



SUNWAY 申威

ICH2 低速 IO 设备 详细设计方案

2017 年 10 月

成都申威科技有限责任公司



免责声明

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

成都申威科技有限责任公司

Chengdu Sunway Technology Corporation Limited

地址：成都市华府大道四段电子科大科技园 D22 栋

Building D22, National University Science and technology park,
Section 4, Huafu Avenue, Chengdu

Mail: sales@swcpu.cn

Tel : 028-68769016

Fax: 028-68769019



阅读指南

《ICH2 低速 IO 设备详细设计方案》主要描述了国产第二代套片的结构框图、各种外设功能接口等内容。

文档修订

文档更新记录	文档名	ICH2 低速 IO 设备详细设计方案
	版本号	V1.0
	创建人	研发部
	创建日期	2017-10-8

版本更新

版本号	更新内容	更新日期
V1.0	初稿	2017-10-8

技术支持

可通过邮箱或问题反馈网站向我司提交产品使用的问题，并获取技术支持。

售后服务邮箱：sales@swcpu.cn

问题反馈网址：<http://www.swcpu.cn/>

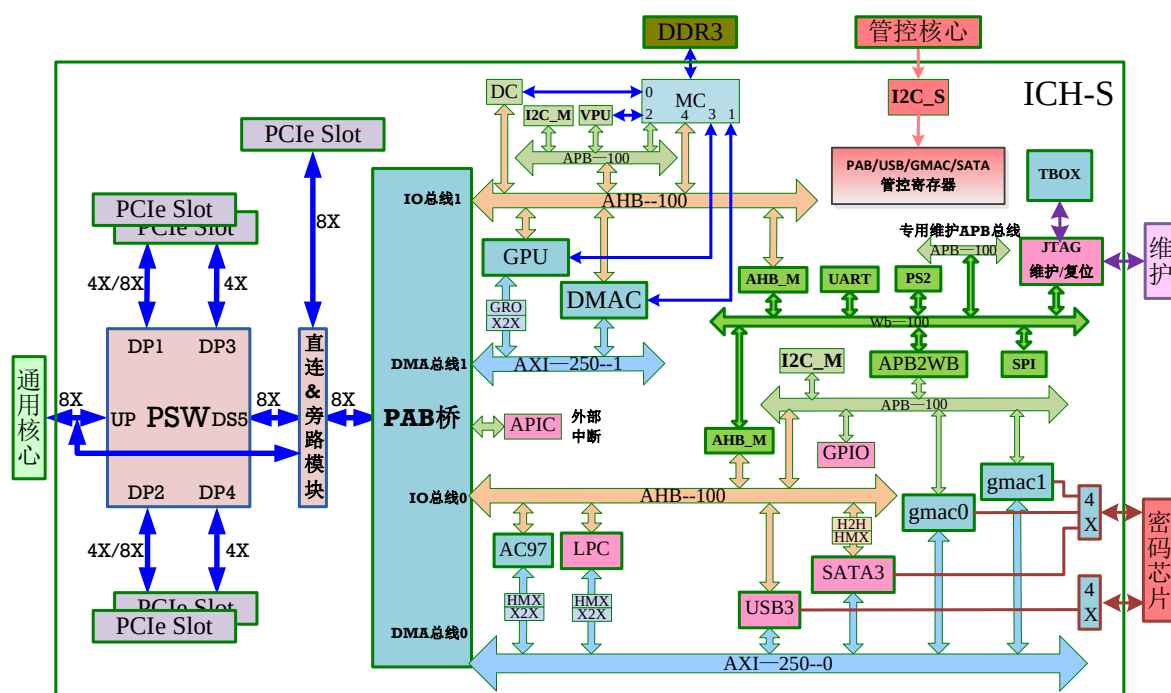
目 录

1	安全 IO 芯片和低速 IO 设备结构框图	1
1.1	安全 IO 芯片 ASIC 架构	1
1.2	低速 IO 设备架构	2
2	JTAG 接口	3
2.1	接口概述	3
2.2	接口信号	3
2.3	访问命令和指令支持	4
2.4	工作流程	4
3	I2C 管控接口	8
3.1	I2C 协议说明	8
3.2	寄存器定义 (i2c slave address='h08)	9
3.3	写寄存器实现	11
3.4	读寄存器实现	12
3.5	I2C 管控模块接口	13
3.6	约定说明	15
4	SPI 接口	16
4.1	功能描述	16
4.1.1	概述	16
4.1.2	时钟	16
4.1.3	发送和接收 FIFO	17
4.1.4	SSI 中断	17
4.1.5	传输模式	18
4.1.6	操作模式	19
4.2	参数定义	19
4.3	信号描述	21
4.4	寄存器定义	21
4.5	工作流程	24
4.5.1	Flash 访问命令	24
4.5.2	编程流程	26
4.5.3	EEPROM 读流程	27
4.5.4	正常工作读写流程	29
5	GPIO 接口	30
5.1	IP 配置参数	30
5.2	接口信号	32
5.3	寄存器定义	33
5.4	软件控制实现	34
6	UART 接口	35
7	PS/2 接口	37
8	APB2WB 桥	39
9	WB2AHB 桥	41
10	WB2APB 桥	41
11	WISHBONE 交叉开关	43

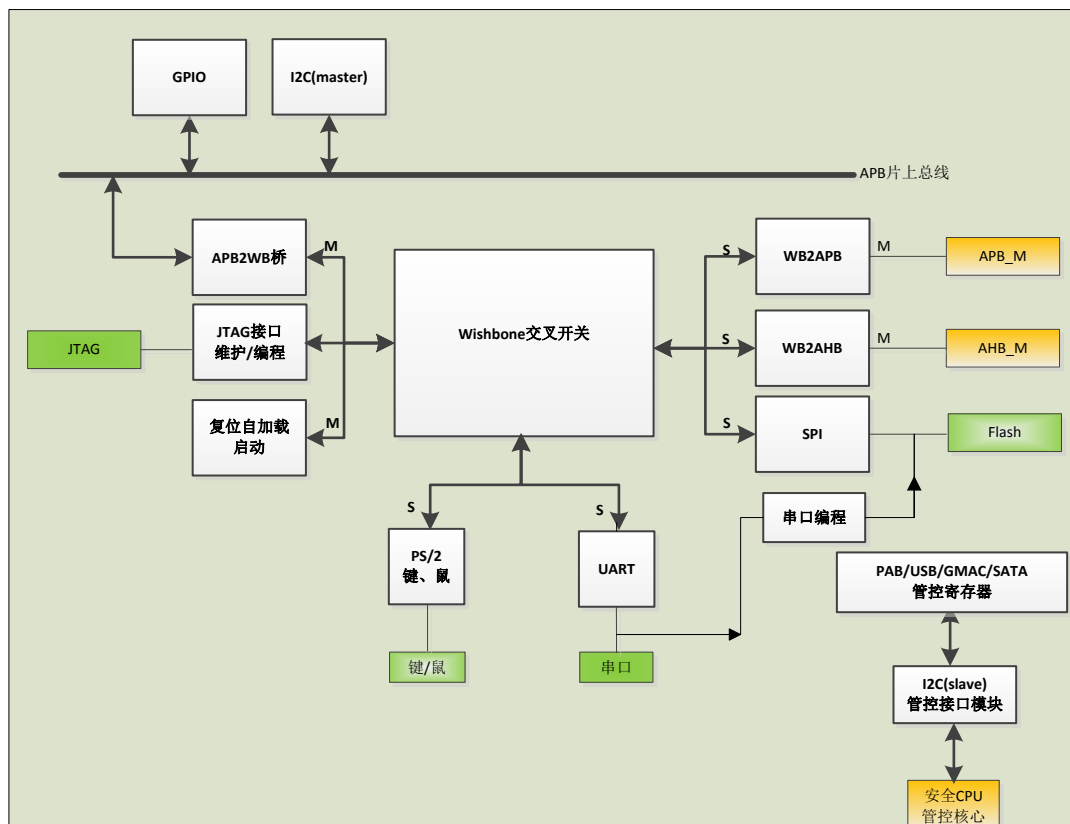
12	复位.....	44
13	串口编程 Flash 芯片（prg_spi）	46
13.1	概述.....	46
13.2	编程命令包格式.....	46
13.3	编程工作流程.....	47
13.4	接口信号.....	48

