 增加索引。CPU 内第一个被转换的索引被应用。  因此，使用的索引是：索引，索引+VAL，索引+2*VAL 等等  当 CPU 转换的索引被使用，VAL 正道每一个转换的索引。如，索引 0，索引 1，索引 2 等从 CPU 发送，那么在 HI 内使用规定索引是：所以 0+VAL，索引 1+VAL, 索引 2+VAL 等等。	0x00000001

42.3.10.5 顶点缓冲区地址寄存器（FGHI\_VBADDR）

寄存器	地址	读/写	描述	复位值
FGHI_VBADDR	0x72008010	读/写	写：设置目标地址寄存器  读：地址被用来读取下一个向 VB 转换的几何数据。这个值可以用来计算多少个数据被转换。  当向顶点缓冲区写入四个 DWORD 时，FGHI_VBADDR 自动更新。	0x00000000

FES	位	描述	初始状态
VAL	[31:0]	复制几何数据属性的开始地址	0x0

42.3.10.6 顶点缓冲区进入端口地址（FGHI\_VBDATA）

寄存器	地址	读/写	描述	复位值
FGHI_VBDATA	0x7200E000 ~ 0x7200FFFF	读/写	写：用于向 VB 内写几何数据 可以突发写入。  写入 0x0000E000~0x0000FFFF 内的 DWORD 储存在顶点缓冲区内  读：读取最后写入 FGHI_VBDATA 内的数据	0x00000000

VBD	位	描述	初始状态
DATA	[31:0]	顶点缓存区的数据输入端口。这个寄存器的写入数据位 4 的倍数。开始地址自动增加。	0x0

42.3.10.7 属性控制寄存器（FGHI\_ATTRIB0~FGHI\_ATTRIB9）

如果顶点数据类型为字节型，无符号字节型，归一字节，或归一无符号字节型，CPU 向主机接口转换的 DWORD 必须包括四个组成部分。如图 42-7 所示的例子 DWORD，未使用的 8 位数据被忽略了。



图 42-7 DWORD 结构

在上面的例子中，（8 位 x，8 位 y，8 位 w）属性可用一个 DWORD 转换。

如果顶点数据类型为短整型，无符号短整型，归一短整型，或归一无符号短整型，一个 DWORD 内有两个属性。因此(16 位 x，16 位 y，16 位 z，16 位 w)需要两个 DWORD，如图 42-8 所示。

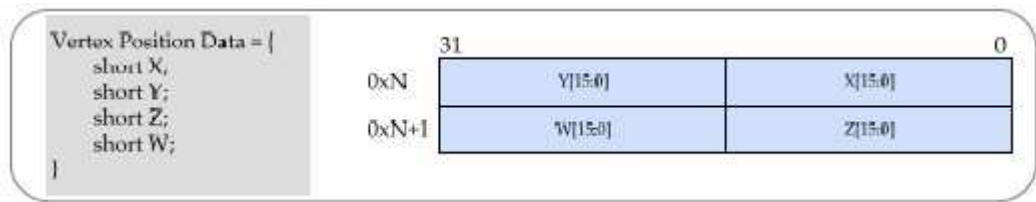


图 42-8 DWORD 结构

注意，当 **DWORD** 储存在顶点缓冲区内时，上面的规则同样应用。

寄存器	地址	读/写	描述	复位值
FGHI_ATTRIB0	0x72008040	读/写	输入属性 0 控制寄存器	0x000000E4
FGHI_ATTRIB1	0x72008044	读/写	输入属性 1 控制寄存器	0x000000E4
FGHI_ATTRIB2	0x72008048	读/写	输入属性 2 控制寄存器	0x000000E4
FGHI_ATTRIB3	0x7200804C	读/写	输入属性 3 控制寄存器	0x000000E4
FGHI_ATTRIB4	0x72008050	读/写	输入属性 4 控制寄存器	0x000000E4
FGHI_ATTRIB5	0x72008054	读/写	输入属性 5 控制寄存器	0x000000E4
FGHI_ATTRIB6	0x72008058	读/写	输入属性 6 控制寄存器	0x000000E4
FGHI_ATTRIB7	0x7200805C	读/写	输入属性 7 控制寄存器	0x000000E4
FGHI_ATTRIB8	0x72008060	读/写	输入属性 8 控制寄存器	0x000000E4
FGHI_ATTRIB9	0x72008064	读/写	输入属性 9 控制寄存器	0x000000E4

FGHI_ATTRIBn	位	描述	初始状态
LastAttr	[31]	0b=指明 ATTRIBn 被使用 1b=指明 ATTRIBn 是最后的属性 设置以后，所有的 last 值为 0，意思是默认使用一个属性。 如： Hghi_Attrib0[31]=0, Hghi_Attrib1[31]=0 Hghi_Attrib2[31]=1 Hghi_Attrib3~9[31]=不考虑 Hghi_Attrib0~2 被使用	1b
Reserved	[30:16]	保留	0
Dt	[15:12]	属性 n 的每个组成部分被转换为 位 数据类型 范围 0000 字节 -127~128	0

		0001            端整型                    -23768~32767 0010            整型                        -2147483648~2147483647 0011            固定的                                -32768~32768 0100            无符号字节                        0~255 0101            无符号短整型                        0~65535 0110            无符号整型                        0~4294967295 1000            浮点型                        IEEE 754 单精度 1001            归一字节                        -1.0f~1.0f 1010            归一短整型                        -1.0f~1.0f 1011            归一固定                        0.0f~1.0f 1100            归一无符号字节                        0.0f~1.0f 1101            归一无符号短整型                        0.0f~1.0f 1110            归一无符号整型                        0.0f~1.0f 1111            半浮点                        s/5/10 格式 当浮点或半浮点数据类型使用时，必须不能 NaN 或无穷数使用	
Reserved	[11:10]	保留	0
NunComp	[9:8]	组成部分的数量 00b=只有一个组成部分被转换 (a, b, c, d) = (1 <sup>st</sup> , 0, 0, 1) 01b=两个组成部分被转换 (a, b, c, d) = (1 <sup>st</sup> , 2 <sup>nd</sup> , 0, 1) 10b=三个组成部分被转换 (a, b, c, d) = (1 <sup>st</sup> , 2 <sup>nd</sup> , 3 <sup>rd</sup> , 1) 11b=四个组成部分被转换 (a, b, c, d) = (1 <sup>st</sup> , 2 <sup>nd</sup> , 3 <sup>rd</sup> , 4 <sup>th</sup> ) (a, b, c, d) 通常用来选择 (X, Y, Z, W, )	00b
SrcW	[7:6]	选择 W 组成部分 00b=选择 a 组成部分作为 W 01b=选择 b 组成部分作为 W	11b

		10b=选择 c 组成部分作为 W 11b=选择 d 组成部分作为 W 注意：a~d 在 NumComp 区域定义	
SrcZ	[5:4]	选择 Z 组成部分 00b=选择 a 组成部分作为 Z 01b=选择 b 组成部分作为 Z 10b=选择 c 组成部分作为 Z 11b=选择 d 组成部分作为 Z 注意：a~d 在 NumComp 区域定义	10b
SrcY	[3:2]	选择 Y 组成部分 00b=选择 a 组成部分作为 Y 01b=选择 b 组成部分作为 Y 10b=选择 c 组成部分作为 Y 11b=选择 d 组成部分作为 Y 注意：a~d 在 NumComp 区域定义	01b
SrcX	[1:0]	选择 X 组成部分 00b=选择 a 组成部分作为 X 01b=选择 b 组成部分作为 X 10b=选择 c 组成部分作为 X 11b=选择 d 组成部分作为 X 注意：a~d 在 NumComp 区域定义	00b

#### 42.3.10.顶点缓冲区控制寄存器（FGHI\_ATTRIB0\_VBCTRL~FGHI\_ATTRIB9\_VBCTRL）

FGHI\_ATTRIBn\_VBCTRL.Stride 代表顶点缓冲区内下一个输入属性的字节数目。

FGHI\_ATTRIBn\_VBCTRL.num 代表在顶点缓冲区内有多少个输入属性。

寄存器	地址	读/写	描述	复位值
FGHI_ATTRIB0_VBCTRL	0x72008080	读/写	输入属性 0 的顶点缓冲区控制寄存器	0x00000000

FGHI_ATTRIB1_VBCTRL	0x72008084	读/写	输入属性 1 的顶点缓冲区控制寄存器	0x00000000
FGHI_ATTRIB2_VBCTRL	0x72008088	读/写	输入属性 2 的顶点缓冲区控制寄存器	0x00000000
FGHI_ATTRIB3_VBCTRL	0x7200808C	读/写	输入属性 3 的顶点缓冲区控制寄存器	0x00000000
FGHI_ATTRIB4_VBCTRL	0x72008090	读/写	输入属性 4 的顶点缓冲区控制寄存器	0x00000000
FGHI_ATTRIB5_VBCTRL	0x72008094	读/写	输入属性 5 的顶点缓冲区控制寄存器	0x00000000
FGHI_ATTRIB6_VBCTRL	0x72008098	读/写	输入属性 6 的顶点缓冲区控制寄存器	0x00000000
FGHI_ATTRIB7_VBCTRL	0x7200809C	读/写	输入属性 7 的顶点缓冲区控制寄存器	0x00000000
FGHI_ATTRIB8_VBCTRL	0x720080A0	读/写	输入属性 8 的顶点缓冲区控制寄存器	0x00000000
FGHI_ATTRIB9_VBCTRL	0x720080A4	读/写	输入属性 9 的顶点缓冲区控制寄存器	0x00000000

FGHI_ATTRIBn_VBCTRL	位	描述	初始状态
Stride	[31:24]	在字节内的下一属性位置	0xb
Reserved	[24:16]	保留	0
Range	[15:0]	顶点缓冲区内所有的有效索引范围。 此值用来指明索引的几何数据是否在顶点缓冲区内	0x0

**42.3.10.9 顶点缓冲区基础地址寄存器 (FGHI\_ATTR0\_VBBASE~FGH-ATTR9\_VBBASE)**

寄存器	地址	读/写	描述	复位值
FGHI_ATTRIB0_VBBASE	0x720080C0	读/写	输入属性 0 的顶点缓冲区基础地址寄存器	0x00000000
FGHI_ATTRIB1_VBBASE	0x720080C4	读/写	输入属性 1 的顶点缓冲区基础地址控制寄存器	0x00000000
FGHI_ATTRIB2_VBBASE	0x720080C8	读/写	输入属性 2 的顶点缓冲区基础地址控制寄存器	0x00000000
FGHI_ATTRIB3_VBBASE	0x720080CC	读/写	输入属性 3 的顶点缓冲区基础地址控制寄存器	0x00000000
FGHI_ATTRIB4_VBBASE	0x720080D0	读/写	输入属性 4 的顶点缓冲区基础地址控制寄存器	0x00000000
FGHI_ATTRIB5_VBBASE	0x720080D4	读/写	输入属性 5 的顶点缓冲区基础地址控制寄存器	0x00000000
FGHI_ATTRIB6_VBBASE	0x720080D8	读/写	输入属性 6 的顶点缓冲区基础地址控制寄存器	0x00000000
FGHI_ATTRIB7_VBBASE	0x720080DC	读/写	输入属性 7 的顶点缓冲区基础地址控制寄存器	0x00000000

FGHI_ATTRIB8_VBBASE	0x720080E0	读/写	输入属性 8 的顶点缓冲区基础地址控制寄存器	0x00000000
FGHI_ATTRIB9_VBBASE	0x720080E4	读/写	输入属性 9 的顶点缓冲区基础地址控制寄存器	0x00000000

FGHI_ATTRIBn_VBCTRL	位	描述	初始状态
Reserved	[31:16]	保留	0
Addr	[15:0]	顶点缓冲区内的输入属性的基础地址	0x0

## 42.4 顶点着色器

### 1.概述

顶点着色器是 3D 图形特殊处理器，可以处理顶点，替代传统的固定功能图形管道。顶点着色器可以使用户定义特殊功能。顶点着色器支持着色器模版 3.0，包括顶点纹理性能和不同的流量控制。

### 2. 最初的操作

顶点着色器程序由指令序列，算数运算的常量浮点值，流量控制地整数值和布尔值组成。这些值应该在执行程序之前储存在寄存器内或存储器内。当主机写入所有的顶点属性时顶点着色器自动开始运行。

### 3. 顶点着色器特殊寄存器

着色器指令和常量值在顶点着色器操作中储存在特殊寄存器内。这些寄存器可以通过主机接口更新。

### 4.指令存储器

指令存储器有 512 个插槽，每个插槽由 4 个字组成。

寄存器	地址	读/写	描述	复位值
FGVS_INSTMEM	0x72010000 ~0x72011FFF	读/写	顶点着色器的指令存储器	0xX

### 5. 常量浮点寄存器

常量浮点数可以储存在常量浮点寄存器内，用于程序中的计算操作。常量浮点寄存器有 256 个入口。每个入口由 4 个通道 x，y，z，w 组成。每个通道是 32 位字，屏息有 IEEE 单定居浮点格式。

寄存器	地址	读/写	描述	复位值
FGVS_CFLOAT	0x72014000 ~0x72014FFF	读/写	顶点着色器的常量浮点寄存器	

### 字 3 (0x72014XXC)

寄存器	位	描述	复位值
W	[127:96]	恒量浮点 W 组成部分值	0xFFFFFFFF

### 字 2 (0x72014XX8)

寄存器	位	描述	复位值
Z	[95:64]	恒量浮点 Z 组成部分值	0xFFFFFFFF

### 字 1 (0x72014XX4)

寄存器	位	描述	复位值
Y	[63:32]	恒量浮点 Y 组成部分值	0xFFFFFFFF

### 字 0 (0x72014XX0)

寄存器	位	描述	复位值
X	[31:0]	恒量浮点 X 组成部分值	0xFFFFFFFF

### IEEE 单精度浮点格式

	位	描述	复位值
S	[31]	Sign 位	XXXXXXXh
E	[30:23]	偏向指数	XXh
F	[22:0]	分数	

## 6.常量整数寄存器

常量整数值存储在常量整数寄存器内。常量整数值只用于流量控制，是循环计数的迭代或相关地址的索引。常量整数寄存器有 16 个入口，每个入口由 4 个 8 位无符号整数值的通道组成。



寄存器	地址	读/写	描述	复位值
FGVS_CINT	0x72018000 ~0x7201803F	读/写	顶点着色器的常量整数寄存器	0xFFFFFFFF

字 0 (0x72018XX0)

	位	描述	复位值
W	[31:24]	常量整数 W 组成部分值	0xX
Z	[23:16]	常量整数 Z 组成部分值	0xX
Y	[15:8]	常量整数 Y 组成部分值	0xX
X	[7:0]	常量整数 X 组成部分值	0xX

### 7.常量布尔寄存器

常量布尔值可以储存在常量布尔寄存器内。常量布尔值只用于静态流量控制。常量布尔寄存器是 16 位的布尔寄存器。寄存器序号与每个位位置相对应。**TRUE** 用 1 表示，**FALSE** 用 0 表示。

寄存器	地址	读/写	描述	复位值
FGVS_CB00L	0x72018400	读/写	顶点着色器的常量布尔寄存器	0xFFFFFFFF

CB00L	位	描述	初始状态
Reserved	[31:16]	保留	0xFFFF
REG15	[15]	常量布尔寄存器 15	X
REG14	[14]	常量布尔寄存器 14	X
REG13	[13]	常量布尔寄存器 13	X
REG12	[12]	常量布尔寄存器 12	X
REG11	[11]	常量布尔寄存器 11	X
REG10	[10]	常量布尔寄存器 10	X
REG9	[9]	常量布尔寄存器 9	X
REG8	[8]	常量布尔寄存器 8	X
REG7	[7]	常量布尔寄存器 7	X