1 安全 IO 芯片和低速 IO 设备结构框图

1.1 安全 IO 芯片 ASIC 架构



1.2 低速 IO 设备架构



上图中的安全 IO 芯片低速 IO 接口都是没有 DMA 功能的，主要包含如下接口：

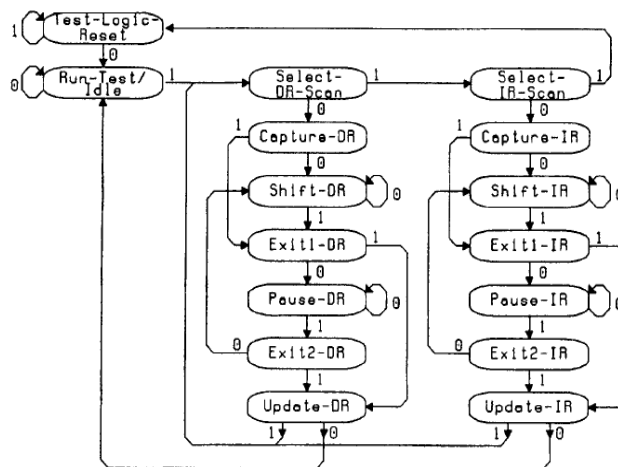
- n 个 I2C master 接口，可以接 DAC 数模转换显示芯片，可用于 SROM 读取挂在 I2C 总线上的内存条信息，以及外接时钟芯片等；
- 1 个 I2C slave 接口，与安全 CPU 管控核心的管控接口连接，接收并转发管控策略和设备使能控制信息，并发送给 PAB 和 USB、GMAC、SATA 需要被管控的设备；
- AMBA 总线连接接口
 - APB 片上总线接口，用于系统对低速 IO 设备的配置访问和读取外接芯片信息等；
 - AHB master 接口，2 个，分别和显示子系统和非显示子系统的 AHB 总线连接，用于对非低速 IO 子系统的设备进行维护访问；
 - APB master 接口，用于对 PAB 桥、USB、GMAC 等状态监测和 PHY 寄存器配置等专用通路访问；
- 1 个 JTAG 维护调试和编程接口，可完成对套片系统的维护调试和 SPI Flash 芯片的编程；
- 1 个复位自加载启动模块，可完成上电复位后对 SPI Flash 芯片的信息自动读取，包括时钟、复位等信息，以及管控策略的初始化配置；
- 1 个 SPI 接口，配置成 master 模式，可对 Flash 进行信息读取加载到系统，也可通过 JTAG 接口对 Flash 进行编程；
- 1 个 UART 接口，1 个 PS2 键/鼠接口；
- 1 个 GPIO 接口。

2 JTAG 接口

2.1 接口概述

安全 IO 芯片的 JTAG 接口由两部分组成：JTAG TAP 控制器和 DBG 模块，其中 JTAG TAP 控制器按照 JTAG 协议将标准的 jtag 输入信号转换成对指令寄存器或数据寄存器的操作，输出给 DBG 模块，DBG 模块将这些操作转换成对自身或 wishbone 交叉开关下挂接设备的访问；同样 wishbone 交叉开关下挂接设备访问返回的响应，通过 DBG 模块转换成对指令寄存器或数据寄存器的操作，JTAG TAP 控制器将该操作转换成 jtag 输出信号。

TAP 控制器是一个 16 状态的状态机，和 IEEE1149.1 标准兼容，状态机转换如下：



安全 IO 芯片的 JTAG 接口要能实现对 wishbone 交叉开关下挂接的 UART、SPI、PS/2 接口的读写访问，其中对 SPI 接口的 Flash/EEPROM 能实现读写访问和编程命令，SPI 接口 Flash/EEPROM 的访问是间接访问，具体访问方式和工作流程参考第 4 章的说明。另外，还要求 JTAG 接口能访问 WB2AHB 桥接模块下挂接的维护子系统地址空间，以及 WB2APB 桥接模块下挂接的专用维护子系统地址空间。

UART、PS/2 的寄存器不管读还是写都是字节粒度访问，SPI 的 DR 寄存器是字节粒度访问，SPI 的其它寄存器有可以使用非字节粒度访问的。维护子系统下挂接的设备地址有字节、字和双字粒度访问模式。

2.2 接口信号

JTAG 接口的外部信号包括标准的 JTAG 输入输出信号，以及连接到 wishbone 交叉开关上对挂在交叉开关上的 slave 设备进行访问的 wishbone master 信号。具体信号列表如下：

信号	属性	含义
与外部 jtag master 接口信号		
tck_pad_i	输入	JTAG 时钟
tms_pad_i	输入	JTAG 状态信号
tdi_pad_i	输入	JTAG 串行输入数据
tdo_pad_o	输出	JTAG 串行数据输出

trstn_pad_i	输入	复位
与 wishbone 交叉开关接口信号		
wb_cyc_o	输出	总线请求/继续标志
wb_stb_o	输出	请求开始总线周期
wb_ack_i	输入	周期结束响应
wb_sel_o[3:0]	输出	字节使能
wb_we_o	输出	写使能
wb_err_i	输入	总线错误标志
wb_adr_o[31:0]	输出	总线地址
wb_dat_i[31:0]	输入	读数据
wb_dat_o[31:0]	输出	写数据

2.3 访问命令和指令支持

安全 IO 芯片的 JTAG TAP 控制器支持的指令如下表所示：

指令名称	编码
EXTEST	4'b0000
SAMPLE_PRELOAD	4'b0001
IDCODE	4'b0010
DEBUG	4'b1000
MBIST	4'b1001
BYPASS	4'b1111

对于 wishbone 子系统设备空间访问相关的指令是 IDCODE 和 DEBUG，IDCODE 可以访问标志寄存器，DEBUG 指令是进入 debug 模式访问 wishbone 子系统。

安全 IO 芯片 JTAG DBG 模块支持的访问命令如下表所示：

命令名称	编码
BWRITE8	4'h1
BWRITE16	4'h2
BWRITE32	4'h3
BREAD8	4'h5
BREAD16	4'h6
BREAD32	4'h7
IREG_WR (internal register select/write)	4'h9
IREG_SEL (internal register select)	4'hd