REG6	[6]	常量布尔寄存器 6	X
REG5	[5]	常量布尔寄存器 5	X
REG4	[4]	常量布尔寄存器 4	X
REG3	[3]	常量布尔寄存器 3	X
REG2	[2]	常量布尔寄存器 2	X
REG1	[1]	常量布尔寄存器 1	X
REG0	[0]	常量布尔寄存器 0	X

8. 顶点着色器配置寄存器

全局寄存器包含多种配置和环境进行全局操作。

寄存器	地址	读/写	描述	复位值
FGVS_Config	0x7201C800	写	顶点着色器配置寄存器	0x00000000
FGVS_Status	0x7201C804	读	内部状态寄存器	0x00000000
FGVS_PCRange	0x72020000	读/写	顶点着色器程序开始和结束地址	0x01FF0000
FGVS_AttributeNum	0x72020004	读/写	当前文本属性的数量	0x00010001

FGVS_Config	位	描述	初始状态
Reserved	[31:1]	保留	0
ClrStatus	[1]	当此位被设置为 1 时，FGVS_Status 寄存器所有的值被清除。 清除操作完成后此位自动为 0.	0b
CopyPC	[0]	当此位设置为 1，FGVS_PCRange 寄存器值被复制到定点着色器 里面。复制完成后，此位自动清除到 0.  在没有任何复制命令的时候，FGVS_PCRange 的值不被使用，用 先前的值运行开始或结束地址。	0b

FGVS_PCRange	位	描述	初始状态
IgnorePCEnd	[31]	当此位设置为 1 时，PCEnd 值被忽略，只有程序计数器栈空条	0

		件被程序终端应用。	
Reserved	[30:25]	保留	0
PCEnd	[24:16]	<p>当程序计数器达到 PCEnd 的值，着色器程序在寄存器内的指令执行后被终止。这种方法可以保存指令槽和指令计数的数量用于执行。</p> <p>另一种终止顶点着色器程序的方法：</p> <p>顶点着色器程序可以通过额外的“ret”指令终止，“ret”指令使程序计数栈为空条件。通常“call”和“ret”指令成对工作。但是故意不匹配的“ret”使顶点着色器终止。通过这个特例，顶点着色器程序可以被终止。</p>	0x1FF
Reserved	[15:9]	保留	0
PCStart	[8:0]	<p>当顶点着色器开始运行，从指令存储器获取的第一个指令被储存在 PCStart 上。</p> <p>这个寄存器值可以复制到顶点着色器程序计数器上，在着色器开始之前通过 PCCopy 寄存器完成。</p>	0x00

FGVS_AttributeNum	位	描述	初始状态
Reserved	[31:4]	保留	0x0001000
InAttributeNum	[3:0]	<p>顶点着色器输入属性的序号。</p> <p>需要与当前着色器程序输入寄存器序号匹配。</p>	0x1

## 9. 输入属性的索引寄存器

通常，顶点着色器输入寄存器的寄存器序号与主机输入属性的顺序相匹配。这种关系可以通过输入属性索引寄存器重新映射。主机的第 n 个输入属性实际上是位值的读数，通过索引寻找着色器内输入寄存器相应的寄存器序号可以指明。

寄存器	地址	读/写	描述	复位值
FGVS_InAttrIndex0	0x72020008	写	输入属性 0~3 的索引	0x03020100
FGVS_InAttrIndex1	0x7202000C	写	输入属性 4~7 的索引	0x07060504
FGVS_InAttrIndex2	0x72020010	写	输入属性 8~9 的索引	0x0B0A0908

FGVS_InAttrIndex0	位	描述	初始状态
Reserved	[31:28]	保留	0
Attrib3	[27:24]	输入属性 3 的索引	0x3
Reserved	[23:20]	保留	0
Attrib2	[19:16]	输入属性 2 的索引	0x2
Reserved	[15:12]	保留	0
Attrib1	[11:8]	输入属性 1 的索引	0x1
Reserved	[7:4]	保留	0
Attrib0	[3:0]	输入属性 0 的索引	0x0

FGVS_InAttrIndex1	位	描述	初始状态
Reserved	[31:28]	保留	0
Attrib7	[27:24]	输入属性 7 的索引	0x7
Reserved	[23:20]	保留	0
Attrib6	[19:16]	输入属性 6 的索引	0x6
Reserved	[15:12]	保留	0
Attrib5	[11:8]	输入属性 5 的索引	0x5
Reserved	[7:4]	保留	0
Attrib4	[3:0]	输入属性 4 的索引	0x4

FGVS_InAttrIndex2	位	描述	初始状态
Reserved	[31:12]	保留	0x0B0A0
Attrib9	[11:8]	输入属性 9 的索引	0x9
Reserved	[7:4]	保留	0
Attrib8	[3:0]	输入属性 8 的索引	0x8

10.输出属性的索引寄存器

通常，像素着色器输入寄存器的寄存器序号与顶点着色器的输出寄存器的序号相对应。这种关系可以通过输出属性索引寄存器重新映射。第 n 个输出属性实际上是位值的写入，通过索引寻找着色器内输出寄存器相应的寄存器序号可以指明。输入寄存器 0 从输出寄存器 1 获得数值，因为顶点着色器的输出寄存器 0 的位置是特别指明的。因此，顶点着色器的输出寄存器 1-8 与像素着色器的 0-7 相对应。

寄存器	地址	读/写	描述	复位值
FGVS_OutAttrIndex0	0x72020014	写	输出属性 0~3 的索引	0x03020100
FGVS_OutAttrIndex1	0x72020018	写	输出属性 4~7 的索引	0x07060504
FGVS_OutAttrIndex2	0x7202001C	写	输出属性 8~9 的索引	0x0B0A0908

FGVS_OutAttrIndex0	位	描述	初始状态
Reserved	[31:28]	保留	0
Attrib3	[27:24]	输出属性 3 的索引	0x3
Reserved	[23:20]	保留	0
Attrib2	[19:16]	输出属性 2 的索引	0x2
Reserved	[15:12]	保留	0
Attrib1	[11:8]	输出属性 1 的索引	0x1
Reserved	[7:4]	保留	0
Attrib0	[3:0]	输出属性 0 的索引	0x0

FGVS_OutAttrIndex1	位	描述	初始状态
Reserved	[31:28]	保留	0
Attrib7	[27:24]	输出属性 7 的索引	0x7
Reserved	[23:20]	保留	0
Attrib6	[19:16]	输出属性 6 的索引	0x6
Reserved	[15:12]	保留	0
Attrib5	[11:8]	输出属性 5 的索引	0x5
Reserved	[7:4]	保留	0

Attrib4	[3:0]	输出属性 4 的索引	0x4
---------	-------	------------	-----

FGVS_OutAttrIndex2	位	描述	初始状态
Reserved	[31:12]	保留	0x0B0A0
Attrib9	[11:8]	输出属性 9 的索引	0x9
Reserved	[7:4]	保留	0
Attrib8	[3:0]	输出属性 8 的索引	0x8

42.5 图元引擎

42.5.1.概述

图元引擎是一个有线硬件模块，可以处理一系列的操作，包括原始集合，平面着色，视图体裁剪，视图划分和视图映射。视图体裁剪划分为近/远平面裁剪操作和左/右/顶/底平面裁剪操作。在图元引擎内，只处理近/远平面裁剪操作，除了当裁剪的顶点 **w**-坐标为 0 以外。使用窗口裁剪操作，其它裁剪操作可以通过三角设置引擎和光栅化引擎处理。当顶点的 **W**-坐标为 0 时，图元引擎运行其他裁剪操作避免输出顶点的 **W**-坐标变为 0.

图 42-9 所示图形表示出图元引擎的概念功能级模块图，解释说明了图元引擎的操作。

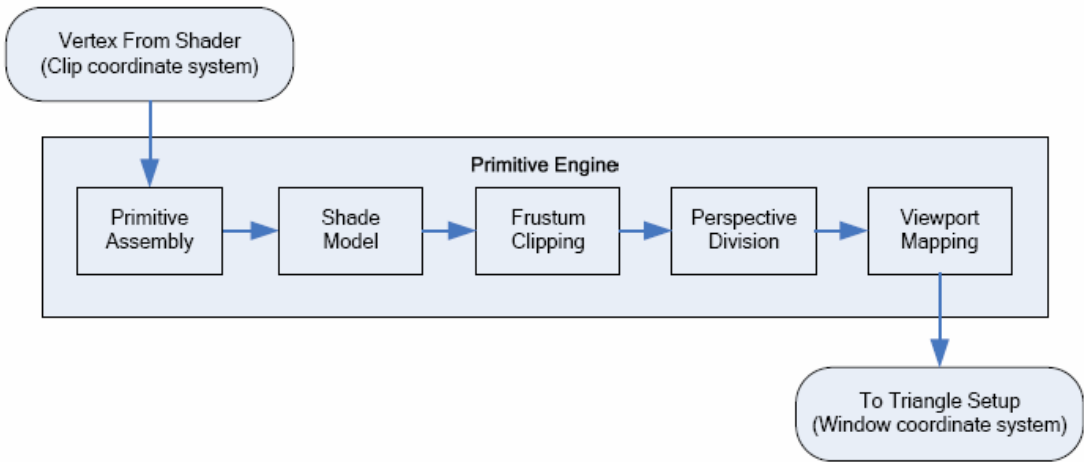


图 42-9 图元引擎的概念功能级模块图

## 42.5.2.图元引擎特殊寄存器

在图元引擎内有两种特殊寄存器。一个寄存器用于表示顶点信息，如图元类型，相关数据的数目以及色调模式。另一个寄存器用于表示视口参数。

## 42.5.3.顶点背景寄存器

在图元引擎内，顶点背景寄存器将一个输入顶点分为 3 种信息进行处理。29-29 之间的位区域表示返回的图元类型。支持图元引擎寄存器，三角形，三角扇形，三角地带，线，线环，带，点和点带。其他类型的如多边形，方形，方形带不支持，但是对于这些形状的位已经保留了。每个图元类型的位区域都相互正交。这就暗示着当用点带模式是，点带的位必须从新设置。

13-10 之间的位区域表示编码模式下与顶点相关数据的序号。MSB 的位是 13，LSB 的位是 10.相关数据的序号位宽度假设为 128。如果顶点相关数据是 5 个 32 位值，那么相关数据的序号必须设置为 2。当想要使用顶点着色器点尺寸模式时，相关数据的序号必须加 1。如上所述，相关数据的序号与顶点着色器的变量和顶点着色器程序点尺寸模式有关，因此相关数据需要的范围是 0-9。可以通过下面的式子表示相关数据的序号和 VSOUT:

$$VSOUT = \frac{\text{变量数}}{4} + 1, \quad \text{顶点程序点尺寸模式}$$

$$VSOUT = \frac{\text{变量数}}{4}, \quad \text{其他情况}$$

9-0 之间的位区域表示色调模式信息。位 9 是主标志，表示色调模式是平滑的或平坦的。0-8 位区域与顶点着色器输出有关，为了用顶点着色器输出 Slot0 为平坦颜色通道，位 9 和位 4 必须设置为 1.

位 31 和 30 是内部使用的位，不要触摸这两个位。

注意：在 3D 图形内，顶点着色器支持的输入属性数量可达 8 个，变量数可达 32 个。

寄存器	地址	读/写	描述	复位值
FGVS_VERTEX_CONTEXT	0x72030000	读/写	顶点背景格式定义	0x00000000

FGVS_VERTEX_CONTEXT	位	描述	初始状态
INTUSE	[31:30]	保留的位，用于内部使用。不要触摸这些位。	0b
POLYGON	[29]	保留的位，用于多边形图元类型	0b
QUADS	[28]	保留的位，用于方形图元类型	0b
QUADSTRIP	[27]	保留的位，用于方形带图元类型	0b
TRIANGLES	[26]	1b=三角形图元类型 0b=非三角形图元类型	0b
TRIANGLEFAN	[25]	1b=三角扇形图元类型 0b=非三角扇形图元类型	0b
TRIANGLESTRIP	[24]	1b=三角带形图元类型 0b=非三角带形图元类型	0b
LINES	[23]	1b=线形图元类型 0b=非线形图元类型	0b
LINELOOP	[22]	1b=线环形图元类型 0b=非线环形图元类型	0b
LINESTRIP	[21]	1b=线带形图元类型 0b=非线带形图元类型	0b
POINTS	[20]	1b=点图元类型 0b=非点图元类型	0b
POINTSPRITE	[19]	1b=点带图元类型 0b=点带角形图元类型	0b
VERTEXPROGRAMPOINTSIZ	[18]	1b=顶点着色器程序点尺寸模式 0b=非顶点着色器程序点尺寸模式	0b
Reserved	[17:14]	保留	0
VSOUT[3:0]	[13:10]	顶点着色器使用的位置，输出数目	0x0



FLAT_SHADE	[9]	1b=使用平坦色调模式 0b=使用平滑色调模式	0b
FLAT_MODEL8	[8]	1b=顶点着色器输出 8 使用平坦色调模式 0b=顶点着色器输出 8 使用平滑色调模式	0b
FLAT_MODEL7	[7]	1b=顶点着色器输出 7 使用平坦色调模式 0b=顶点着色器输出 7 使用平滑色调模式	0b
FLAT_MODEL6	[6]	1b=顶点着色器输出 6 使用平坦色调模式 0b=顶点着色器输出 6 使用平滑色调模式	0b
FLAT_MODEL5	[5]	1b=顶点着色器输出 5 使用平坦色调模式 0b=顶点着色器输出 5 使用平滑色调模式	0b
FLAT_MODEL4	[4]	1b=顶点着色器输出 4 使用平坦色调模式 0b=顶点着色器输出 4 使用平滑色调模式	0b
FLAT_MODEL3	[3]	1b=顶点着色器输出 3 使用平坦色调模式 0b=顶点着色器输出 3 使用平滑色调模式	0b
FLAT_MODEL2	[2]	1b=顶点着色器输出 2 使用平坦色调模式 0b=顶点着色器输出 2 使用平滑色调模式	0b
FLAT_MODEL1	[1]	1b=顶点着色器输出 1 使用平坦色调模式 0b=顶点着色器输出 1 使用平滑色调模式	0b
FLAT_MODEL0	[0]	1b=顶点着色器输出 0 使用平坦色调模式 0b=顶点着色器输出 0 使用平滑色调模式	0b

#### 42.5.4. 视口参数寄存器

视口转变由适口宽度和高度像素  $P_x$  和  $P_y$  决定，视口中心 ( $O_x, O_y$ ) 也是像素模式。顶点窗口坐标  $\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix}$

由下面的式子表示：

$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} (p_x/2)x_d + o_x \\ (p_y/2)y_d + o_y \\ [(f-n)/2]z_d + (n+f)/2 \end{pmatrix}$$

应用到  $Z_d$  的因子和补偿区通过近深度范围  $n$  和远深度范围  $f$  编码。视口中心  $(O_x, O_y)$  可以用  $(x_0 + p_x/2, y_0 + p_y/2)$  表示，假设视口原点为  $(x_0, y_0)$ 。在 3D 图形内，使用 Y 轴滚动窗口坐标系统。为了产生 y 周滚动窗口坐标系统，可以简单的替换 y 轴相关等式， $y_w = (p_y/2)y_d + o_y$  和  $o_y = y_0 + p_y/2$ ， $y_w = (-p_y/2)y_d + o_y$  和  $o_y = (\text{窗口高度}) - y_0 - p_y/2$ 。

寄存器	地址	读/写	描述	复位值
FGPE_VEIWPOROT_OX	0x72030004	读/写	视口中心 X 坐标	0xX
FGPE_VEIWPOROT_OY	0x72030008	读/写	视口中心 Y 坐标	0xX
FGPE_VEIWPOROT_HALF_PX	0x7203000C	读/写	视口宽度的一半	0xX
FGPE_VEIWPOROT_HALF_PY	0x72030010	读/写	视口高度的一半	0xX
FGPE_DEPTHRANGE_HALF_F_SUB_N	0x72030014	读/写	远深度范围的一般减近深度范围的一半	0xX
FGPE_DEPTHRANGE_HALF_F_ADD_N	0x72030018	读/写	远深度范围的一半加近深度范围的一半	0xX