2.4 工作流程

为了实现 JTAG 接口对 wishbone 交叉开关下挂载的设备空间的访问，需要进行如下工作流程：

- 1) 进入 DEBUG 模式：设置指令寄存器的值为 DEBUG(4'b1000)，即按照 TAP 控制器的状态机 set_ir 的转换对 TMS 和 TDI 进行合适的赋值，TAP 控制器会在 TCK 的上升沿对 TMS 和 TDI 的值采样。对 TMS 依次写入'b01100 后进入 shift_ir 状态，TMS 依次写入'b000，同时 TDI 依次写入'b000 (ir[2:0])，然后 TMS 写入'b1，TDI 写入'b1(ir[3])，跳出 shift_ir 状态。最后 TMS 依次写入'b0110 跳到 Run-Test Idle 状态，完成指令的设置。
- 2) 进行读写命令访问：通过对 TMS 和 TDI 进行合适的赋值，执行写数据寄存器的操作，发送读写命令，并可以读取数据寄存器获得 TDO 的输出。

a) 读命令：类似上面的写 DBG 指令的过程，按照 TAP 控制器的状态流转控制 TMS 输入的值，进入 Shift DR 状态，将 53 位的数据依次移位 TDI 输入，格式如下：

data_scan_in[52] — module_cmd=~data_scan_in[52]，module_cmd 为 1，代表访问该模块，否则是访问顶层的操作。对 wishbone 子系统的访问 data_scan_in[52]=0；

data_scan_in[51:48] — operation，对应 2.3 表中的 BREAD8、BREAD16、BREAD32 命令；

data_scan_in[47:16] — address，其中[47:40]位对应 wishbone 交叉开关 slave 号，ps/2=4'h0，uart=4'h1，spi=4'h2，WB2AHB 桥=4'h3，WB2APB 桥=4'h4，地址[39:16]对应具体的寄存器地址；

data_scan_in[15:0] — byte count，连续发送的命令数目，为“1”即为 single 操作，如果不为“1”，在 wishbone 总线上表现为发出连续的 N 个操作。

然后退出 Shift DR 状态，直到回到 Run-Test Idle 状态，完成读命令的发送。

然后再次进入 Shift DR 状态，依次获得 TDO 的输出数据，即读响应数据。在获得读响应数据时需要注意，TAP 状态机在进入 Shift DR 状态后，移出的数据首先是若干拍 biu_ready=0 的信号，然后是一拍 biu_ready=1 的信号，接下来才是真正的数据。由于 TCK 时钟和 WB_CLK 时钟频差浮动，biu_ready（即 wb_ack，读响应有效）有效的时刻和进入 Shift DR 时刻的节拍差也不定，所以需要从移出的 TDO 数据的最低位依次向高位判断，第一个值为“1”的位就是 biu_ready，然后才是真正的读响应数据。

需要注意的是，在获得 TDO 输出数据时，除了有读取寄存器的内容外还会有 CRC 校验数据紧随在正常数据之后，可以直接忽略该 CRC 数据。

总结：第一次进入 shift_dr 等一系列数据移位的完整 tap 状态机流程是把读控制等信息发出，而第二次进入 shift_dr 等 tap 状态机流程才是读取响应数据，并且在第二次 shift_dr 状态进行读响应数据移出时，如果是 BREAD32，移出的数据位数必须大于 32 位，要判断读到第一个“1”的值就是 read status，下一位开始才是真正的数据，实际环境中移出 33 位即可，因为由于两个命令之间的时间间隔一定会使 read status ok 提前准备好，在模拟环境中可能需要移出更多位，除非两个命令之间间隔合适，或者频差在一定范围；如果是 BREAD8，移出的数据位数必须大于 8，adbg 模块会将读响应数据挪到最低位开始存放。当然，无论是什么粒度的读操作，都可以移出大于 32 位的数据，只要按照需要的位数取出即可。

b) 写命令：写命令的执行前面类似读命令，不同的是 data_scan_in[51:48]的 operation 对应的是 BWRITE8、BWRITE16 和 BWRITE32。

退出 Shift DR 状态，直到回到 Run-Test Idle 状态，完成写命令的发送。

然后再次进入 Shift DR 状态，data_scan_in[52:21]的写入数据依次输入 TDI 接口，直到回到 Run-Test Idle 状态，完成写数据的发送。

在发送写数据后，可以按照同样的方法再发送 32 位 CRC 数据，DBG 模块会进行写数据校验，为了简化处理，可以不发送 CRC 数据，只发送纯写数据即可。

总结：类似读命令，第一次进入 shift_dr 是发送写控制信息，第二次进入 shift_dr 才是移入数据。如果是 BWRITE32 操作，那么移入的数据要是 33 位，第[0]位必须为“1”，是一个写数据开始的标志，第[32:1]才是真正写入的数据；如果是 BWRITE8 操作，移入的数据可以是 9 位就行，同样第[0]位必须为“1”，[8:1]才是真正写入的数据。不论什么粒度的写操作都可以统一处理成移位 33 位数据，adbg 会取出合适的数据。而且无论写的地址是否 4B 对接，只要把有效写数据从第[1]位开始放置即可，adbg 会按照写的粒度和地址低两位的值，把写数据放到合适的位置发到 wishbone 总线上。

3) DBG 内部寄存器访问：写入的命令不是 READ 或 WRITE，而是通过命令 IREG_WR 和 IREG_SEL 选择执行写入内部寄存器的操作以及操作的具体寄存器。目前实现了 5 个内部寄存器，编址和内容如下：

寄存器号	名称	含义	默认值 / 属性	备注
0	error	Wishbone 响应错误指示	'h0 RO	ASIC 和 FPGA 版本
1	syncrst	同步复位，复位信号低有效	'h1 RW	仅 FPGA 版本
2	corerst	核心逻辑复位，复位信号低有效	'h1 RW	仅 FPGA 版本
3	mtrst	维护复位，复位信号低有效	'h1 RW	仅 FPGA 版本
4	fpagst[15:0]	FPGA 状态： [0]：屏蔽 MC，缺省为 1，表示不屏蔽 [1]：屏蔽 PSW，缺省为 0，表示屏蔽 [2]：屏蔽 GPU，缺省为 0，表示屏蔽 [3]：屏蔽 VPU，缺省为 0，表示屏蔽 [6:4]：保留 [7]:gmac0_phy_intf_sel, 缺省为 0， “0”GMII/MII,“1” RGMII [15:8]：版本号	'h1 [7:0]RW [15:8]RO	仅 FPGA 版本
5	axi_timeout[10:0]	AXI 总线硬连线接入总线超时相关信息	'h0 RO	ASIC 和 FPGA 版本
6	lpc32m_sel[3:0]	用于配置 LPC 的 MEMIO 和 FWIO 的访问哪个 32MB 空间偏移	'h0 RW	ASIC 和 FPGA 版本
7	clu_par[4:0]	用于配置 CLU 相关环振参数：	'h0	ASIC 和

		[0]: wbsys2ring_en, 环振使能, 1-使能, 0-关闭 [1]: wbsys2ring_lh_sel, 环振 HVT/LVT 输出选择, 1 选择 LVT, 0 选择 HVT [4:2]: wbsys2clu_ring_out_sel, 8 个环振的输出选择信号	RW	FPGA 版本
--	--	--	----	---------

访问内部寄存器的格式类似访问 wishbone 寄存器, [52]=0, [51:48]位为操作码 (写操作为 IREG_WE, 4'h9, 读操作为 IREG_SEL, 4'hd), [47:45]为内部寄存器地址, [44:13]为写数据内容 (如果寄存器只有 1 位数据, 数据定义在[44]位, 如果只有 8 位数据, 数据定义在[44:37]位, [44]位对应要写的最高位数据)。但是对于 fpgast 寄存器, 仍然要按照 16 位写数据操作, 只不过高 8 位数据写不进去, 低 8 位数据才会被写入, 就是说寄存器定义为多少位, 写数据长度就应该是多长。