$P_x$ : 视口宽度像素

$P_y$ : 视口高度像素

$X_0$ : 窗口坐标系统内视口中心的 x 坐标

$Y_0$ : 窗口坐标系统内视口中心的 y 坐标

$n$ : 深度范围的近值

$f$ : 深度范围的远值

$H$ : 窗口高度像素

	位	描述	初始状态
FGPE_VIEW PORT_OX	[31:0]	视口中心的 x 坐标。 $x_0 + \frac{p_x}{2}$	0xX

	位	描述	初始状态
FGPE_VIEW PORT_OY	[31:0]	视口中心的 x 坐标。  $y_0 + \frac{p_y}{2}$ 如果想产生 y 滚动窗口坐标，设置 SFR 如下：  $(H - y_0) - \frac{p_y}{2}$	0xX

	位	描述	初始状态
FGPE_VIEW PORT_HALE_PX	[31:0]	视口宽度的一半值  $\frac{p_x}{2}$	0xX

	位	描述	初始状态
FGPE_VIEW PORT_HALE_PY	[31:0]	视口高度的一半 $\frac{p_y}{2}$  如果想产生 y 滚动窗口坐标，设置 SFR 如下： $-\frac{p_y}{2}$	0xX

	位	描述	初始状态
FGPE_DEPTHARANGE_H_ALF_F_SUB_N	[31:0]	深度范围远值减去近值的一半值  $\frac{f - n}{2} \quad (n, f \in [0, 1])$	0x3F000000

	位	描述	初始状态
FGPE_DEPTHARANGE_H_ALF_F_ADD_N	[31:0]	深度范围远值加上近值的一半值  $\frac{f + n}{2} \quad (n, f \in [0, 1])$	0x3F000000

# 42.6 光栅引擎

光栅引擎包括山脚设计引擎和光栅化引擎

## 42.6.1.三角设置引擎概述

图元 y 值边界

边缘插值

三角形梯度计算

深度补偿计算

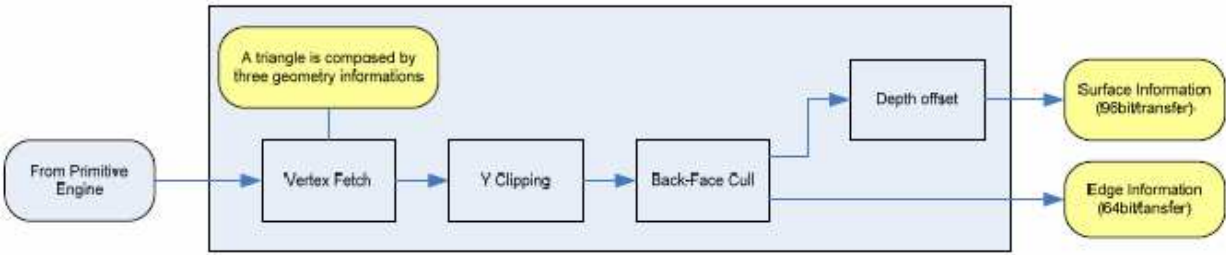


图 42-10 三角设置引擎模块图

## 42.6.2.光栅化引擎概述

支持三角形，线，点，点带

一个周期产生一个片段

片段位置值计算

片段值计算

LOD 选择系数的计算

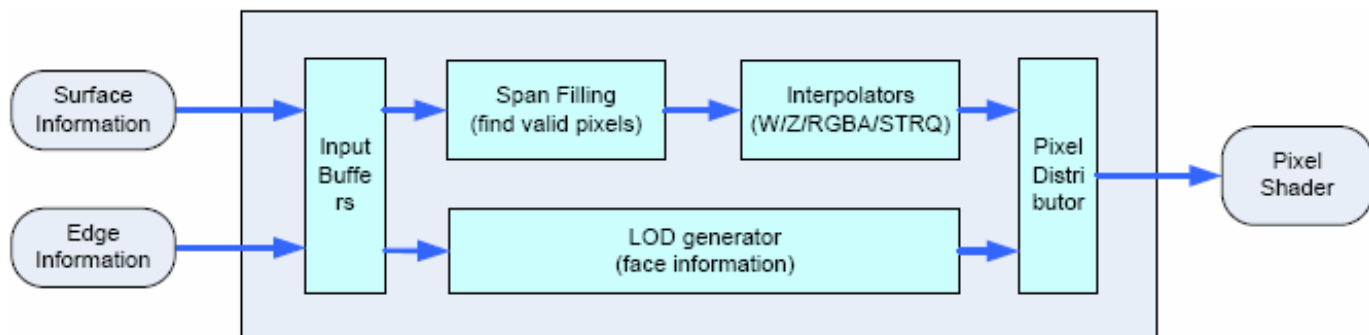


图 42-11 光栅化引擎模块图

### 42.6.3.初始操作

光栅引擎操作资料：

采样位置控制寄存器：这个寄存器指明在像素的中心或新像素进行采样。

深度补偿控制寄存器：设置此寄存器非常必要，因数值和对象单元使用深度补偿。深度补偿寄存器值只在背景转换的时间内改变。

背面采集控制寄存器：运行背面采集寄存器、前面采集寄存器并使所有的寄存器正确的设置。这个寄存器值只在背景转换时间内改变。

LOD 控制寄存器：如果像素着色器用 DDX/DDY/LOD，设置此寄存器非常必要。这个寄存器值可以在像素着色器没有使用 DDX/DDY/LOD 时设置。这个寄存器值只在背景转换时间内改变。

窗口 X/Y 剪裁范围控制寄存器：通过观看截图剪裁窗口范围

Coord 替换控制寄存器：只用于点带成像。对于 PointSprite，TSE 内产生的纹理坐标属性附于最后属性的后面。因此 Coord 替换控制位应该被正确的设置为与附加的纹理坐标属性相对应的属性序号。

点宽度控制寄存器：点尺寸值词义通过 SFR,PointWidth 值设置。点尺寸值可以从顶点着色器内取得。因此，应该设置图元引擎内的 VPPS SPR 。

线宽度控制寄存器：用于分配线宽度值。

## 42.6.4.光栅引擎特殊寄存器

### 42.6.4.1.采样位置寄存器

寄存器	地址	读/写	描述	复位值
FGRA_PixSamp	0x72038000	读/写	指明像素采样位置寄存器	0x00000000

FGRA_PixSamp	位	描述	初始状态
Reserved	[31:1]	保留	
PixComerSamp	[0]	选择取得纹理像素或色调像素使用的采样位置 0b= (0.5, 0.5) 中心位置 1b= (0, 0) 左上角位置	0b

### 42.6.4.2. 深度补偿寄存器

多边形的补偿值  $O$  的表达式如下：

$$O=m*\text{因子}+r*\text{单元}$$

$M$  是多边形深度值增量比的最大值。 $O$  值的范围是[0,1]。 $R$  是 H/W 性能值，与深度缓冲区的可用分辨率有关。补偿值  $O$  应用于多边形所有像素的深度值。

寄存器	地址	读/写	描述	复位值
FGRA_DoffEn	0x72038004	读/写	深度补偿使能寄存器	0x00000000
FGRA_DoffFactor	0x72038008	读/写	深度补偿因子寄存器	0x00000000
FGRA_DoffUnites	0x7203800C	读/写	深度补偿单元寄存器	0x00000000
FGRA_DoffRln	0x72038010	读/写	深度补偿执行常量 $r$ 值	0x34000001

FGRA_DoffEn	位	描述	初始状态
Reserved	[31:1]	保留	0
DoffEn	[0]	深度补偿使用控制寄存器	0b

		0b=disable      1b=enable	
--	--	---------------------------	--

FGRA_DoffFactor	位	描述	初始状态
DoffFactor	[31:0]	因子乘以多边形的最大深入斜度. 用于计算深度补偿值	0x0

FGRA_ DoffUnites	位	描述	初始状态
DoffUnites	[31:0]	单元乘以与深度缓冲区的使用分辨率有关的执行依据常量。 用于计算深度补偿值	0x0

FGRA_ DoffRln	位	描述	初始状态
DoffRln	[31:0]	与深度缓冲区的使用分辨率有关的执行依据常量。 用于计算多边形深度补偿值	0x0

3. 背面采集控制寄存器

寄存器	地址	读/写	描述	复位值
FGRA_BFCULL	0x72038014	读/写	背面采集控制寄存器	0x00000000

FGRA_BFCULL	位	描述	初始状态
Reserved	[31:4]	保留	0
BCullEn	[3]	背面采集使能 0b=disable      1b=enable	0b
BcullFront	[2]	前面选择 0b=CCW      1b=CW	0b
BcullFace	[1:0]	采集面选择 00=背面      01b=前面 10b=保留      11b=前面和背面	00b

42.6.4.4.窗口 Y 剪裁区域寄存器

在编程窗口内，屏幕左下角的像素为（0，0）。在 H/W 视图内，（0，0）像素在左上角。因此，设置 Y 裁剪值时需要滚动 Y 的坐标值。从设备驱动内接收到 dd\_min\_val 和 dd\_max\_val 值后，用下面的等式取得最大值和最小值

MAX=(screen\_height\_val -1)- dd\_min\_val  
MIN=(screen\_height\_val -1)- dd\_max\_val

寄存器	地址	读/写	描述	复位值
FGRA_YCLIP	0x72038018	读/写	Y 剪裁坐标寄存器	0x00000000

FGRA_YCLIP	位	描述	初始状态
Reserved	[31:28]	保留	0
YMaxVal	[27:16]	Y 裁剪最大坐标（y< YMaxVal）	0x0
Reserved	[15:12]	保留	0
YMinVal	[11:0]	Y 裁剪最小坐标（YMaxVal ≤ y）	0x0

42.6.4.5 细节等级控制寄存器

这个寄存器服务更多的可编程像素着色器。LOD 系数可以用在像素着色器上，可以通过 LODCTL 控制 LOD 系数。根据 LODCTL 产生的 LOD 系数如下：

DDY	DDX	LOD	LOD 系数			
0	0	0	没有			
0	0	1	K1 K3 K5	K2 K4 K6		
0	1	0	K1 K3	K2 K4	K7 K9	
			K8 K10			
0	1	1	K1 K3 K5	K2 K4 K6	K7 K9	K8 K10



1	0	0	K1 K5 K2 K6 K7 K11 K8 K12
1	0	1	K1 K3 K5 K2 K4 K6 K7 K11 K8 K12
1	1	0	K1 K3 K5 K2 K4 K6 K7 K9 K11 K8 K10 K12
1	1	1	K1 K3 K5 K2 K4 K6 K7 K9 K11 K8 K10 K12

寄存器	地址	读/写	描述	复位值
FGRA_LODCTL	0x7203C000	读/写	指明 LOD 计算控制寄存器	0x00000000

FGRA_LODCTL	位	描述	初始状态
Reserved	[31:24]	保留	0
LodCon7	[23:21]	属性 7 的 {DDY, DDX, LOD} 000b=所有的 LOD 系数不可用 001b=LOD 系数计算可用 010b= DDX 系数计算可用 011b=DDX 和 LOD 系数可用 100b=DDY 系数计算可用 101b=DDY 和 LOD 系数可用 110b=DDY 和 DDX 系数可用 111b=所有 LOD 系数可用	000b
LodCon6	[20:18]	与上面的相同	000b
LodCon5	[17:15]	与上面的相同	000b
LodCon4	[14:12]	与上面的相同	000b
LodCon3	[11:9]	与上面的相同	000b
LodCon2	[8:6]	与上面的相同	000b
LodCon1	[5:3]	与上面的相同	000b
LodCon0	[2:0]	与上面的相同	000b

42.6.4.6 窗口 X 剪裁范围寄存器

寄存器	地址	读/写	描述	复位值
FGRA_CLIPX	0x72038018	读/写	Y 剪裁坐标寄存器	0x00000000

FGRA_CLIPX	位	描述	初始状态
Reserved	[31:28]	保留	0
Xright	[27:16]	X 裁剪右边（最大）坐标 ( $y < X_{right}$ )	0x0
Reserved	[15:12]	保留	0
Xleft	[11:0]	X 裁剪左边（最小）坐标 ( $X_{left} \leq y$ )	0x0

42.6.4.7 点宽度控制寄存器

点宽度值在点宽度最小值和电宽度最大值之间

寄存器	地址	读/写	描述	复位值
FGRA_PWIDTH	0x7203801C	读/写	点宽度控制寄存器	0x3F800000
FGRA_PSIZE_MIN	0x72038020	读/写	点宽度最小值控制寄存器	0x3F800000
FGRA_PSIZE_MAX	0x72038024	读/写	点宽度最大值控制寄存器	0x45000000

FGRA_PWIDTH	位	描述	初始状态
PointWidth	[31:0]	指定点宽度值（浮点）	0x3F800000

FGRA_PSIZE_MIN	位	描述	初始状态
PointSize_Min	[31:0]	指定点宽度最小值（浮点）	0x3F800000

FGRA_PSIZE_MAX	位	描述	初始状态
PointSize_Max	[31:0]	指定点宽度最大值（浮点）	0x45000000

### 42.6.4.8 COORD 替换控制寄存器

这个寄存器用于点带成像时间内的纹理坐标产生。8 个控制位中只有一个位可以被激活。产生的点带纹理坐标被储存在属性寄存器内，由选择的序号表示。

寄存器	地址	读/写	描述	复位值
FGRA_COORDREPLACE	0x72038028	读/写	Coord 替换控制寄存器	0x00000000

FGRA_COORDREPLACE	位	描述	初始状态
Reserved	[31:8]	保留	0x0
CoordReplace7	[7]	属性 7 的 Coord 替换控制位	0
CoordReplace6	[6]	属性 6 的 Coord 替换控制位	0
CoordReplace5	[5]	属性 5 的 Coord 替换控制位	0
CoordReplace4	[4]	属性 4 的 Coord 替换控制位	0
CoordReplace3	[3]	属性 3 的 Coord 替换控制位	0
CoordReplace2	[2]	属性 2 的 Coord 替换控制位	0
CoordReplace1	[1]	属性 1 的 Coord 替换控制位	0
CoordReplace0	[0]	属性 0 的 Coord 替换控制位	0

### 42.6.4.9 线宽度控制寄存器

寄存器	地址	读/写	描述	复位值
FGRA_LWIDTH	0x7203802C	读/写	线宽度控制寄存器	0x3F800000

FGRA_LWIDTH	位	描述	初始状态
LineWidth	[31:0]	指定线宽度值（浮点）	0x3F800000

# 42.7 像素着色器

## 42.7.1.概述

着色器由 4 路浮点 SIMD 构架和小标量核心组成。每个数据路径有 4 路浮点类型寄存器和标量寄存器。

简化的像素着色器模块图和它的接口如图 42-12 所示。输入数据是像素属性，如位置，颜色，纹理坐标。用于着色器执行的指令和预定常量从主机处理器内下载。临时寄存器，循环计数寄存器，属性寄存器互相协作用于像素处理。对于纹理映射而言，像素着色器与纹理单元相互作用。处理的像素数据最后通过数据寄存器转换到预片元单元。

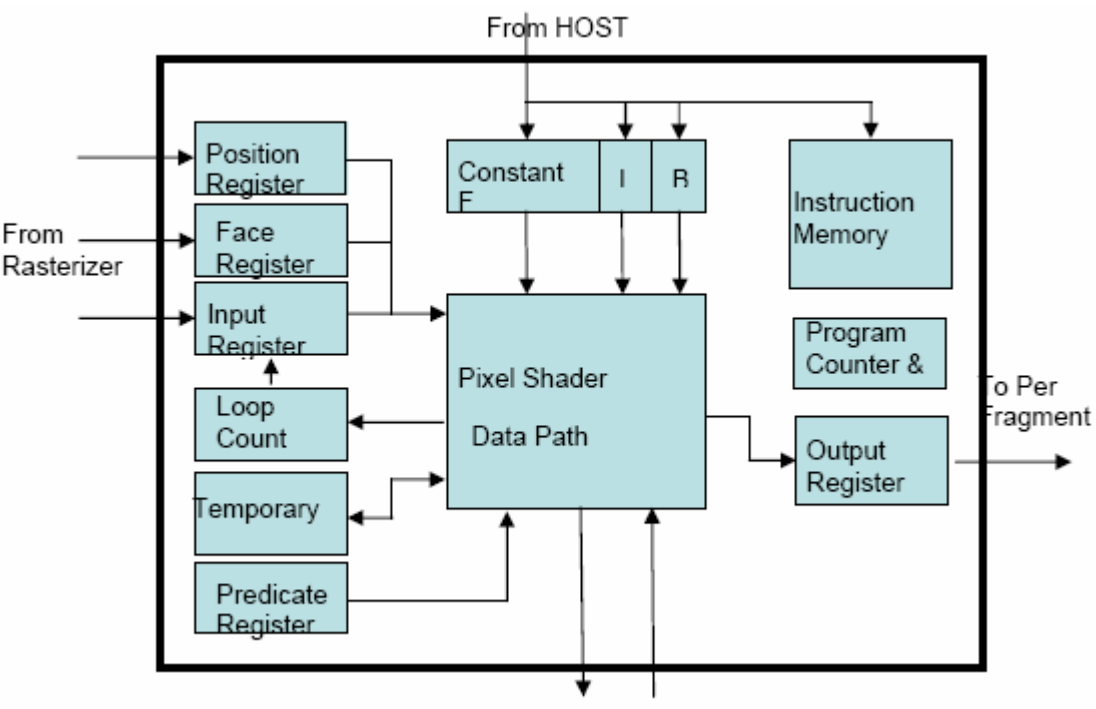


图 42-12 像素着色器模块图

可编程着色器根据不同的用途分为两个寄存器组。一个寄存器组是用于 HW 配置的特殊功能寄存器，另一个寄存器组是着色器程序的程序寄存器。只能通过主机 CPU 访问 SFR。一些程序寄存器如指令存储器，常量浮点寄存器，常量整数寄存器和常量布尔寄存器可以通过主机 CPU 和可编程的着色器操作模式的着色器本身访问。在此文本内，简要的概述了可以通过主机 CPU 访问的寄存器。

### 42.7.2. 指令存储器

指令存储器有 512 个槽，每个槽由 4 个字组成。

寄存器	地址	读/写	描述	复位值
INSTMEM	0x72040000 ~ 0x72041FFF	写	像素着色器的指令存储器	0xX

### 42.7.3. 常量浮点寄存器

常量浮点数可以储存在常量浮点寄存器内，用于在程序中计数。常量浮点寄存器有 256 个入口，每个入口由 4 个通道 x, y, z, w 组成。每个通道是 32 位字，并且是 IEEE 单精度浮点格式（不支持双精度归一化数）

寄存器	地址	读/写	描述	复位值
FGPS_CFLOAT	0x72044000~ 0x72044FFF	写	像素着色器的常量浮点寄存器	

字 3 (0x72044XXC)

字 3	位	描述	初始状态
W	[127:96]	常量浮点 W 组成部分值	0xFFFFFFFF

字 2 (0x72044XX8)

字 2	位	描述	初始状态
Z	[95:64]	常量浮点 Z 组成部分值	0xFFFFFFFF

字 1 (0x72044XX4)

字 1	位	描述	初始状态
Y	[63:32]	常量浮点 Y 组成部分值	0xFFFFFFFF

字 0 (0x72044XX0)

字 0	位	描述	初始状态
X	[31:0]	常量浮点 X 组成部分值	0xFFFFFFFF

IEEE 单精度浮点格式

	位	描述	复位值
S	[15]	Sign 位	0xXXXXXX
E	[14:10]	偏向指数	0xXXX
F	[9:0]	分数	0xXXX

## 42.7.4.常量整数寄存器

常量整数值可以储存在常量整数寄存器内。常量整数值只用于流量控制。常量整数寄存器有 16 个入口，每个入口由 4 个通道 8 位无符号整数值。

寄存器	地址	读/写	描述	复位值
FGPS_CINT	0x72048000 ~0x7204803F	写	像素色器的常量整数寄存器	0xFFFFFFFF

字 (0x72048XX0)

	位	描述	复位值
W	[31:24]	常量整数 W 组成部分值	0xX
Z	[23:16]	常量整数 Z 组成部分值	0xX
Y	[15:8]	常量整数 Y 组成部分值	0xX
X	[7:0]	常量整数 X 组成部分值	0xX

### 42.7.5.常量布尔寄存器

常量布尔值可以储存在常量布尔寄存器内。常量布尔值只用于静态流量控制。常量布尔寄存器是 16 位的布尔寄存器。寄存器序号与每个位位置相对应。**TRUE** 用 1 表示，**FALSE** 用 0 表示。

寄存器	地址	读/写	描述	复位值
FGPS_CB00L	0x72048400	读/写	像素着色器的常量布尔寄存器	0x0

FGPS_CB00L	位	描述	初始状态
Reserved	[31:16]	保留	0xFFFF
REG15	[15]	常量布尔寄存器 15	X
REG14	[14]	常量布尔寄存器 14	X
REG13	[13]	常量布尔寄存器 13	X
REG12	[12]	常量布尔寄存器 12	X
REG11	[11]	常量布尔寄存器 11	X
REG10	[10]	常量布尔寄存器 10	X
REG9	[9]	常量布尔寄存器 9	X
REG8	[8]	常量布尔寄存器 8	X
REG7	[7]	常量布尔寄存器 7	X
REG6	[6]	常量布尔寄存器 6	X
REG5	[5]	常量布尔寄存器 5	X
REG4	[4]	常量布尔寄存器 4	X
REG3	[3]	常量布尔寄存器 3	X
REG2	[2]	常量布尔寄存器 2	X
REG1	[1]	常量布尔寄存器 1	X
REG0	[0]	常量布尔寄存器 0	X

### 42.7.6. HW 配置的特殊功能寄存器

全局寄存器包含多种配置和环境进行全局操作。

寄存器	地址	读/写	描述	复位值
FGPS_ExeMode	0x7204C800	读/写	像素着色器之星模式控制寄存器	0x0
FGPS_PCStart	0x7204C804	读/写	像素着色器程序开始地址	0x0
FGPS_PCEnd	0x7204C808	读/写	像素着色器程序结束地址	0x0
FGPS_PCCopy	0x7204C80C	读/写	复制 PSPCS_ADDR 值到程序计数器	0x0
FGPS_AttributeNum	0x7204C810	读/写	当前环境的属性序号	0x0
FGPS_IBStatus	0x7204C814	读	PS 输入缓冲区初始化的状态信号未准备好 0: 准备好     1: 未准备好	0x0

FGPS_ExeMode	位	描述	初始状态
Reserved	[31:1]	保留	0
FGPS_ExeMode	[0]	<p>选择像素着色器执行模式寄存器</p> <p>0: 主机访问模式。主机 CPU 可以访问 HW 配置寄存器。一些程序寄存器，如指令存储器，浮点数/整数/布尔常量寄存器可以在此模式下下载。</p> <p>1: 像素着色器执行模式。(PSExeMode)。像素着色器运行常规操作。在这种模式下，主机 CPU 可以读取 FGPS_IBStatus 和 FGPS_IsNotEmpty 状态寄存器。如果主机 CPU 改变其他寄存器值，像素着色器操作将难以预测。</p> <p>模式变化包括：</p> <p>（1）PSHostMode→PSExeMode：</p> <p>设置所有配置寄存器值</p> <p>下载指令，常数 F/B/I 寄存器</p> <p>确定 FGPS_IBStatus 值为 0</p> <p>声明 FGPS_ExeMODE 为 1。</p> <p>（2）PSExeMode→PSHostMode：</p>	0b



	确定所有的像素着色器的 IsNotEmpty_PS 为 0 声明 FGPS_ExeMODE 为 0。	
--	---	--

FGPS_PCStart	位	描述	初始状态
Reserved	[31:9]	保留	0
FGPS_PCStart	[8:0]	当像素着色器开始运行,从指令存储器获取的第一个指令被储存在 PCStart 上。这个寄存器值可以复制到像素着色器程序计数器上，在着色器开始之前通过 FGPS_PCCopy 寄存器完成操作。这个值是在像素着色器开始时执行的第一个指令的地址。通过命令可以复制到像素着色器的程序计数器上。	0x0

FGPS_PCEnd	位	描述	初始状态
Reserved	[31:10]	保留	0
FGPS_IgnorePCEnd	[9]	当此位设置为 1，FGPS_PCEnd 值被忽略，只有程序计数器堆栈空条件可以用于程序终止。	
FGPS_PCEnd	[8:0]	当程序计数器达到 FGPS_PCEnd 的值时，着色器程序在寄存器执行完指令之后终止。这种方法可以保护指令槽和指令计数用于执行。  另一种终止像素着色器程序的方法：  像素着色器程序可以通过额外的“ret”指令终止，“ret”指令使程序计数栈为空条件。通常“call”和“ret”指令成对工作。但是故意不匹配的“ret”使顶点着色器终止。通过这个特例，顶点着色器程序可以被终止。	0x1FF

FGPS_PCCopy	位	描述	初始状态
Reserved	[31:1]	保留	0
FGPS_PCCopy	[0]	当此位设置为 1，FGVS_PCStart 寄存器值被复制到像素着色器里面。复制完成后，此位自动清除到 0。  在没有任何复制命令的时候，FGVS_PCStart 的值不被使用，用先前的值运行开始或结束地址。	0b

FGPS_AttributeNum	位	描述	初始状态
Reserved	[31:4]	保留	0
FGPS_AttributeNum	[3:0]	寄存器的值的范围在 1 到 8 之间，根据颜色和纹理的坐标值决定，这些光栅化引擎内的颜色和纹理坐标被转换到像素着色器输入寄存器内。  如果没有语义被转换到输入寄存器内，只有位置被转换到位置寄存器，FGPS_AttributeNum 应该设置为 0. 否则，这个寄存器被设置为被转换到像素着色器输入寄存器的语义序号。  如果像素着色器程序使用过了比转换到像素着色器输入寄存器内的语义更多的语义，像素着色器输出不可预测。	0x8

FGPS_IBStatus	位	描述	初始状态
Reserved	[31:1]	保留	0
FGPS_IBStatus	[0]	这是只读寄存器，用于镜像像素着色器输入缓冲区 IsNotReady  当 FGPS_AttributeNum 值改变，像素着色器输入缓冲区初始化序列开始。在输入缓冲区初始化期间，这个位置被设置为 1. 初始化之后自动清除为 0.	0b

## 42.8 纹理单元

### 42.8.1.概述

纹理单元（针对像素）

纹理单元包括地址产生逻辑，过滤单元，文本缓存和减压单元。顶点纹理单元包括地址产生单元和顶点纹理缓存。

纹理单元支持非  $2^N$  尺寸的纹理（任意矩形纹理）：

（1）达到 8 个交融纹理（最多 8 个纹理采样器）：每个纹理背景通过专用寄存器设置，每个纹理可以控制最少/最多 Mipmap 级别。

（2）纹理的最大宽度/高度是：2048x2048

- (3) 最大 Mipmap 级别是：12 级
- (4) Mipmap 级别的纹理尺寸
- (5) 纹理单元支持二维纹理，Cubemap,和 3D 纹理

顶点纹理单元（针对顶点）

不支持过滤。顶点纹理单元只用于获取 32 位纹理数据。

不支持压缩格式和色块纹理。

### 42.8.2. 纹理单元特殊寄存器

#### 42.8.1. 纹理状态寄存器

寄存器	地址	读/写	描述	复位值
FGTU_TSTA0	0x72060000	读/写	纹理 0 的状态	0x08000000
FGTU_TSTA1	0x72060050	读/写	纹理 1 的状态	0x08000000
FGTU_TSTA2	0x720600A0	读/写	纹理 2 的状态	0x08000000
FGTU_TSTA3	0x720600F0	读/写	纹理 3 的状态	0x08000000
FGTU_TSTA4	0x72060140	读/写	纹理 4 的状态	0x08000000
FGTU_TSTA5	0x72060190	读/写	纹理 5 的状态	0x08000000
FGTU_TSTA6	0x720601E0	读/写	纹理 6 的状态	0x08000000
FGTU_TSTA7	0x72060230	读/写	纹理 7 的状态	0x08000000

FGTU_TSTAn	位	描述	初始状态
Reserved	[31:29]	保留	000b
TYPE	[28:27]	纹理类型 00b=保留                      01b=二维可用 10b=Cube 可用              11b=3D 可用	01b
Reserved	[26:23]	保留	0000b

CK_SEL	[22:21]	颜色 密钥使能/选择 00b=不可用 01b=可用（用色键寄存器 1 或色键 YUV 寄存器） 10b=不可用 11b=可用（用色键寄存器 2 或色键 YUV 寄存器）	01b
TEX_EXP	[20]	纹理值扩展方法 0b=重复 LSB 1b=0 填充	0b
AFORMAT_SEL	[19]	Alpha 位置选择 0b=ARGB 1b=RGBA	0b
PAL_TEX_FORMAT	[18:17]	调色板内的 RGB 格式 00b=1555 01b=565 10b=4444 11b=8888	00b
TEXTURE_FORMATE	[16:12]	纹理格式 00000b=1555 00001b=565 00010b=4444 00011b=深度组成部分 16 00100b=88 00101b=8 00110b=8888 00111b=1BPP 01000b=2 BPP 01001b=4 BPP 01010b=8 BPP	00000b

		01011b=S3TC 01100b=YUV422, Y1VY0U 排序 01101b=YUV422, VY1UY0 排序 01110b=YUV422, Y1UV0Y 排序 01111b=YUV422, UY1VY0 排序 10000b~11111b=保留	
UADDR_MODE	[11:10]	U 地址内使用的模式 00b=重复 01b=倒装 10b=固定到边沿 11b=保留	00b
VADDR_MODE	[9:8]	V 地址内使用的模式 00b=重复 01b=倒装 10b=固定到边沿 11b=保留	00b
PADDR_MODE	[7:6]	P 地址内使用的模式 00b=重复 01b=倒装 10b=固定到边沿 11b=保留	00b
Reserved	[5]	保留	0b
TEX_COOR	[4]	纹理处理坐标系统 0b=参数 1b=非参数	0b
MAG_FILTER	[3]	双线性过滤控制 0b=映射中的像素之间不过滤 1b=映射中的像素之间选项行过滤	0b
MIPMAP_EN	[1:0]	MIPMAP 控制	00b

		00b=不使用	
		01b=使用 MIPMAP，选择最近映射	
		10b=使用 MIPMAP，	
		11b=保留	

## 42.8.2. 纹理 U 尺寸寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_USIZE0	0x72060004	读/写	纹理 0 的 U 尺寸	0x00000000
FGTU_USIZE1	0x72060054	读/写	纹理 1 的 U 尺寸	0x00000000
FGTU_USIZE2	0x720600A4	读/写	纹理 2 的 U 尺寸	0x00000000
FGTU_USIZE3	0x720600F4	读/写	纹理 3 的 U 尺寸	0x00000000
FGTU_USIZE4	0x72060144	读/写	纹理 4 的 U 尺寸	0x00000000
FGTU_USIZE5	0x72060194	读/写	纹理 5 的 U 尺寸	0x00000000
FGTU_USIZE6	0x720601E4	读/写	纹理 6 的 U 尺寸	0x00000000
FGTU_USIZE7	0x72060234	读/写	纹理 7 的 U 尺寸	0x00000000

FGTU_USIZE <sub>n</sub>	位	描述	初始状态
Reserved	[31:11]	保留	0
U_SIZE	[10:0]	0 级纹理的 U 尺寸	0x0