读操作第一步都需要移入 53 位控制信息, 第二步移出寄存器定义位数的数据即可。写操作只需要移入 53 位控制信息和数据。

3 I2C 管控接口

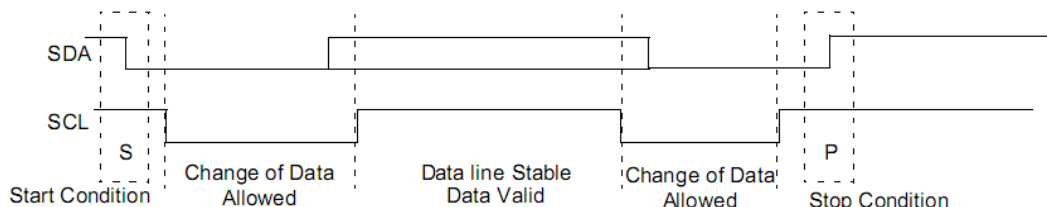
通过标准的 I2C 接口与安全处理器相连，接收来自管控核心的配置请求，生成启闭控制和工作模式控制信息，该 I2C 接口对安全处理器的计算核心不可见，只对管控核心可见。设置 USB、GMAC、SATA 的分设备和端口的策略配置寄存器，以及 PAB 桥的设备配置空间开放使能寄存器，并且可以接收上电复位自动加载模块对这些寄存器的写操作。I2C 管控模块并不实现这些寄存器，这些寄存器只存在于对应的 PAB 和设备模块，管控接口只实现对这些寄存器的读写访问通路。并且这些寄存器只有管控可见，对安全 IO 芯片的 IO 通路不可见。

- 两线 I2C 串行接口，包括串行数据总线（SDA）和串行时钟（SCL）；
- 采用 I2C 7 位地址模式，并且实现成一个 I2C slave；
要求安全处理器的 I2C master 驱动只配置使用 7 位地址模式。
- 只支持固定速率模式（standard mode-100Kb/s，fast mode-400Kb/s，high speed mode-3.4Mb/s），由于不需要考虑兼容其他的 I2C 设备，只是管控专用接口，与 I2C master 约定三种速率中的一种实现；
要求实现 standard mode 和 fast mode 速率模式。
- 不支持软件对 I2C 接口的参数配置，包括 slave address 等设置成固定值，并与 I2C master 形成约定，**同时固件输出 3 位低位 ID 地址配置信号**，slave address[2:0]位可变，防止和其他 I2C Slave 地址造成冲突；
I2C 管控接口定义一个 slave address 地址，要求安全处理器的 I2C master 驱动配置 IC_TAR 寄存器为该地址值。
- 把 I2C master 接口发来的 I2C 请求转化成策略寄存器和信息寄存器的访问，并以 I2C 协议格式返回要求的响应或数据给 master。

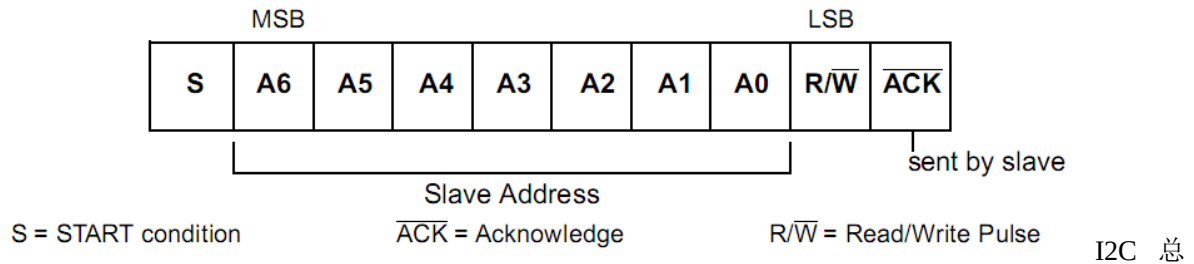
按照上面的与 I2C master 的约定以及传输格式的定义，不需要驱动的参与就可以完成管控核心对策略配置寄存器等的访问。

3.1 I2C 协议说明

当 I2C 总线处于空闲状态，SCL 和 SDA 信号都会被外部上拉电阻置成“1”的值。当 I2C master 想要在总线上开始一次传输，master 会往总线发送一个 START 事件，标志是 SCL 为“1”期间，SDA 会从高电平变成低电平；当 I2C master 想要结束当前传输，master 会往总线发送一个 STOP 事件，标志是 SCL 为“1”期间，SDA 会从低电平变成高电平。当数据在 I2C 总线上传输时，在 SCL 为“1”期间，SDA 需要保持稳定的值。

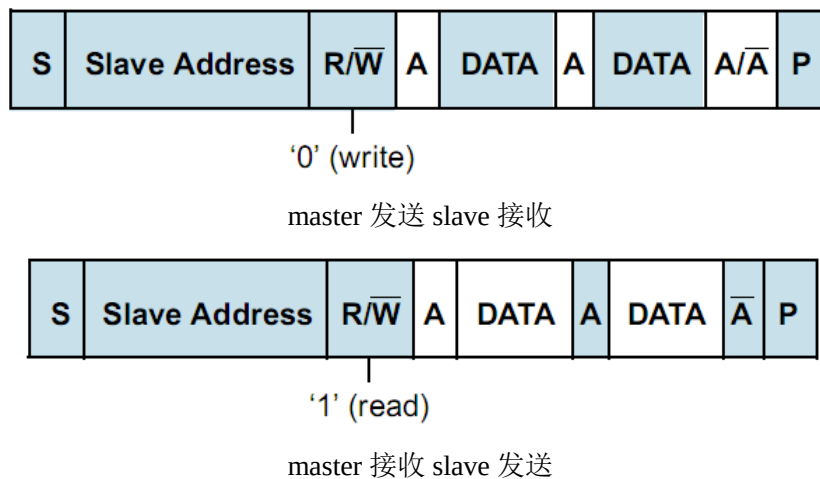


I2C 协议本身支持两种格式的地址模式，7-bit 和 10-bit 地址，I2C 管控接口只实现 7-bit 地址格式，地址字节的高 7 位 bit[7:1]用来传递要访问的 slave 设备地址，最低位 bit[0]用来表示读写标志，R/W 为 0 代表写操作，R/W 为 1 代表读操作。



线上的所有传输都以 byte 格式传输，每一次数据传输的 byte 数目没有限制。
 master 首先发送给 slave 发送一个字节的 slave 地址和 R/W 标志，slave 要给出 ACK 应答。
 master 每次给 slave 发送一个字节的的数据，slave 都需要给出 ACK 应答。
 如果是 master 接收 slave 发送的数据，除了最后一个字节，master 每次接收到 slave 发送的一个字节数据都需要给出 ACK 应答。slave 发送过来的最后一个字节数据时，master 给出 NACK 应答，以此通知 slave 传输即将结束。slave 在收到 NACK 应答后释放 SDA 总线，master 可以发起 STOP 事件了。

如果 master 还有请求要传输，不想发起 STOP 事件释放总线，那么 master 可以发起一个 RESTART 事件。RESTART 事件和 START 事件动作相同，不同的是 RESTART 事件发生在 NACK 应答后，而 START 事件只会发生在 STOP 事件之后或总线空闲期间。



From Master to Slave
 From Slave to Master

A = Acknowledge (SDA low) R = RESTART Condition
 $\overline{\text{A}}$ = No Acknowledge (SDA high) P = STOP Condition
 S = START Condition