## 42.8.3. 纹理 V 尺寸寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_VSIZE0	0x72060008	读/写	纹理 0 的 V 尺寸	0x00000000
FGTU_VSIZE1	0x72060058	读/写	纹理 1 的 V 尺寸	0x00000000
FGTU_VSIZE2	0x720600A8	读/写	纹理 2 的 V 尺寸	0x00000000
FGTU_VSIZE3	0x720600F8	读/写	纹理 3 的 V 尺寸	0x00000000

FGTU_VSIZE4	0x72060148	读/写	纹理 4 的 V 尺寸	0x00000000
FGTU_VSIZE5	0x72060198	读/写	纹理 5 的 V 尺寸	0x00000000
FGTU_VSIZE6	0x720601E8	读/写	纹理 6 的 V 尺寸	0x00000000
FGTU_VSIZE7	0x72060238	读/写	纹理 7 的 V 尺寸	0x00000000

FGTU_VSIZE <sub>n</sub>	位	描述	初始状态
Reserved	[31:11]	保留	0
V_SIZE	[10:0]	0 级纹理的 V 尺寸	0x0

#### 42.8.4.纹理 P 尺寸寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_PSIZE0	0x7206000C	读/写	纹理 0 的 P 尺寸	0x00000000
FGTU_PSIZE1	0x7206005C	读/写	纹理 1 的 P 尺寸	0x00000000
FGTU_PSIZE2	0x720600AC	读/写	纹理 2 的 P 尺寸	0x00000000
FGTU_PSIZE3	0x720600FC	读/写	纹理 3 的 P 尺寸	0x00000000
FGTU_PSIZE4	0x7206014C	读/写	纹理 4 的 P 尺寸	0x00000000
FGTU_PSIZE5	0x7206019C	读/写	纹理 5 的 P 尺寸	0x00000000
FGTU_PSIZE6	0x720601EC	读/写	纹理 6 的 P 尺寸	0x00000000
FGTU_PSIZE7	0x7206023C	读/写	纹理 7 的 P 尺寸	0x00000000

FGTU_PSIZE <sub>n</sub>	位	描述	初始状态
Reserved	[31:11]	保留	0
P_SIZE	[10:0]	0 级纹理的 P 尺寸	0x0

42.8.5.纹理 L1 补偿寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_TOFFS_L1_0	0x72060010	读/写	纹理 0 的级别 1 的纹理补偿	0x00000000
FGTU_TOFFS_L1_1	0x72060060	读/写	纹理 1 的级别 1 的纹理补偿	0x00000000
FGTU_TOFFS_L1_2	0x720600B0	读/写	纹理 2 的级别 1 的纹理补偿	0x00000000
FGTU_TOFFS_L1_3	0x72060100	读/写	纹理 3 的级别 1 的纹理补偿	0x00000000
FGTU_TOFFS_L1_4	0x72060150	读/写	纹理 4 的级别 1 的纹理补偿	0x00000000
FGTU_TOFFS_L1_5	0x720601A0	读/写	纹理 5 的级别 1 的纹理补偿	0x00000000
FGTU_TOFFS_L1_6	0x720601F0	读/写	纹理 6 的级别 1 的纹理补偿	0x00000000
FGTU_TOFFS_L1_7	0x72060240	读/写	纹理 7 的级别 1 的纹理补偿	0x00000000

FGTU_TOFFS_L1n	位	描述	初始状态
Reserved	[31:23]	保留	0
OFFSET	[22:0]	1 级纹理补偿	0x0

42.8.6.纹理 L2 补偿寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_TOFFS_L2_0	0x72060014	读/写	纹理 0 的级别 2 的纹理补偿	0x00000000
FGTU_TOFFS_L2_1	0x72060064	读/写	纹理 1 的级别 2 的纹理补偿	0x00000000
FGTU_TOFFS_L2_2	0x720600B4	读/写	纹理 2 的级别 2 的纹理补偿	0x00000000
FGTU_TOFFS_L2_3	0x72060104	读/写	纹理 3 的级别 2 的纹理补偿	0x00000000
FGTU_TOFFS_L2_4	0x72060154	读/写	纹理 4 的级别 2 的纹理补偿	0x00000000
FGTU_TOFFS_L2_5	0x720601A4	读/写	纹理 5 的级别 2 的纹理补偿	0x00000000
FGTU_TOFFS_L2_6	0x720601F4	读/写	纹理 6 的级别 2 的纹理补偿	0x00000000
FGTU_TOFFS_L2_7	0x72060244	读/写	纹理 7 的级别 2 的纹理补偿	0x00000000



FGTU_TOFFS_L2n	位	描述	初始状态
Reserved	[31:23]	保留	0
OFFSET	[22:0]	2 级纹理补偿	0x0

## 42.8.7.纹理 L3 补偿寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_TOFFS_L3_0	0x72060018	读/写	纹理 0 的级别 3 的纹理补偿	0x00000000
FGTU_TOFFS_L3_1	0x72060068	读/写	纹理 1 的级别 3 的纹理补偿	0x00000000
FGTU_TOFFS_L3_2	0x720600B8	读/写	纹理 2 的级别 3 的纹理补偿	0x00000000
FGTU_TOFFS_L3_3	0x72060108	读/写	纹理 3 的级别 3 的纹理补偿	0x00000000
FGTU_TOFFS_L3_4	0x72060158	读/写	纹理 4 的级别 3 的纹理补偿	0x00000000
FGTU_TOFFS_L3_5	0x720601A8	读/写	纹理 5 的级别 3 的纹理补偿	0x00000000
FGTU_TOFFS_L3_6	0x720601F8	读/写	纹理 6 的级别 3 的纹理补偿	0x00000000
FGTU_TOFFS_L3_7	0x72060248	读/写	纹理 7 的级别 3 的纹理补偿	0x00000000

FGTU_TOFFS_L3n	位	描述	初始状态
Reserved	[31:23]	保留	0
OFFSET	[22:0]	3 级纹理补偿	0x0

## 42.8.8. 纹理 L4 补偿寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_TOFFS_L4_0	0x7206001C	读/写	纹理 0 的级别 4 的纹理补偿	0x00000000
FGTU_TOFFS_L4_1	0x7206006C	读/写	纹理 1 的级别 4 的纹理补偿	0x00000000
FGTU_TOFFS_L4_2	0x720600BC	读/写	纹理 2 的级别 4 的纹理补偿	0x00000000
FGTU_TOFFS_L4_3	0x7206010C	读/写	纹理 3 的级别 4 的纹理补偿	0x00000000
FGTU_TOFFS_L4_4	0x7206015C	读/写	纹理 4 的级别 4 的纹理补偿	0x00000000

FGTU_TOFFS_L4_5	0x720601AC	读/写	纹理 5 的级别 4 的纹理补偿	0x00000000
FGTU_TOFFS_L4_6	0x720601FC	读/写	纹理 6 的级别 4 的纹理补偿	0x00000000
FGTU_TOFFS_L4_7	0x7206024C	读/写	纹理 7 的级别 4 的纹理补偿	0x00000000

FGTU_TOFFS_L4n	位	描述	初始状态
Reserved	[31:23]	保留	0
OFFSET	[22:0]	4 级纹理补偿	0x0

#### 42.8.9.纹理 L5 补偿寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_TOFFS_L5_0	0x72060020	读/写	纹理 0 的级别 5 的纹理补偿	0x00000000
FGTU_TOFFS_L5_1	0x72060070	读/写	纹理 1 的级别 5 的纹理补偿	0x00000000
FGTU_TOFFS_L5_2	0x720600C0	读/写	纹理 2 的级别 5 的纹理补偿	0x00000000
FGTU_TOFFS_L5_3	0x72060110	读/写	纹理 3 的级别 5 的纹理补偿	0x00000000
FGTU_TOFFS_L5_4	0x72060160	读/写	纹理 4 的级别 5 的纹理补偿	0x00000000
FGTU_TOFFS_L5_5	0x720601B0	读/写	纹理 5 的级别 5 的纹理补偿	0x00000000
FGTU_TOFFS_L5_6	0x72060200	读/写	纹理 6 的级别 5 的纹理补偿	0x00000000
FGTU_TOFFS_L5_7	0x72060250	读/写	纹理 7 的级别 5 的纹理补偿	0x00000000

FGTU_TOFFS_L5n	位	描述	初始状态
Reserved	[31:23]	保留	0
OFFSET	[22:0]	5 级纹理补偿	0x0

#### 42.8.10. 纹理 L6 补偿寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_TOFFS_L6_0	0x72060024	读/写	纹理 0 的级别 6 的纹理补偿	0x00000000

FGTU_TOFFS_L6_1	0x72060074	读/写	纹理 1 的级别 6 的纹理补偿	0x00000000
FGTU_TOFFS_L6_2	0x720600C4	读/写	纹理 2 的级别 6 的纹理补偿	0x00000000
FGTU_TOFFS_L6_3	0x72060114	读/写	纹理 3 的级别 6 的纹理补偿	0x00000000
FGTU_TOFFS_L6_4	0x72060164	读/写	纹理 4 的级别 6 的纹理补偿	0x00000000
FGTU_TOFFS_L6_5	0x720601B4	读/写	纹理 5 的级别 6 的纹理补偿	0x00000000
FGTU_TOFFS_L6_6	0x72060204	读/写	纹理 6 的级别 6 的纹理补偿	0x00000000
FGTU_TOFFS_L6_7	0x72060254	读/写	纹理 7 的级别 6 的纹理补偿	0x00000000

FGTU_TOFFS_L6n	位	描述	初始状态
Reserved	[31:23]	保留	0
OFFSET	[22:0]	6 级纹理补偿	0x0

#### 42.8.11. 纹理 L7 补偿寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_TOFFS_L7_0	0x72060028	读/写	纹理 0 的级别 7 的纹理补偿	0x00000000
FGTU_TOFFS_L7_1	0x72060078	读/写	纹理 1 的级别 7 的纹理补偿	0x00000000
FGTU_TOFFS_L7_2	0x720600C8	读/写	纹理 2 的级别 7 的纹理补偿	0x00000000
FGTU_TOFFS_L7_3	0x72060118	读/写	纹理 3 的级别 7 的纹理补偿	0x00000000
FGTU_TOFFS_L7_4	0x72060168	读/写	纹理 4 的级别 7 的纹理补偿	0x00000000
FGTU_TOFFS_L7_5	0x720601B8	读/写	纹理 5 的级别 7 的纹理补偿	0x00000000
FGTU_TOFFS_L7_6	0x72060208	读/写	纹理 6 的级别 7 的纹理补偿	0x00000000
FGTU_TOFFS_L7_7	0x72060258	读/写	纹理 7 的级别 7 的纹理补偿	0x00000000

FGTU_TOFFS_L7n	位	描述	初始状态
Reserved	[31:23]	保留	0
OFFSET	[22:0]	7 级纹理补偿	0x0

42.8.12. 纹理 L8 补偿寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_TOFFS_L8_0	0x7206002C	读/写	纹理 0 的级别 8 的纹理补偿	0x00000000
FGTU_TOFFS_L8_1	0x7206007C	读/写	纹理 1 的级别 8 的纹理补偿	0x00000000
FGTU_TOFFS_L8_2	0x720600CC	读/写	纹理 2 的级别 8 的纹理补偿	0x00000000
FGTU_TOFFS_L8_3	0x7206011C	读/写	纹理 3 的级别 8 的纹理补偿	0x00000000
FGTU_TOFFS_L8_4	0x7206016C	读/写	纹理 4 的级别 8 的纹理补偿	0x00000000
FGTU_TOFFS_L8_5	0x720601BC	读/写	纹理 5 的级别 8 的纹理补偿	0x00000000
FGTU_TOFFS_L8_6	0x7206020C	读/写	纹理 6 的级别 8 的纹理补偿	0x00000000
FGTU_TOFFS_L8_7	0x7206025C	读/写	纹理 7 的级别 8 的纹理补偿	0x00000000

FGTU_TOFFS_L8n	位	描述	初始状态
Reserved	[31:23]	保留	0
OFFSET	[22:0]	8 级纹理补偿	0x0

42.8.13.纹理 L9 补偿寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_TOFFS_L9_0	0x72060030	读/写	纹理 0 的级别 9 的纹理补偿	0x00000000
FGTU_TOFFS_L9_1	0x72060080	读/写	纹理 1 的级别 9 的纹理补偿	0x00000000
FGTU_TOFFS_L9_2	0x720600D0	读/写	纹理 2 的级别 9 的纹理补偿	0x00000000
FGTU_TOFFS_L9_3	0x72060120	读/写	纹理 3 的级别 9 的纹理补偿	0x00000000
FGTU_TOFFS_L9_4	0x72060170	读/写	纹理 4 的级别 9 的纹理补偿	0x00000000
FGTU_TOFFS_L9_5	0x720601C0	读/写	纹理 5 的级别 9 的纹理补偿	0x00000000
FGTU_TOFFS_L9_6	0x72060210	读/写	纹理 6 的级别 9 的纹理补偿	0x00000000
FGTU_TOFFS_L9_7	0x72060260	读/写	纹理 7 的级别 9 的纹理补偿	0x00000000

FGTU_TOFFS_L9n	位	描述	初始状态
Reserved	[31:23]	保留	0
OFFSET	[22:0]	9 级纹理补偿	0x0

#### 42.8. 14. 纹理 L10 补偿寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_TOFFS_L10_0	0x72060034	读/写	纹理 0 的级别 10 的纹理补偿	0x00000000
FGTU_TOFFS_L10_1	0x72060084	读/写	纹理 1 的级别 10 的纹理补偿	0x00000000
FGTU_TOFFS_L10_2	0x720600D4	读/写	纹理 2 的级别 10 的纹理补偿	0x00000000
FGTU_TOFFS_L10_3	0x72060124	读/写	纹理 3 的级别 10 的纹理补偿	0x00000000
FGTU_TOFFS_L10_4	0x72060174	读/写	纹理 4 的级别 10 的纹理补偿	0x00000000
FGTU_TOFFS_L10_5	0x720601C4	读/写	纹理 5 的级别 10 的纹理补偿	0x00000000
FGTU_TOFFS_L10_6	0x72060214	读/写	纹理 6 的级别 10 的纹理补偿	0x00000000
FGTU_TOFFS_L10_7	0x72060264	读/写	纹理 7 的级别 10 的纹理补偿	0x00000000

FGTU_TOFFS_L10n	位	描述	初始状态
Reserved	[31:23]	保留	0
OFFSET	[22:0]	10 级纹理补偿	0x0

#### 42.8.15. 纹理 L11 补偿寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_TOFFS_L11_0	0x72060034	读/写	纹理 0 的级别 11 的纹理补偿	0x00000000
FGTU_TOFFS_L11_1	0x72060084	读/写	纹理 1 的级别 11 的纹理补偿	0x00000000

FGTU_TOFFS_L11_2	0x720600D4	读/写	纹理 2 的级别 11 的纹理补偿	0x00000000
FGTU_TOFFS_L11_3	0x72060124	读/写	纹理 3 的级别 11 的纹理补偿	0x00000000
FGTU_TOFFS_L11_4	0x72060174	读/写	纹理 4 的级别 11 的纹理补偿	0x00000000
FGTU_TOFFS_L11_5	0x720601C4	读/写	纹理 5 的级别 11 的纹理补偿	0x00000000
FGTU_TOFFS_L11_6	0x72060214	读/写	纹理 6 的级别 11 的纹理补偿	0x00000000
FGTU_TOFFS_L11_7	0x72060264	读/写	纹理 7 的级别 11 的纹理补偿	0x00000000

FGTU_TOFFS_L11n	位	描述	初始状态
Reserved	[31:23]	保留	0
OFFSET	[22:0]	11 级纹理补偿	0x0

## 42.8.16. 纹理最小级寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_T_MIN_L0	0x7206003C	读/写	纹理 0 的 Mipmap 最小级	0x00000000
FGTU_T_MIN_L1	0x7206008C	读/写	纹理 1 的 Mipmap 最小级	0x00000000
FGTU_T_MIN_L2	0x720600DC	读/写	纹理 2 的 Mipmap 最小级	0x00000000
FGTU_T_MIN_L3	0x7206012C	读/写	纹理 3 的 Mipmap 最小级	0x00000000
FGTU_T_MIN_L4	0x7206017C	读/写	纹理 4 的 Mipmap 最小级	0x00000000
FGTU_T_MIN_L5	0x720601CC	读/写	纹理 5 的 Mipmap 最小级	0x00000000
FGTU_T_MIN_L6	0x7206021C	读/写	纹理 6 的 Mipmap 最小级	0x00000000
FGTU_T_MIN_L7	0x7206026C	读/写	纹理 7 的 Mipmap 最小级	0x00000000