3.2 寄存器定义 (i2c slave address='h08)

addr[6:0]	Description	Default	Type
0x0	DVEN:设备配置空间开放使能寄存器，高表示使能，32 位有效 [31:4]: 保留 Bit[3]: GMAC1 Bit[2]: GMAC0 Bit[1]: SATA Bit[0]: USB	0xf	W

	该寄存器没有实现读通路，因为管控接口给 PAB 桥的输出转换成 4 根电平敏感的信号线，而没有实现读写寄存器通路。所以只能按照上面的位定义进行写操作。		
0x4	usbmode_port0: USB 端口 0 设备管控策略寄存器 [31:4]保留 [3]硬件初始设置态 HardSet: 0-受管控核心管控 1-受固件管控 [2]配置态 State: 0-管控核心未配置 1-管控核心已配置 [1:0]工作模式态 Mode: 00-明通 01-密通 10-单向 11-禁用	0x0	RW
0x8	usbmode_port1: USB 端口 1 设备管控策略寄存器	0x0	RW
0xc	usbmode_port2: USB 端口 2 设备管控策略寄存器	0x0	RW
0x10	usbmode_port3: USB 端口 3 设备管控策略寄存器	0x0	RW
0x14	usbmode_port4: USB 端口 4 设备管控策略寄存器	0x0	RW
0x18	usbmode_port5: USB 端口 5 设备管控策略寄存器	0x0	RW
0x1c	gmac0mode: GMAC0 设备管控策略寄存器 [31:4]保留 [3]硬件初始设置态 HardSet: 0-受管控核心管控 1-受固件管控 [2]配置态 State: 0-管控核心未配置 1-管控核心已配置 [1:0]工作模式态 Mode: 00-明通 01-密通 1x-禁用	0x0	RW
0x20	gmac1mode: GMAC1 设备管控策略寄存器	0x0	RW
0x24	satamode_port0: SATA 端口 0 设备管控策略寄存器 [31:4]保留 [3]硬件初始设置态 HardSet: 0-受管控核心管控 1-受固件管控 [2]配置态 State: 0-管控核心未配置 1-管控核心已配置 [1:0]工作模式态 Mode: 00-明通 01-密通 1x-禁用	0x0	RW
0x28	satamode_port1: SATA 端口 1 设备管控策略寄存器	0x0	RW
0x2c	satamode_port2: SATA 端口 2 设备管控策略寄存器	0x0	RW
0x30	usb_id_port0: USB 端口 0 设备信息寄存器-0 [31:16]: USB 端口 0 的 Product ID [15:0]: USB 端口 0 的 Vendor ID	0x0	R
0x34	usb_sn0_port0: USB 端口 0 设备信息寄存器-1	0x0	R

	[31:0]: USB 端口 0 的 Serial Number 低 32 位		
0x38	usb_sn1_port0: USB 端口 0 设备信息寄存器-2 [31:0]: USB 端口 0 的 Serial Number 高 32 位	0x0	R
0x3c	usb_id_port1	0x0	R
0x40	usb_sn0_port1	0x0	R
0x44	usb_sn1_port1	0x0	R
0x48	usb_id_port2	0x0	R
0x4c	usb_sn0_port2	0x0	R
0x50	usb_sn1_port2	0x0	R
0x54	usb_id_port3	0x0	R
0x58	usb_sn0_port3	0x0	R
0x5c	usb_sn1_port3	0x0	R
0x60	usb_id_port4	0x0	R
0x64	usb_sn0_port4	0x0	R
0x68	usb_sn1_port4	0x0	R
0x6c	usb_id_port5	0x0	R
0x70	usb_sn0_port5	0x0	R
0x74	usb_sn1_port5	0x0	R

I2C 管控接口只提供寄存器访问通路，并不实现管控寄存器，各设备自己实现管控寄存器，但是编址统一，地址按照 4 字节对界编址。所有策略寄存器和 ID 信息寄存器只使用维护复位信号进行复位。USB 的 ID 和 SN 寄存器对管控核心是只读的，其他管控寄存器是可读可写。

PAB 的 DVEN 寄存器和管控策略寄存器要配合使用，管控策略是提供给设备控制器来对端口进行管控，而 DVEN 寄存器控制系统发到已关闭设备的请求可以在 PAB 桥一级就直接返回。DVEN 的管控无法做到按设备端口控制，所以是更高一级的控制，使用时需要注意。除了 GMAC 可以和设备端口保持一直，USB 和 SATA 都分别只有在全部端口禁用的情况下，才能利用 DVEN 关闭设备。

维护复位之后，策略寄存器的值配置成 0x0，即受管控核心管控且处于管控未配置态，此时不需要关心工作模式态：

- 如果系统中有管控核心，那么管控核心会将策略寄存器写成'b01xx，设置成管控配置态和指定工作模式态（每个端口工作模式态可以不同）。如果 USB 设备从端口移除，USB 控制器只将第[2]位清 0 即可，工作模式态保持不变。
- 如果系统中没有管控核心，固件将策略寄存器写成'b1xxx，设置成管控配置态和指定工作模式态（每个端口工作模式态要相同）。如果 USB 设备从端口移除，同样控制器只要将第[2]位清 0 即可，工作模式态保持不变。

如果没有管控核心，只有固件配置，USB、SATA 和 GMAC 设备首先看策略寄存器的[3]位，确定是固件管控，则只看[1:0]的工作模式态，不再看[2]位的管控配置态。

I2C master 接口发送来的 I2C 请求和上电复位自加载的写寄存器请求不会同时存在，上电复位自动加载配置了管控策略后，如果有管控核心的存在，再通过管控接口配置 IO 端口的策略，覆盖之前的配置值。

3.3 写寄存器实现

根据上面的寄存器定义，按照 I2C 协议传输格式，在地址字节传输后，设置 1 个字节的

command byte 传输，和 1 个字节的 data byte 传输。

command byte	
位	含义
[7]	命令定义，1'b0，写寄存器命令
[6:0]	要访问的寄存器地址，地址 DW 对界

一个完整的 I2C 写寄存器包的传输格式如下：

1 st Cycle			
START	7 6 5 4 3 2 1	0	ACK/NAK
S	Slave Address[7:1]	Read/Write Bit 0=Write	A

Command Cycle	
7 6 5 4 3 2 1 0	ACK/NAK
Command Byte	A

Write Data Cycle		
76543210	ACK/NAK	STOP
Data Byte 0 selected register byte 0	A	P

from master to slave
 from slave to master

3.4 读寄存器实现

I2C 读寄存器的实现使用两个包完成，第一个包用来传输地址和命令，选择要读取的寄存器；第二个包用来传输地址和读响应数据。地址和命令字节的定义和写寄存器类似。

也可以使用一个包来实现读寄存器命令，在第一个包传输结束后，master 如果不产生 STOP 事件，而是直接产生 RESTART 事件，后面继续跟地址和数据，这样两个包就变成了一个包传输，I2C slave 接口端也要支持这种方式。

为了简化处理，不管是访问策略寄存器还是 ID 寄存器都返回 4 个字节的响应数据，并且每个字节传输的数据对应寄存器的字节位置如下面表格 — 第二个数据包的 Read Data Cycle 所示。

command byte	
位	含义
[7]	命令定义，1'b1，读寄存器命令
[6:0]	要访问的寄存器地址，地址 DW 对界

一个完整的 I2C 读寄存器包的传输格式如下：

第一个包传输格式

1 st Cycle			
START	7 6 5 4 3 2 1	0	ACK/NAK
S	Slave Address[7:1]	Read/Write Bit 0=Write	A