FGTU_T_MIN_Ln	位	描述	初始状态
Reserved	[31:4]	保留	0
MIN_LEVEL	[3:0]	纹理的 Mipmap 最小级	0x0

42.8.17. 纹理最大级寄存器 0~7

寄存器	地址	读/写	描述	复位值
FGTU_T_MAX_L0	0x72060040	读/写	纹理 0 的 Mipmap 最大级	0x00000000
FGTU_T_MAX_L1	0x72060090	读/写	纹理 1 的 Mipmap 最大级	0x00000000
FGTU_T_MAX_L2	0x720600E0	读/写	纹理 2 的 Mipmap 最大级	0x00000000
FGTU_T_MAX_L3	0x72060130	读/写	纹理 3 的 Mipmap 最大级	0x00000000
FGTU_T_MAX_L4	0x72060180	读/写	纹理 4 的 Mipmap 最大级	0x00000000
FGTU_T_MAX_L5	0x720601D0	读/写	纹理 5 的 Mipmap 最大级	0x00000000
FGTU_T_MAX_L6	0x72060220	读/写	纹理 6 的 Mipmap 最大级	0x00000000
FGTU_T_MAX_L7	0x72060270	读/写	纹理 7 的 Mipmap 最大级	0x00000000

FGTU_T_MAX_Ln	位	描述	初始状态
Reserved	[31:4]	保留	0
MIN_LEVEL	[3:0]	纹理的 Mipmap 最大级	0x0

42.8.18. 纹理基础地址寄存器 0~7

	地址	读/写	描述	复位值
FGTU_TBADD0	0x72060044	读/写	纹理 0 的基础地址	0x00000000
FGTU_TBADD1	0x72060094	读/写	纹理 1 的基础地址	0x00000000
FGTU_TBADD2	0x720600E4	读/写	纹理 2 的基础地址	0x00000000
FGTU_TBADD3	0x72060134	读/写	纹理 3 的基础地址	0x00000000
FGTU_TBADD4	0x72060184	读/写	纹理 4 的基础地址	0x00000000
FGTU_TBADD5	0x720601D4	读/写	纹理 5 的基础地址	0x00000000
FGTU_TBADD6	0x72060224	读/写	纹理 6 的基础地址	0x00000000
FGTU_TBADD7	0x72060274	读/写	纹理 7 的基础地址	0x00000000

FGTU_TBADDn	位	描述	初始状态
ADDR	[31:0]	0 级纹理的基础地址	0xFFFFFFFF

### 42.8.19.纹理色键寄存器

寄存器	地址	读/写	描述	复位值
FGTU_CKEY1	0x72060280	读/写	3D 色键 1 寄存器	0x00000000
FGTU_CKEY2	0x72060284	读/写	3D 色键 2 寄存器	0x00000000
FGTU_CKEYUV	0x72060288	读/写	3D 色键 YUV 寄存器	0x00000000
FGTU_CKMASK	0x7206028C	读/写	3D 色键屏蔽寄存器	0x00000000
FGTU_PALLETTE_ADDR	0x72060290	写	索引纹理调色板地址	0x00000000
FGTU_PALLETTE_IN	0x72060294	写	索引纹理调色板地址	0x00000000

FGTU_CKEY1	位	描述	初始状态
Reserved	[31:24]	保留	
R	[23:16]	色键红色值 不是 YUV, CK_SEL=01	0x0
G	[15:8]	色键绿色值 不是 YUV, CK_SEL=01	0x0
B	[7:0]	色键蓝色值 不是 YUV, CK_SEL=01	0x0

FGTU_CKEY2	位	描述	初始状态
Reserved	[31:24]	保留	
R	[23:16]	色键红色值 不是 YUV, CK_SEL=11	0x0
G	[15:8]	色键绿色值	0x0



		不是 YUV, CK_SEL=11	
B	[23:16]	色键蓝色值 不是 YUV, CK_SEL=11	0x0

FGTU_CKYUV	位	描述	初始状态
Reserved	[31:16]	保留	
VALU	[15:8]	色键 U 值 YUV, CK_SEL=01	0x0
VALV	[23:16]	色键 V 值 YUV, CK_SEL=01	0x0

FGTU_CKMASK	位	描述	初始状态
Reserved	[31:3]	保留	
VAL	[2:0]	000b=不屏蔽色键位  001b=屏蔽每个 CK 颜色组成部分的 1 个最低有效位 010b=屏蔽每个 CK 颜色组成部分的 2 个最低有效位 011b=屏蔽每个 CK 颜色组成部分的 3 个最低有效位 100b=屏蔽每个 CK 颜色组成部分的 4 个最低有效位 101b=屏蔽每个 CK 颜色组成部分的 5 个最低有效位 110b=屏蔽每个 CK 颜色组成部分的 6 个最低有效位 1111b=屏蔽每个 CK 颜色组成部分的 7 个最低有效位	000b

FGTU_PALLETTE_ADDR	位	描述	初始状态
Reserved	[31:8]	保留	0
ADDR	[7:0]	调色板地址	0x0

FGTU_PALLETTE_IN	位	描述	初始状态
DATA	[31:0]	调色板数据输入	0x0

42.8.20.顶点纹理状态寄存器

寄存器	地址	读/写	描述	复位值
FGVTU_VTSTA0	0x720602C0	读/写	顶点纹理 0 的状态	0x00000000
FGVTU_VTSTA1	0x720602C8	读/写	顶点纹理 1 的状态	0x00000000
FGVTU_VTSTA2	0x720602D0	读/写	顶点纹理 2 的状态	0x00000000
FGVTU_VTSTA3	0x720602D8	读/写	顶点纹理 3 的状态	0x00000000

FGVTU_VTSTAn	位	描述	初始状态
DATA	[31:12]	保留	
UMOD	[11:10]	U 地址内用使用的模式 00b=重复                      01b=倒装 11b=固定到边沿            11b=保留	00b
VMOD	[9:8]	V 地址内用使用的模式 00b=重复                      01b=倒装 11b=固定到边沿            11b=保留	00b
USIZE	[7:4]	纹理 U 尺寸 0000b=1 个像素            0001b=2 个像素 0010b=4 个像素            0011b=8 个像素 0100b=16 个像素           0101b=32 个像素 0110b=64 个像素           0111b=128 个像素 1000b=256 个像素          1001b=512 个像素 1010b=1024 个像素        1011b=2048 个像素 1100b~111b=保留	0x0
VSIZE	[3:0]	纹理 V 尺寸 0000b=1 个像素            0001b=2 个像素 0010b=4 个像素            0011b=8 个像素 0100b=16 个像素           0101b=32 个像素	0x0

		0110b=64 个像素    0111b=128 个像素	
		1000b=256 个像素    1001b=512 个像素	
		1010b=1024 个像素    1011b=2048 个像素	
		1100b~1111b=保留	

### 42.8.21.顶点纹理基础地质寄存器

寄存器	地址	读/写	描述	复位值
FGVTU_VTBADDR0	0x720602C4	读/写	顶点纹理 0 的基础地址	0x00000000
FGVTU_VTBADDR1	0x720602CC	读/写	顶点纹理 1 的基础地址	0x00000000
FGVTU_VTBADDR2	0x720602D4	读/写	顶点纹理 2 的基础地址	0x00000000
FGVTU_VTBADDR3	0x720602DC	读/写	顶点纹理 3 的基础地址	0x00000000

FGVTU_VTBADDRn	位	描述	初始状态
ADDR	[31:0]	顶点纹理基础地址	0xFFFFFFFF

## 42.9 预片元单元

### 42.9.1.概述

支持所有的 OpenGL2.0 预片元操作。支持深度缓冲区和模版缓冲区，深度缓冲区位的深度是 24 位，模版缓冲区位的深度是 8 位。

- (1) 预片元单元支持 Scissor 测试
- (2) 预片元单元支持 Alpha 测试
- (3) 预片元单元支持模版测试和模版操作。同时支持前模版缓冲区和后模版还充区。
- (4) 预片元单元支持深度测试（深度缓冲区是 24 位）
- (5) 预片元单元支持 Alpha 混合
- (6) 预片元单元支持逻辑操作

- (7) 预片元单元支持 16/32 位颜色模式
- (8) 增强型预片元单元支持抖动。

像素所有权测试决定通过重叠窗口是否能看到目标像素。图 42-13 是预片元单元的功能模块图。

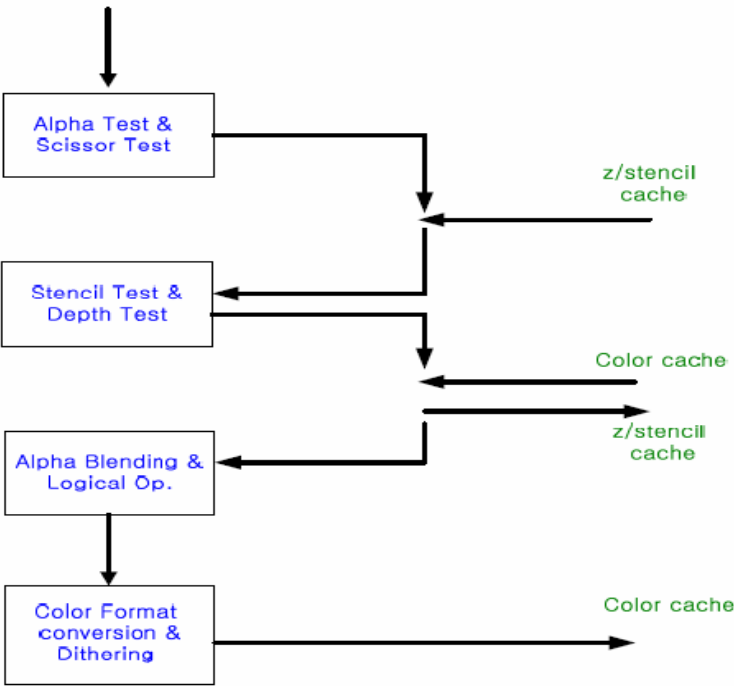


图 42-13 预片元单元的功能模块图

## 42.9.2.预片元单元特殊寄存器

### 42.9.2.1.SCISSOER 测试控制寄存器

如果想丢弃所有的片元，需要进行以下设置：XMin=0，XMax=0，YMax=0.如果想通过所有片元，需要进行以下设置：XMin=0，XMax=最大屏幕 X 宽度，YMin=0，YMax=最大屏幕 Y 高度。

寄存器	地址	读/写	描述	复位值
FGPF_SCISSOR_X	0x72070000	读/写	X 坐标像素裁剪区域 (X 坐标的范围从 0 到 2047)	0x00000000

FGPF_SCISSOR_Y	0x72070004	读/写	Y 坐标像素裁剪区域 (Y 坐标的范围从 0 到 2047)	0x00000000
----------------	------------	-----	-----------------------------------	------------

FGPF_SCISSOR_X	位	描述	初始状态
ScissorTestEnable	[31]	Scissor 测试使能位 0b=使不能 1b=使能	0b
Reserved	[30:28]	保留	0
XMax	[27:16]	像素的 X 坐标 $\geq$ 最大值，没有写入帧缓冲区	0x0
Reserved	[15:12]	保留	0
XMin	[11:0]	像素的 X 坐标 $<$ 最小值，没有写入帧缓冲区	0x0

FGPF_SCISSOR_X	位	描述	初始状态
Reserved	[31:28]	保留	0
YMax	[27:16]	像素的 Y 坐标 $\geq$ 最大值，没有写入帧缓冲区	0x0
Reserved	[15:12]	保留	0
YMin	[11:0]	像素的 Y 坐标 $<$ 最小值，没有写入帧缓冲区	0x0

#### 42.9.2.2.Alpha 测试控制寄存器

寄存器	地址	读/写	描述	复位值
FGPF_ALPHAT	0x72070008	读/写	Alpha 测试控制寄存器	0x00000000

FGPF_ALPHAT	位	描述	初始状态
Reserved	[31:12]	保留	0
AlphaTestValue	[11:4]	8 位 Alpha 测试参考值	0x0
AlphaTestMode	[3:1]	Alpha 测试使用模式 000b=NEVER                      001b=ALWAYS	000b

		010b=LESS                      011b=LEQUAL 100b=EQUAL                      101b=GREATER 110b=GEQUAL                      111b=NOTEQUAL	
AlphaTestEnable	[0]	Alpha 测试使能位 0b=不能进行 Alpha 测试 1b=能进行 Alpha 测试	0b

### 42.9.2.3 模版测试控制寄存器

寄存器	地址	读/写	描述	复位值
FGPF_FRONTST	0x7207000C	读/写	前面模版测试控制寄存器	0x00000000
FGPF_BACKST	0x72070010	读/写	背面模版测试控制寄存器	0x00000000

FGPF_FRONTST	位	描述	初始状态
FrontStencil_dppass	[31:29]	模版深度缓冲区通过行动 000b=KEEP                      001b=ZERO 010b=REPLACE                      011b=INCR 100b==DECR                      101b=INVERT 110b=INCR_WRAP                      111b=DECR_WRAP	000b
FrontStencil_dpfail	[28:26]	模版深度缓冲区失败行动 同上	000b
FrontStencil_sfail	[25:23]	模版失败行动 同上	000b
Reserved	[22:20]	保留	
FrontStencilMaskValue	[19:12]	8 位模版屏蔽值	0x0
FrontStencilTestValue	[11:4]	8 位模版参考值	0x0
FrontStencilTestMode	[3:1]	模版测试使用模式 000b=NEVER                      001b=ALWAYS 010b=LESS                      011b=LEQUAL	000b

		100b=EQUAL                      101b=GREATER 110b=GEQUAL                      111b=NOTEQUAL	
StencilTestEnable	[0]	模版测试使能位  0b=不能进行模版测试  1b=能进行模版测试	0b

FGPF_BACKST	位	描述	初始状态
BackStencil_dppass	[31:29]	模版深度缓冲区通过行动  000b=KEEP                      001b=ZERO 010b=REPLACE                      011b=INCR 100b==DECR                      101b=INVERT 110b=INCR_WRAP                      111b=DECR_WRAP	000b
BackStencil_dpfail	[28:26]	模版深度缓冲区失败行动 同上	000b
BackStencil_sfail	[25:23]	模版失败行动 同上	000b
Reserved	[22:20]	保留	
BackStencilMaskValue	[19:12]	8 位模版屏蔽值	0x0
BackStencilTestValue	[11:4]	8 位模版参考值	0x0
BackStencilTestMode	[3:1]	模版测试使用模式  000b=NEVER                      001b=ALWAYS 010b=LESS                      011b=LEQUAL 100b=EQUAL                      101b=GREATER 110b=GEQUAL                      111b=NOTEQUAL	000b
Reserved	[0]	保留	0

42.9.2.4 深度测试控制寄存器

寄存器	地址	读/写	描述	复位值
FGPF_DEPTH	0x72070014	读/写	深度测试控制寄存器	0x00000002

FGPF_DEPTH	位	描述	初始状态
Reserved	[31:4]	保留	
DepthTestMode	[3:1]	深度测试使用模式  000b=NEVER                001b=ALWAYS 010b=LESS                011b=LEQUAL 100b=EQUAL               101b=GREATER 110b=GEQUAL             111b=NOTEQUAL	000b
DepthTestEnable	[0]	深度测试使能位  0b=不能进行深度缓冲区测试 1b=能进行深度缓冲区测试	0b

42.9.2.5 混合控制寄存器

寄存器	地址	读/写	描述	复位值
FGPF_CCLR	0x72070018	读/写	混合常量颜色	0x00000000
FGPF_BLEND	0x7207001C	读/写	混合控制寄存器	0x00000000

FGPF_CCLR	位	描述	初始状态
VAL	[31:0]	混合常量 RGBA 颜色	0x0

FGPF_BLEND	位	描述	初始状态
Reserved	[31:23]	保留	0
ABlendEquation	[22:20]	Alpha 混合方程式	000b



		000b=Add 001b=Subtract 010b= Reverse Subtract 011b=Min 100b=Max	
BlendEquation	[19:17]	混合方程式 000b=Add 001b=Subtract 010b= Reverse Subtract 011b=Min 100b=Max	000b
AblendDstBlendFunc	[16:13]	混合目标函数使用的模式 0000b=ZERO 0001b=ONE 0010b= SRC_COLOR 0011b=ONE_MINUS_SRC_COLOR 0100b=DST_COLOR 0101b=ONE_MINUS_DST_COLOR 0110b=SRC_ALPHA 0111b=ONE_MINUS_SRC_ALPHA 1000b=DST_ALPHA 1001b= ONE_MINUS_ DST_ALPHA 1010b=CONSTANT_COLOR 1011b= ONE_MINUS_ CONSTANT_COLOR 1100b=CONSTANT_ ALPHA 1101b= ONE_MINUS_ CONSTANT_ ALPHA 1110b=SRC_ALPHA_SATURATE	0x0
ColorDstBlendFunc	[12:9]	混合目标函数使用的模式 同上	0x0

AlphaSrcBlendFunc	[8:5]	混合源函数使用的模式 同上	0x0
ColorSrcBlendFunc	[4:1]	混合目标函数使用的模式 同上	0x0
BlendingEnable	[0]	0b=不能混合 1b=能混合	0b

### 42.9.2.6 逻辑操作控制寄存器

寄存器	地址	读/写	描述	复位值
FGPF_LOGOP	0x72070020	读/写	RGBA 颜色逻辑运行使能&0 功能	0x00000000

FGPF_LOGOP	位	描述	初始状态																										
Reserved	[31:9]	保留	0																										
AlphaLogOpEnable	[8:5]	<div>逻辑操作内使用的模式</div> <table><thead><tr><th>参数值</th><th>操作</th></tr></thead><tbody><tr><td>0000</td><td>CLEAR</td></tr><tr><td>0001</td><td>AND s&amp;d</td></tr><tr><td>0010</td><td>AND_REVERSE s&amp;~d</td></tr><tr><td>0011</td><td>COPY s</td></tr><tr><td>0100</td><td>AND_INVERTED ~s&amp;d</td></tr><tr><td>0101</td><td>NOOP d</td></tr><tr><td>0110</td><td>XOR s xor d</td></tr><tr><td>0111</td><td>OR s   d</td></tr><tr><td>1000</td><td>NOR ~(s   d)</td></tr><tr><td>1001</td><td>EQUIV ~(s xor d)</td></tr><tr><td>1010</td><td>INVERT ~d</td></tr><tr><td>1011</td><td>OR_REVERSE s   ~d</td></tr></tbody></table>	参数值	操作	0000	CLEAR	0001	AND s&d	0010	AND_REVERSE s&~d	0011	COPY s	0100	AND_INVERTED ~s&d	0101	NOOP d	0110	XOR s xor d	0111	OR s   d	1000	NOR ~(s   d)	1001	EQUIV ~(s xor d)	1010	INVERT ~d	1011	OR_REVERSE s   ~d	0x0
参数值	操作																												
0000	CLEAR																												
0001	AND s&d																												
0010	AND_REVERSE s&~d																												
0011	COPY s																												
0100	AND_INVERTED ~s&d																												
0101	NOOP d																												
0110	XOR s xor d																												
0111	OR s   d																												
1000	NOR ~(s   d)																												
1001	EQUIV ~(s xor d)																												
1010	INVERT ~d																												
1011	OR_REVERSE s   ~d																												

		1100 COPY_INVERTED $\sim s$	
		1101 OR_INVERTED $\sim s \mid d$	
		1110 NAND $\sim (s \& d)$	
		1111 SET all 1's	
ColorLogOpEnable	[4:1]	颜色逻辑操作内使用的模式。同上	0x0
LogOpEnable	[0]	0b=不能进行颜色逻辑操作 1b=可以进行颜色逻辑操作	0b

### 42.9.2.7 颜色缓冲区写屏蔽寄存器

寄存器	地址	读/写	描述	复位值
FGPF_CBMSK	0x72070024	读/写	RGBA 模式内颜色写屏蔽	0x00000000

FGPF_CBMSK	位	描述	初始状态
Reserved	[31:4]	保留	
FbColorWrMask	[3:0]	此寄存器用于屏蔽向颜色缓冲区内写入 R, G, B 和 A 值。r, g, b 和 a 用于表示是否写入 R, G, B, A 值。初始状态下，可以写入所有的位和所有颜色值。  0000b=所有屏蔽无效  0001b=a 屏蔽有效  0010b=b 屏蔽有效  0100b= g 屏蔽使能  1000b=r 屏蔽使能	0x0

42.9.2. 8 深度/模版缓冲区写屏蔽寄存器

寄存器	地址	读/写	描述	复位值
FGPF_DBMSK	0x72070028	读/写	深度/模版写屏蔽	0x00000000

FGPF_DBMSK	位	描述	初始状态
Back_FbStencilWrMask	[31:24]	<p>这个寄存器用于模版缓冲区写屏蔽。专针对于背面的像素。</p> <p>可以控制模版缓冲区写位能否工作。在原始状态下，当模版测试打开时，模版缓冲区可以写入数据。</p> <p>模版缓冲区每个像素有 8 位，这个寄存器的每个位可以屏蔽模版缓冲区对应的值。</p> <p>如：</p> <p>00000000b，在这种情况下，模版缓冲区内所有的 8 个位都更新。</p> <p>00000001b，在这种情况下，模版缓冲区内除了最低有效位 1 以外所有的位被更新。</p> <p>00000011b，在这种情况下，除了最低有效位 2 以外所有的位被更新。</p>	0x0
Front_FbStencilWrMask	[23:16]	<p>这个寄存器用于模版缓冲区写屏蔽。专针对于前面的像素。</p> <p>可以控制模版缓冲区写位能否工作。在原始状态下，当模版测试打开时，模版缓冲区可以写入数据。</p> <p>模版缓冲区每个像素有 8 位，这个寄存器的每个位可以屏蔽模版缓冲区对应的值。</p>	0x0
Reserved	[15:1]	保留	0

FbDepthWrMask	[0]	可以控制深度缓冲区能或不能写入深度值。 在初始状态下，深度缓冲区可以写入深度值。 0b=深度缓冲区写入 01=不写入深度缓冲区	0b
---------------	-----	--	----

### 42.9.2.9 帧缓冲区控制寄存器

寄存器	地址	读/写	描述	复位值
FGPF_FBCTL	0x7207002C	读/写	帧缓冲区写控制寄存器	0x00000000

FGPF_FBCTL	位	描述	初始状态
Reserved	[31:21]	保留	0
OpaqueAlpha	[20]	1: Alpha 混合以后，迫使 Alpha 值到 opaque 0: 常规操作	0b
AlphaThreshold	[19:12]	当编码 16 位 1555 格式时使用。 如果（内部 Alpha 值 > Alpha 阈值） Alpha=1; 否则 Alpha=0;	0x0h
AlphaConst	[11:4]	常量 Alpha 值	0x0h
DitherOn	[3]	控制像素从 ARGB8888 格式转换到 16 位输出 像素 0b=抖动无效     1b=抖动有效	0b
ColorMode	[2:0]	帧缓冲区颜色使用的模式 000b=555, RGB, 16 位 001b=565, RGB, 16 位 010b=4444, RGB, 16 位 011b=1555, ARGB, 16 位 100b=0888, RGB, 32 位	000b

		101b=0888, ARGB, 32 位 110~111b=保留	
--	--	--------------------------------------	--

### 42.9.2.10 深度缓冲区基础地址寄存器

寄存器	地址	读/写	描述	复位值
FGPF_DBADDR	0x72070030	读/写	深度缓冲区补偿地址	0x00000000

FGPF_DBADDR	位	描述	初始状态
FbDepthOffset	[31:0]	深度缓冲区补偿地址	0x00000000

### 42.9.2.11 颜色缓冲区基础地址寄存器

寄存器	地址	读/写	描述	复位值
FGPF_CBADDR	0x72070034	读/写	颜色缓冲区补偿地址	0x00000000

FGPF_CBADDR	位	描述	初始状态
FbColorOffset	[31:0]	颜色缓冲区补偿地址	00000000h

### 42.9.2.12 帧缓冲宽度寄存器

寄存器	地址	读/写	描述	复位值
FGPF_FBW	0x72070038	读/写	帧缓冲区宽度	0x000007FF

FGPF_FBW	位	描述	初始状态
Reserved	[31:11]	保留	0
FbWidth	[10:0]	帧缓冲区宽度 (0~2048)	800h

## 42.10 AXI 判别器 & AXI DMA

### 42.10.1.概述

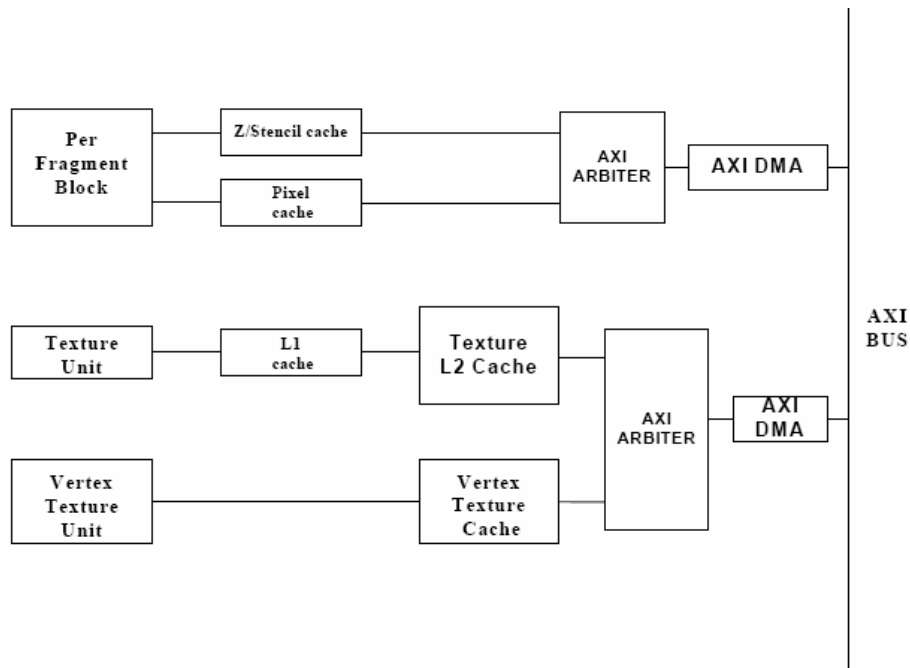


图 42-14 AXI 总线结构

#### (1) AXI 判别器性能

- 1)  $2 \times 1$  判别器
- 2) 0 等待判别
- 3) AXI DMA 接口
- 4) IP 核心接口

#### (2) AXI DMA 性能

- 1) AXI 总线接口：可分为 ADDR/CRTL, DATA 通道；可分为读、写通道；支持 64 位数据总线；支持各种突发长度；支持写字节屏蔽；支持各种突发类型；支持低电源通道
- 2) IP 黑心接口

#### (3) AXI 判别器 0 接口

- 1)接口在 AXI 判别器和 DMA 之间，是 FIFO 借口
- 2) 分为读地址通道和写地址通道
- 3) 分为写地址通道和写数据通道
- 4) 突发长度固定为 INCR8 或 WRAP8
- 5) 数据转换尺寸固定为两个字（64 位）

(4) AXI 判别器 1 接口

- 1)接口在 AXI 判别器和 DMA 之间，是 FIFO 借口
- 2) 分为读地址通道和写地址通道
- 3) 突发长度固定为 INCR8 或 WRAP8
- 4) 数据转换尺寸固定为两个字（64 位）

(5) AXI 总线接口

3D 图形 AXI DMA 支持 AMBA AIX 总线协议



# 43 AXI 总线

## 43.1 概述

S3C6410 包括三种 AXI 总线，这三种总线将不同的组件连接起来形成 SoC。这三种 AXI 总线分别是：

AXI\_SYS：为系统总线

AXI\_PERI：执行 SFR 访问的通道

AXI\_SFR：补充 AXI\_PERI 用于 SFR 访问 AHB 子系统。

### 1.AXI 总线的性能

- (1) 稀疏的连接操作可以减少门计数并提高安全性
- (2) 支持 AXI 的 32 位或 64 位数据宽度
- (3) 支持 AMBA 2 APB 和 AMBA 3 APB 的 32 位数据宽度
- (4) 可编程的 QoS 系统
- (5) APB 接口提供访问程序寄存器的功能

### 2.仲裁制度

所有的 AXI 总线，如 AXI\_SYS, AXI\_PERI 和 AXI\_SFR 执行判决，可以判决多个总线中哪个总线作为主总线。仲裁制度可以为下面形式中的一种：固定优先级；轮转形式；固定优先级与轮转形式相结合。

默认的仲裁制度采用固定优先级或轮转形式，用户可以在开机或运行时间内通过更新仲裁控制寄存器改变仲裁制度。对于写和读通道，每个 AXI 总线执行一个单一的共享仲裁。

#### (1) 优先群体

在一个优先群体内所有的主机有相同的优先级。作为仲裁的结果，主机可以移动其内部优先组，但是不能离开其组，新的主机不可以加入该组。仲裁权利授给最高优先级的组，组内的成员都可以视图赢得机会，但是仲裁权利只会给组内的最高主机。当主机得到仲裁权利时，将被降级到组的底部，其目的是为了确保持取得仲裁权利的主机不能阻止主内其他主机访问从动装置。

(2) 固定优先级操作

如果所有主机仲裁顺序值相同，将执行轮转形式的优先级方案。执行轮转形式优先级方案是因为处理的降级的主机是最后一个访问组的底部，因此从 LRG 主机的顶部开始到 MRG 的底部进行排序。

(3) 同时操作

当控制寄存器以相同或独特的仲裁顺序的混合形式运行时，轮转形式和固定优先级模式同时存在。用户可以混合一个包含优先级群组成员的优先级群组，使优先级群组包含一个或多个仲裁方式。仲裁者的安排没有组序号和成员身份的限制。

3.可编程的服务质量（QoS）

(1) QOS TIDEMARK 寄存器

可以用希望保留给 QoS 主接口联合签发能力槽的数目进行编程。可以写入此区域的最大值比主接口联合签发能力槽的数目值少一个。这将保证了至少有一个未保留槽，因此不会发生僵局现象。如果写入更多的值，超过了寄存器可以容纳的最大允许值，也不会出现错误提示。将允许进行下面的处理，可以覆盖主接口保留槽的最大数目：

- 1) 向 qos\_tidemarkl 寄存器写入 31，就是 0x1F。
- 2) 读回寄存器的内容。获得的值是保留槽的最大可能值。

(2) QOS 访问控制寄存器

此寄存器内任何位置的 1 值将表示允许从接口相对应的位使用设备连接联合接收能力的保留槽。

可以向此寄存器内写入的最大值是： $(2^{\text{从接口的总数}}) - 1$

如果写入更大的值，超过了寄存器可以容纳的最大值，也不会出现任何错误提示。在这些值写入之后，在第一个可能仲裁时间上改变这些值。

43.2 总线互连

这部分主要指出 AXI 总线的互连。

1. AXI\_SYS 的从接口（主端口）

端口 ID	互连	初始仲裁顺序
S0	AHB_I	1

S1	AHB_F	0
S2	AHB_P	2
S3	AXI_V	7
S4	AHB_X	8
S5	AHB_T	9
S6	AHB_M	3
S7	AHB_S	4
S8	ARM_I	11
S9	ARM_RW	12
S10	ARM_DMA	13
S11	AHB_CF	10
S12	G3D_PF0	5
S13	G3D_TC	6
S14	G2D	2