Command Cycle		
7 6 5 4 3 2 1 0	ACK/NAK	STOP
Command Byte	A	P

第二个包传输格式

1 st Cycle			
START	7 6 5 4 3 2 1	0	ACK/NAK
S	Slave Address[7:1]	Read/Write Bit 1=Read	A

Read Data Cycle								
76543210	ACK/NAK	76543210	ACK/NAK	76543210	ACK/NAK	76543210	ACK/NAK	STOP
Register Byte 3	A	Register Byte 2	A	Register Byte 1	A	Register Byte 0	A	P

from master to slave
 from slave to master

3.5 I2C 管控模块接口

I2C slave 接口模块的对外接口，除了有和 I2C master 连接的 SCL 和 SDA 信号外，还有和 PAB，USB 控制器，GMAC 控制器和 SATA 控制器的寄存器读取接口信号，以及上电复位自动加载模块对策略寄存器的写通路。

接口定义如下：

信号	方向	含义
时钟复位		
Clk	输入	I2C 管控接口核心工作时钟，频率 100MHz，和 PAB、USB、GMAC、SATA 的寄存器访问时钟同频，和自加载模块也同频。
Rst_N	输入	I2C 管控接口核心复位
和安全处理器接口		
SCLi	输入	I2C 输入时钟
SDAi	输入	I2C 输入数据
SDAo	输出	I2C 输出数据
和设备接口		
IIC2PAB_USB_DVEN	输出	I2C 管控接口输出给 PAB 桥 DVEN 寄存器[5]位

		USB 使能, 电平信号, 高电平表示使能, 低电平表示不使能
IIC2PAB_SATA_DVEN	输出	I2C 管控接口输出给 PAB 桥 DVEN 寄存器[11] 位 SATA 使能, 电平信号, 高电平表示使能, 低电平表示不使能
IIC2PAB_GMAC0_DVEN	输出	I2C 管控接口输出给 PAB 桥 DVEN 寄存器[12] 位 GMAC0 使能, 电平信号, 高电平表示使能, 低电平表示不使能
IIC2PAB_GMAC1_DVEN	输出	I2C 管控接口输出给 PAB 桥 DVEN 寄存器[13] 位 GMAC1 使能, 电平信号, 高电平表示使能, 低电平表示不使能
IIC2USB_RQVALID	输出	对 USB 寄存器访问请求有效信号, 寄存器输出, 一拍脉冲
IIC2SATA_RQVALID	输出	对 SATA 寄存器访问请求有效信号, 寄存器输出, 一拍脉冲
IIC2GMAC0_RQVALID	输出	对 GMAC0 寄存器访问请求有效信号, 寄存器输出, 一拍脉冲
IIC2GMAC1_RQVALID	输出	对 GMAC1 寄存器访问请求有效信号, 寄存器输出, 一拍脉冲
IIC2DEV_TYPE	输出	读还是写寄存器标志, 0-Write, 1-Read
IIC2DEV_ADDR[6:0]	输出	寄存器地址
IIC2DEV_WDATA[7:0]	输出	写数据
USB2IIC_RDATA[31:0]	输入	IIC2USB_RQVALID 有效下一拍, USB 寄存器返回读响应数据, 除了 ID 和 Serial Number 寄存器, 有效数据在字节 0
SATA2IIC_RDATA[7:0]	输入	IIC2SATA_RQVALID 有效下一拍, SATA 寄存器返回读响应数据
GMAC02IIC_RDATA[7:0]	输入	IIC2GMAC0_RQVALID 有效下一拍, GMAC0 寄存器返回读响应数据
GMAC12IIC_RDATA[7:0]	输入	IIC2GMAC1_RQVALID 有效下一拍, GMAC1 寄存器返回读响应数据
和复位自加载启动模块接口		
LOAD_RQVALID	输入	上电复位自动加载模块给出的写寄存器请求有效信号, 脉冲触发, 一拍有效
LOAD_ADDR[5:0]	输入	写寄存器地址, 除了 USB 的 ID 和 Serial Number 信息寄存器
LOAD_WDATA[7:0]	输入	写数据
LOAD_I2C_SPEED_SEL	输入	0-选择 standard 速率模式, 1-选择 fast 速率模式
IIC_ADDR[2:0]	输入	I2C Slave 地址的低 3 位输入

3.6 约定说明

安全处理器的 I2C 接口除了接 IO 芯片的管控，还挂接其它的 I2C 芯片：

- 1) 安全 IO 芯片的 I2C 管控接口只支持 7 位地址模式。
- 2) 安全 IO 芯片既支持 **fast mode** 速率模式，也支持 **standard** 速率模式，由自引导固件输出的值选择当前使用的速率模式。
- 3) 安全 IO 芯片的 I2C 管控接口定义一个 **slave address** 地址，要求安全处理器的 I2C master 驱动配置 **IC_TAR** 寄存器为该地址值，该 **slave address** 首先确保和已知的要挂接的其他 I2C 设备地址不冲突，且固件可配置保留三位地址可以进行修改。
- 4) 安全处理器的 I2C 接口支持 **RESTART** 使能，即允许读请求的读地址和读响应包合并成一个包，安全 IO 芯片的 I2C 接口需要支持该功能。
- 5) 在 USB 设备移除后，如果新接入一个设备，管控核心需要重新通过 I2C 管控接口对管控策略寄存器进行配置，即使新接入的设备信息和策略同之前的设备相同。

安全处理器的 **IC_USE_COUNTS** 配置值是 0 还是 1？如果是 1，代表使用 **HCNT** 和 **LCNT** 确定时钟周期配置值，如果是 0，代表直接使用配置参数 **IC_CLOCK_PERIOD** 的值作为 **ic_clk**。

fast mode **IC_CLK_PERIOD=100ns**? **ic_clk=10MHz**

standard mode **IC_CLK_PERIOD=100ns**? IP 生成，还是 500ns？

地址 7'b1010xxx 和 7'b0011xxx 不能使用，否则与内存条地址冲突。

4 SPI 接口

实现一个 Wishbone 接口的 SPI 控制器, SPI 控制器对 Wishbone 交叉开关是一个 Wishbone Slave 端口, 对 Flash 芯片是一个 SPI Master 端口。

通过 Wishbone 交叉开关对 SPI 接口的访问来源主要有:

- JTAG 接口对 SPI 接口的维护访问, 包括 Wishbone 接口的读和写的操作;
- JTAG 接口对 SPI 接口的编程访问, 包括 Wishbone 接口的自定义编程包的读和写操作;
- 系统通过 APB2WB 桥对 SPI 接口的访问, 包括 Wishbone 接口的读和写操作;
- 自启动加载模块对 SPI 接口的访问, 包括 Wishbone 接口的读操作。

EEPROM 和 Flash 芯片的主要区别在于, 首先 EEPROM 的容量一般比较小, 通常都使用 16 位地址 (少数也有使用 24 位地址), 容量是 KB 级的; Flash 的容量相比 EEPROM 更大一些, 通常使用 24 位地址 (现在也有使用 32 位地址的大容量 Flash), 容量是 MB 级的。

其次, EEPROM 一般存放各种设置参数, 系统运行过程中可改变数据并可保存; Flash 一般存放运行代码, 系统运行过程中不可改变, 只能进行读操作, 只有非运行过程中可编程。

4.1 功能描述

4.1.1 概述

DW_apb_ssi 是一个可配置, 可综合, 可编程的全双工 master 或 slave 串行接口, 系统通过 APB 接口对 DW_apb_ssi 进行访问, 包括数据、控制和状态的读写。DW_apb_ssi 可以配置成两种操作模式: 串行 master 和串行 slave 外围设备, 并且以下面的任意一种接口形式:

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

上述三种串行协议类似, 我们的设计中选用的都是 SPI 接口的芯片, 所以基本上只考虑配置成 SPI 接口的 SSI 控制器, 在寄存器 CTRLR0 的 FRF (frame format) 域可以对选用支持的串行协议进行编程配置, 详见寄存器定义。

安全 IO 芯片的 SPI 接口是实现在 wishbone 总线上的 master 接口, 所以需要配置成 master 模式, 还需要把系统访问的 APB 接口修改为 wishbone 接口。

4.1.2 时钟

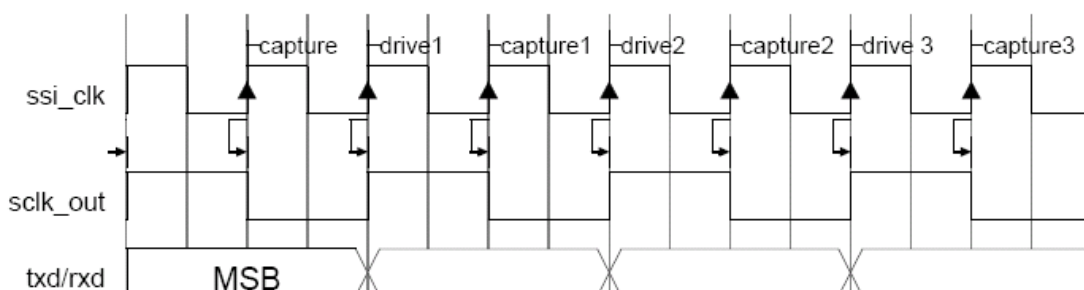
SSI 控制器的串行输入时钟 ssi_clk 频率不能大于系统访问接口时钟频率 (pclk 或 wb_clk), 保证控制信号从 ssi_clk 时钟域到 wb_clk 时钟域的同步。如果 wb_clk 和 ssi_clk 是异步的, 就需要配置 IP 的时候选择生成同步逻辑; 如果 wb_clk 和 ssi_clk 是同步的, 就不需要生成同步逻辑, 减少延迟。

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

$$F_{sclk_out} = F_{ssi_clk} / SCKDV$$

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

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



4.1.3 发送和接收 FIFO

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

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

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

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

4.1.4 SSI 中断

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

- 1) 发送 FIFO 空中断（ssi_txe_intr）
- 2) 发送 FIFO 上溢中断（ssi_txo_intr）
- 3) 接收 FIFO 满中断（ssi_rxf_intr）
- 4) 接收 FIFO 上溢中断（ssi_rxo_intr）
- 5) 接收 FIFO 下溢中断（ssi_rxu_intr）
- 6) 多 master 冲突中断（ssi_mst_intr）：由于只有一个 master，不会使用该中断。
- 7) 合成中断请求（ssi_intr）：上述几种中断请求的或。

4.1.5 传输模式

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

1) 发送和接收

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

2) 只发送

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

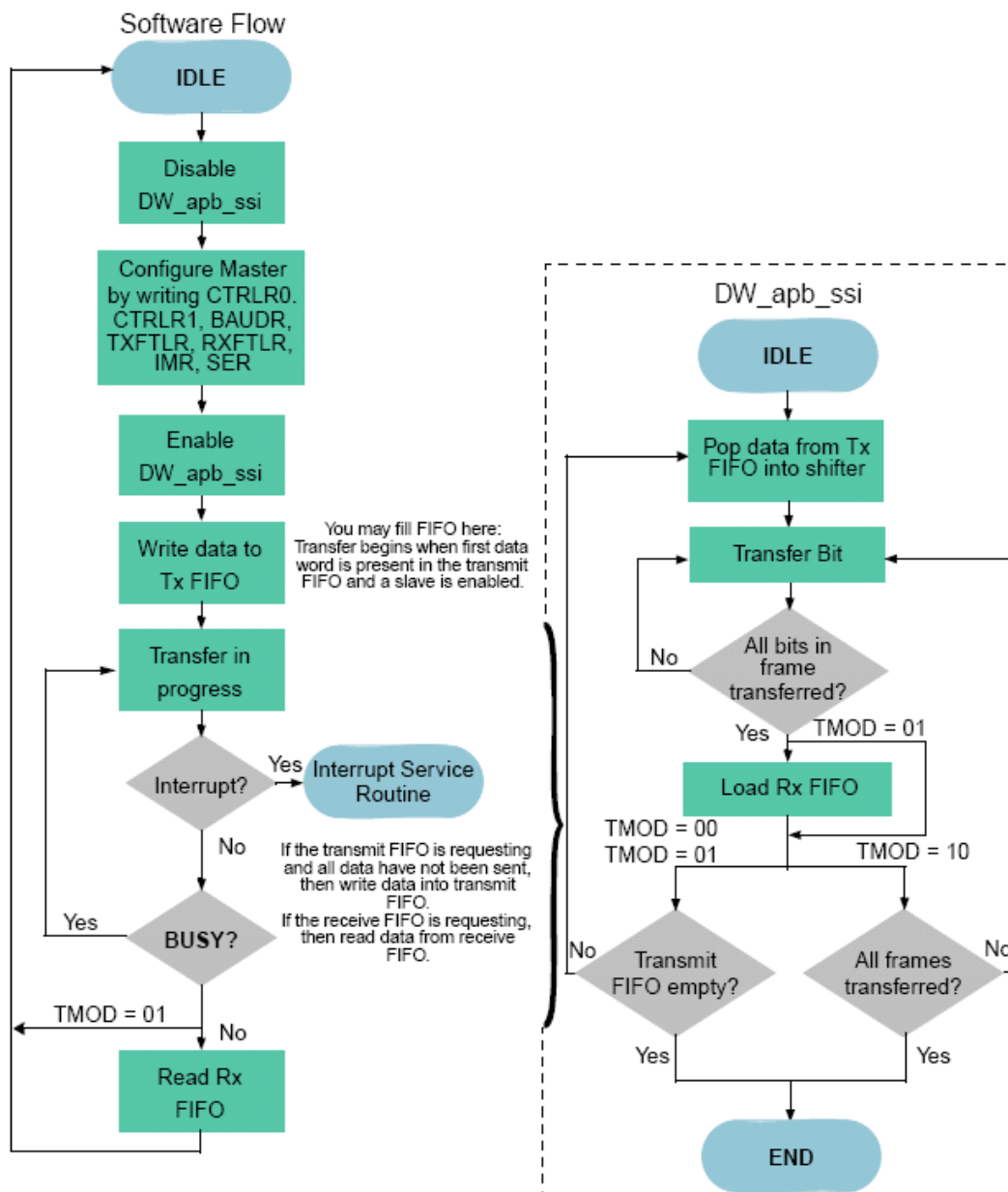
3) 只接收

TMOD=2'b10，只有接收逻辑有效。

4) EEPROM 读

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

4.1.6 操作模式



4.2 参数定义

配置项	参数描述	配置值
APB Data Bus Width	APB_DATA_WIDTH 8,16,32	32
Serial master or slave configuration	SSI_IS_MASTER 0-slave 1-master	1
Receive FIFO buffer depth	SSI_RX_FIFO_DEPTH 2 to 256	256