2.AXI\_SYS 主接口（从端口）

端口 ID	互连
M0	AHB_IROM
M1	AHB_SMC
M2	DMC0
M3	DMC1

3.AXI\_PERI 从接口（主端口）

端口 ID	互连	初始仲裁顺序
S0	AHB_P	1
S1	AHB_M	0
S2	AHB_T	2
S3	AXI_S	3

4.AXI\_PERI 主接口（从端口）

端口 ID	互连
M0	AHB_SMC
M1	AHB_VICTZIC
M2	AHB_G3D
M3	AXI_SFR
M4	APB0
M5	APB1

5.AXI\_SFR 从接口（主端口）

端口 ID	互连	初始仲裁顺序
S0	AHB_PERI	0

6.AXI\_SFR 主接口（从端口）

端口 ID	互连
M0	ETB
M1	AHB_T
M2	AHB_M
M3	AHB_P
M4	AHB_F
M5	AHB_I
M6	AHB_X
M7	AHB_S

# 43.3 稀疏互连

PL301 执行稀疏互连

## 1.AXI\_SYS 稀疏互连

	M0	M1	M2	M3
S0	X	0	0	0
S1	X	0	0	0
S2	X	0	0	0
S3	X	0	0	0
S4	X	0	0	0
S5	X	0	0	0
S6	X	0	0	0
S7	0	0	0	0
S8	0	0	0	0
S9	0	0	0	0
S10	0	0	0	0
S11	X	0	0	0
S12	X	0	0	0
S13	X	0	0	0
S14	X	0	0	0

## 2.AXI\_PERI 稀疏互连

	M0	M1	M2	M3	M4	M5
S0	0	0	0	0	0	0
S1	0	X	0	0	0	0
S2	0	0	0	0	0	0
S3	0	X	0	0	0	0

# 43.4 连接 32 位主总线到 64 位 AXI\_SYS

AXI\_SYS 执行与 64 位数据宽度的 AXI 兼容的骨干总线。当 32 位数据宽度主总线访问 64 位数据宽度的 AXI\_SYS 时，有两个从 32 位数据产生 64 位骨干数据的方法。

(1) 只用低 32 位 AXI\_SYS 管通道，使剩下的高 32 位无效。总线应用的效率低，可以没有限制的应用于任何主从组合。

(2) 尽量减小 AXI\_SYS 的传输，在 AXI\_SYS 上，可以将两个 32 位总线转换打包变为一个 64 位转换。这时有有效的利用 AXI\_SYS 总线通道，但是不可以用于与所有的主从对。

下面的表格显示出对于每个主从对，是否可以将两个 32 位总线打包。

	M0	M1	M2	M3
S0	No channel	No-packing	Packing	Packing
S1	No channel	No-packing	Packing	Packing
S2	No channel	No-packing	Packing	Packing
S3	No channel	No-packing	Packing	Packing
S4	No channel	No-packing	Packing	Packing
S5	No channel	No-packing	Packing	Packing
S6	No channel	No-packing	Packing	Packing
S7	No-packing	No-packing	Packing	Packing
S8	64 bit bus master	64 bit bus master	64 bit bus master	64 bit bus master
S9	64 bit bus master	64 bit bus master	64 bit bus master	64 bit bus master
S10	64 bit bus master	64 bit bus master	64 bit bus master	64 bit bus master
S11	No channel	No-packing	Packing	Packing
S12	No channel	64 bit bus master	64 bit bus master	64 bit bus master
S13	No channel	64 bit bus master	64 bit bus master	64 bit bus master
S14	No channel	64 bit bus master	64 bit bus master	64 bit bus master

上面表格中每部分的含义如下：

No channel: 由于备件连接，总线主从之间没有通道

No-packing: 不允许将两个 32 位转换打包为一个 64 位转换

Packing: 可以将两个 32 位转换打包为一个 64 位转换

64bit bus master: 总线主为 64 位宽度，因此不需要打包

由 0x7E00\_F834 地址上的 BUS\_CACHEABLE\_CON 决定是否使用打包。

### 43.5 AXI 总线主 IDS

在 AXI 总线上，每个总线主有一个专门的唯一的 ID 用于总线从

AXI ID	主总线名	相关 IPS
0000_0000	I 模块	相机, JPEG
0000_0001	F 模块	显示控制器
0000_0010	P 模块	2D, TV 编码器, TB 定标器
XXXX_0011 <sup>1</sup>	V 模块	MFC
0000_0100	X 模块	HSMMC, USB OTG
0000_0101	T 模块	Host I/F
0000_0110	M 模块	DAMO, DMA1
0000_0111	S 模块	安全子模块, SDMA0, SDMA1
0000_1000	ARM 指令	ARM 核心指令
0000_1001	ARM 数据	ARM 核心数据
0000_1010	ARM DMA	ARM 核心 DMA
0000_1011	CF	CFCON
000X_1100 <sup>1</sup>	G 模块	G3D
000X_1101 <sup>1</sup>	G 模块	G3D
0000_1110	G2D	G2D

# 43.6 寄存器描述

## 1.ARBITRATION ORDER NUMBER FOR AXI\_SYS

寄存器	地址	读/写	描述	复位值
AXI_SYS_ARBIT_CTRL_REG0	0x7E003000	读写	从接口 0 的仲裁顺序序号	0x0000_0001
AXI_SYS_ARBIT_CTRL_REG1	0x7E003020	读写	从接口 1 的仲裁顺序序号	0x0000_0000
AXI_SYS_ARBIT_CTRL_REG2	0x7E003040	读写	从接口 2 的仲裁顺序序号	0x0000_0002
AXI_SYS_ARBIT_CTRL_REG3	0x7E003060	读写	从接口 3 的仲裁顺序序号	0x0000_0007
AXI_SYS_ARBIT_CTRL_REG4	0x7E003080	读写	从接口 4 的仲裁顺序序号	0x0000_0008
AXI_SYS_ARBIT_CTRL_REG5	0x7E0030A0	读写	从接口 5 的仲裁顺序序号	0x0000_0009
AXI_SYS_ARBIT_CTRL_REG6	0x7E0030C0	读写	从接口 6 的仲裁顺序序号	0x0000_0003
AXI_SYS_ARBIT_CTRL_REG7	0x7E0030E0	读写	从接口 7 的仲裁顺序序号	0x0000_0004
AXI_SYS_ARBIT_CTRL_REG8	0x7E003100	读写	从接口 8 的仲裁顺序序号	0x0000_000B
AXI_SYS_ARBIT_CTRL_REG9	0x7E003120	读写	从接口 9 的仲裁顺序序号	0x0000_000C
AXI_SYS_ARBIT_CTRL_REG10	0x7E003140	读写	从接口 10 的仲裁顺序序号	0x0000_000D
AXI_SYS_ARBIT_CTRL_REG11	0x7E003160	读写	从接口 11 的仲裁顺序序号	0x0000_000A
AXI_SYS_ARBIT_CTRL_REG12	0x7E003180	读写	从接口 12 的仲裁顺序序号	0x0000_0005
AXI_SYS_ARBIT_CTRL_REG13	0x7E0031A0	读写	从接口 13 的仲裁顺序序号	0x0000_0006
AXI_SYS_ARBIT_CTRL_REG14	0x7E0031C0	读写	从接口 14 的仲裁顺序序号	0x0000_0002

AXI_SYS_ARBIT_CTRL_REGn	位	描述	初始状态
Reserved	[31:8]	保留	0x0000_000
Order value	[7:0]	从接口 n 的仲裁顺序序号	-

## 2. QOS TIDEMARK FOR AXI\_SYS

寄存器	地址	读/写	描述	复位值
AXI_SYS_QOS_CTRL_REG0	0x7E003400	读写	主接口 0 的 QOS Tiemark	0x0000_0000
AXI_SYS_QOS_CTRL_REG1	0x7E003404	读写	主接口 0 的 QOS 访问控制	0x0000_0000



AXI_SYS_QOS_CTRL_REG2	0x7E003420	读写	主接口 1 的 QoS Tiemark	0x0000_0000
AXI_SYS_QOS_CTRL_REG3	0x7E003424	读写	主接口 1 的 QoS 访问控制	0x0000_0000
AXI_SYS_QOS_CTRL_REG4	0x7E003440	读写	主接口 2 的 QoS Tiemark	0x0000_0000
AXI_SYS_QOS_CTRL_REG5	0x7E003444	读写	主接口 2 的 QoS 访问控制	0x0000_0000
AXI_SYS_QOS_CTRL_REG6	0x7E003460	读写	主接口 3 的 QoS Tiemark	0x0000_0000
AXI_SYS_QOS_CTRL_REG7	0x7E003464	读写	主接口 3 的 QoS 访问控制	0x0000_0000

AXI_SYS_QOS_CTRL_REG0	位	描述	初始状态
Reserved	[31:0]	保留	0x0000_0000

AXI_SYS_QOS_CTRL_REG1	位	描述	初始状态
Reserved	[31:0]	保留	0x0000_000

AXI_SYS_QOS_CTRL_REG2	位	描述	初始状态
Reserved	[31:0]	保留	0x0000_000

AXI_SYS_QOS_CTRL_REG3	位	描述	初始状态
Reserved	[31:0]	保留	0x0000_000

AXI_SYS_QOS_CTRL_REG4	位	描述	初始状态
Reserved	[31:6]	保留	0x0
QoS Tidemark	[5:0]	主接口 2 的 QoS Tidemark	0x0

AXI_SYS_QOS_CTRL_REG5	位	描述	初始状态
Reserved	[31:15]	保留	0x0
QoS Tidemark	[14:0]	主接口 2 的 QoS Tidemark	0x0

AXI_SYS_QOS_CTRL_REG6	位	描述	初始状态
-----------------------	---	----	------

Reserved	[31:6]	保留	0x0
QoS Tidemark	[5:0]	主接口 3 的 QoS Tidemark	0x0

AXI_SYS_QOS_CTRL_REG7	位	描述	初始状态
Reserved	[31:15]	保留	0x0
QoS Tidemark	[14:0]	主接口 3 的 QoS Tidemark	0x0

### 3. AXI\_PERI 的仲裁顺序序号

寄存器	地址	读/写	描述	复位值
AXI_PERI_ARBIT_CTRL_REG0	0x7E008000	读写	从接口 0 的仲裁顺序序号	0x0000_0000
AXI_PERI_ARBIT_CTRL_REG1	0x7E008020	读写	从接口 0 的仲裁顺序序号	0x0000_0001
AXI_PERI_ARBIT_CTRL_REG2	0x7E008040	读写	从接口 1 的仲裁顺序序号	0x0000_0002
AXI_PERI_ARBIT_CTRL_REG3	0x7E008060	读写	从接口 1 的仲裁顺序序号	0x0000_0003

AXI_FERI_ARBIT_CTRL_REGn	位	描述	初始状态
Reserved	[31:8]	保留	0x0000_000
Order value	[7:0]	从接口 n 的仲裁顺序序号	-

### 4. QOS TIDEMARK FOR AXI\_PERI

寄存器	地址	读/写	描述	复位值
AXI_PERI_QOS_CTRL_REG0	0x7E008400	读写	主接口 0 的 QOS Tiemark	0x0000_0000
AXI_PERI_QOS_CTRL_REG1	0x7E008404	读写	主接口 0 的 QOS 访问控制	0x0000_0000
AXI_PERI_QOS_CTRL_REG2	0x7E008420	读写	主接口 1 的 QOS Tiemark	0x0000_0000
AXI_PERI_QOS_CTRL_REG3	0x7E008424	读写	主接口 1 的 QOS 访问控制	0x0000_0000
AXI_PERI_QOS_CTRL_REG4	0x7E008440	读写	主接口 2 的 QOS Tiemark	0x0000_0000
AXI_PERI_QOS_CTRL_REG5	0x7E008444	读写	主接口 2 的 QOS 访问控制	0x0000_0000
AXI_PERI_QOS_CTRL_REG6	0x7E008460	读写	主接口 3 的 QOS Tiemark	0x0000_0000
AXI_PERI_QOS_CTRL_REG7	0x7E008464	读写	主接口 3 的 QOS 访问控制	0x0000_0000

AXI_PERI_QOS_CTRL_REG8	0x7E008480	读写	主接口 4 的 QOS Tiemark	0x0000_0000
AXI_PERI_QOS_CTRL_REG9	0x7E008484	读写	主接口 4 的 QOS 访问控制	0x0000_0000
AXI_PERI_QOS_CTRL_REG10	0x7E0084A0	读写	主接口 5 的 QOS Tiemark	0x0000_0000
AXI_PERI_QOS_CTRL_REG11	0x7E0084A4	读写	主接口 5 的 QOS 访问控制	0x0000_0000

AXI_PERI_QOS_CTRL_REGn	位	描述	初始状态
Reserved	[31:0]	保留	0x0000_0000

## 5.AXI\_SFR 的仲裁顺序序号

寄存器	地址	读/写	描述	复位值
AXI_SFR_ARBIT_CTRL_REG0	0x7E009000	读写	从接口 0 的仲裁顺序序号	0x0000_0000

AXI_SFR_ARBIT_CTRL_REGn	位	描述	初始状态
Reserved	[31:8]	保留	0x0000_000
Order value	[7:0]	从接口 n 的仲裁顺序序号	-

## 6. QOS TIDEMARK FOR AXI\_SFR

寄存器	地址	读/写	描述	复位值
AXI_SFR_QOS_CTRL_REG0	0x7E009400	读写	主接口 0 的 QOS Tiemark	0x0000_0000
AXI_SFR_QOS_CTRL_REG1	0x7E009404	读写	主接口 0 的 QOS 访问控制	0x0000_0000
AXI_SFR_QOS_CTRL_REG2	0x7E009420	读写	主接口 1 的 QOS Tiemark	0x0000_0000
AXI_SFR_QOS_CTRL_REG3	0x7E009424	读写	主接口 1 的 QOS 访问控制	0x0000_0000
AXI_SFR_QOS_CTRL_REG4	0x7E009440	读写	主接口 2 的 QOS Tiemark	0x0000_0000
AXI_SFR_QOS_CTRL_REG5	0x7E009444	读写	主接口 2 的 QOS 访问控制	0x0000_0000
AXI_SFR_QOS_CTRL_REG6	0x7E009460	读写	主接口 3 的 QOS Tiemark	0x0000_0000
AXI_SFR_QOS_CTRL_REG7	0x7E009464	读写	主接口 3 的 QOS 访问控制	0x0000_0000
AXI_SFR_QOS_CTRL_REG8	0x7E009480	读写	主接口 4 的 QOS Tiemark	0x0000_0000
AXI_SFR_QOS_CTRL_REG9	0x7E009484	读写	主接口 4 的 QOS 访问控制	0x0000_0000

AXI_SFR_QOS_CTRL_REG10	0x7E0094A0	读写	主接口 5 的 QOS Tiemark	0x0000_0000
AXI_SFR_QOS_CTRL_REG11	0x7E0094A4	读写	主接口 5 的 QOS 访问控制	0x0000_0000
AXI_SFR_QOS_CTRL_REG12	0x7E0094C0	读写	主接口 6 的 QOS Tiemark	0x0000_0000
AXI_SFR_QOS_CTRL_REG13	0x7E0094C4	读写	主接口 6 的 QOS 访问控制	0x0000_0000
AXI_SFR_QOS_CTRL_REG14	0x7E0094E0	读写	主接口 7 的 QOS Tiemark	0x0000_0000
AXI_SFR_QOS_CTRL_REG15	0x7E0094E4	读写	主接口 7 的 QOS 访问控制	0x0000_0000

AXI_SFR_QOS_CTRL_REGn	位	描述	初始状态
Reserved	[31:0]	保留	0x0000_0000