Transmit FIFO buffer depth	SSI_TX_FIFO_DEPTH 2 to 256	256
Number of slave select lines	SSI_NUM_SLAVES 1 to 16	1
Include Programmable RXD Sample Logic	SSI_HAS_RX_SAMPLE_DELAY 1-true 0-false	1
Maximum RXD Sample Delay	SSI_RX_DLY_SR_DEPTH 4 to 255	64
Configure interrupt pin out	SSI_INTR_IO 0- 分别输出中断信号 1- 中断信号或输出	0
Active interrupt level	SSI_INTR_POL 0- 低有效 1- 高有效	0
Are pclk and ssi_clk synchronous?	SSI_SYNC_CLK 1-true 0-false	0
Generate clock enable input for ssi_clk?	SSI_CLK_EN_MODE 1-true 0-false 如果使能, ssi_clk_en 信号会控制 ssi_clk 的翻转; 如果不使能, ssi_clk 时钟始终有效。	0
Hard code the frame format?	SSI_HC_FRF 1-true 0-false 如果为 1, 代表硬件固定了串行协议的类型, 不能通过对寄存器的写修改接口协议类型; 如果为 0, 则代表软件可以修改串行协议类型。	0
Default frame format	SSI_DFLT_FRF 0x0-SPI 0x1-SSP 0x2-Microwire	0x0
Default serial clock polarity	SSI_DFLT_SCPOL 0- 非工作状态时钟为低 1- 非工作状态时钟为高	0
Default serial clock phase	SSI_DFLT_SCPH 具体含义见寄存器 CTRLR0 描述	0
TX_ABW	发送 FIFO 地址宽度, 根据 SSI_TX_FIFO_DEPTH 值确定	8
RX_ABW	接收 FIFO 地址宽度, 根据 SSI_RX_FIFO_DEPTH 值确定	8

4.3 信号描述

信号	方向	含义
时钟和复位信号		
ssi_clk	输入	控制器核心时钟
ssi_rst_n	输入	控制器核心复位
与 wishbone 交叉开关接口信号		
wb_clk	输入	时钟
wb_rst_n	输入	复位
wb_dat_i[31:0]	输入	读数据
wb_dat_o[31:0]	输出	写数据
wb_ack_o	输出	周期结束响应
wb_adr_i[31:0]	输入	总线地址
wb_cyc_i	输入	总线请求/继续标志
wb_sel_i[3:0]	输入	字节使能
wb_stb_i	输入	请求开始总线周期
wb_we_i	输入	写使能
wb_err_o	输出	总线错误标志
与 flash 芯片接口信号		
sclk_out	输出	给芯片的串行时钟
cs_n/ss_0_n	输出	输出使能
mosi/txd	输出	ssi 控制器输出, flash 芯片输入的数据
miso/rxd	输入	flash 芯片输出, ssi 控制器输入的数据
hold_n	输出	保持, 低有效, 不使用, 固定接成“1”
wp_n	输出	保护信号, 低有效, 不使用, 固定接成“1”
其他信号		
ssi_intr_n	输出	
ssi_txe_intr_n		
ssi_txo_intr_n		
ssi_rxf_intr_n		
ssi_rxo_intr_n		
ssi_rxu_intr_n		

4.4 寄存器定义

SSI 控制器的寄存器地址是 4B 对齐访问, 寄存器定义如下 (只列出使用的):

名称	偏移地址	宽度	描述	复位值
CTRLR0	0x0	16 bits	控制寄存器 0	0x7
CTRLR1	0x4	16 bits	控制寄存器 1	0x0
SSIENR	0x8	1 bit	SSI 控制器使能寄存器	0x0
SER	0x10	1 bit	slave 使能寄存器	0x0

BAUDR	0x14	16 bits	波特率选择	0x0
TXFTLR	0x18	TX_ABW	发送 FIFO 阈值	0x0
RXFTLR	0x1c	RX_ABW	接收 FIFO 阈值	0x0
TXFLR	0x20	TX_ABW+1	发送 FIFO 有效条目数	0x0
RXFLR	0x24	RX_ABW+1	接收 FIFO 有效条目数	0x0
SR	0x28	7 bits	状态寄存器	0x6
IMR	0x2c	6 bits	中断屏蔽寄存器	0x0
ISR	0x30	6 bits	中断状态寄存器	0x0
RISR	0x34	6 bits	原生中断状态寄存器	0x0
TXOICR	0x38	1 bit	发送 FIFO 上溢中断清除寄存器	0x0
RXOICR	0x3c	1 bit	接收 FIFO 上溢中断清除寄存器	0x0
RXUICR	0x40	1 bit	接收 FIFO 下溢中断清除寄存器	0x0
ICR	0x48	1 bit	中断清除寄存器	0x0
DR	0x60-0xec	16 bits	数据寄存器	0x0
RS_SAMPLE_DLY	0xf0	8 bits	接收数据采样延迟	0x0

寄存器 CTRLR0

位	名称	属性	描述	复位值
15:12	CFS	R/W	控制帧大小，在 Microwire 协议中才会使用	0x0
11	SRL	R/W	移位寄存器回环 0— 正常模式操作 1— 测试模式操作	0x0
10	SLV_OE	R/W	配置成 slave 模式才使用	0x0
9:8	TMOD	R/W	传输模式 00— Transmit & Receive 01— Transmit Only 10— Receive Only 11— EEPROM Read	0x0
7	SCPOL	R/W	串行时钟极性 0— 串行时钟在不工作状态为低 1— 串行时钟在不工作状态为高	0x0
6	SCPH	R/W	串行时钟相位，选择串行时钟和 slave select 信号之间的相位关系。当 SCPH=0 时，串行时钟的第一个时钟沿就开始捕获数据；当 SCPH=1 时，slave select 有效后串行时钟进行一个周期的翻转，在串行时钟的第二个时钟沿开始捕获数据。 0: 串行时钟在第一个数据 bit 的中间翻转 1: 串行时钟在第一个数据 bit 的	0x0

			开始翻转	
5:4	FRF	R/W	帧格式，选择串行协议 00— Motorola SPI 01— Texas Instruments SSP 10 — National Semiconductors Microwire 11— 保留	0x0
3:0	DFS	R/W	数据帧大小 0000~0010 未使用 0011— 4-bit 串行数据传输 0100— 5-bit 串行数据传输 0101— 6-bit 串行数据传输 0110— 7-bit 串行数据传输 0111— 8-bit 串行数据传输 1000— 9-bit 串行数据传输 1001— 10-bit 串行数据传输 1010— 11-bit 串行数据传输 1011— 12-bit 串行数据传输 1100— 13-bit 串行数据传输 1101— 14-bit 串行数据传输 1110— 15-bit 串行数据传输 1111— 16-bit 串行数据传输	0x7

寄存器 CTRLR1

位	名称	属性	描述	复位值
15:0	NDF	R/W	数据帧的数目。当 TMOD=10 或 11 时，即配置成 receive only 模式或 eeprom read 模式时，确定 SSI 控制器需要接收的数据帧的数目，NDF+1 就是一次传输接收的数据帧个数。	0x0

寄存器 SR

位	名称	属性	描述	复位值
6	DCOL	R	数据冲突错误，多 master 才会使用	0x0
5	TXE	R	传输错误，配置成 slave 模式才使用	0x0
4	RFF	R	接收 FIFO 满	0x0
3	RFNE	R	接收 FIFO 非空	0x0
2	TFE	R	发送 FIFO 空	0x1
1	TFNF	R	发送 FIFO 未滿	0x1
0	BUSY	R	SSI 忙标志	0x0

寄存器 DR

位	名称	属性	描述	复位值
15:0	DR	R/W	数据寄存器 Read = Receive FIFO buffer Write = Transmit FIFO buffer	0x0

寄存器 RX_SAMPLE_DLY

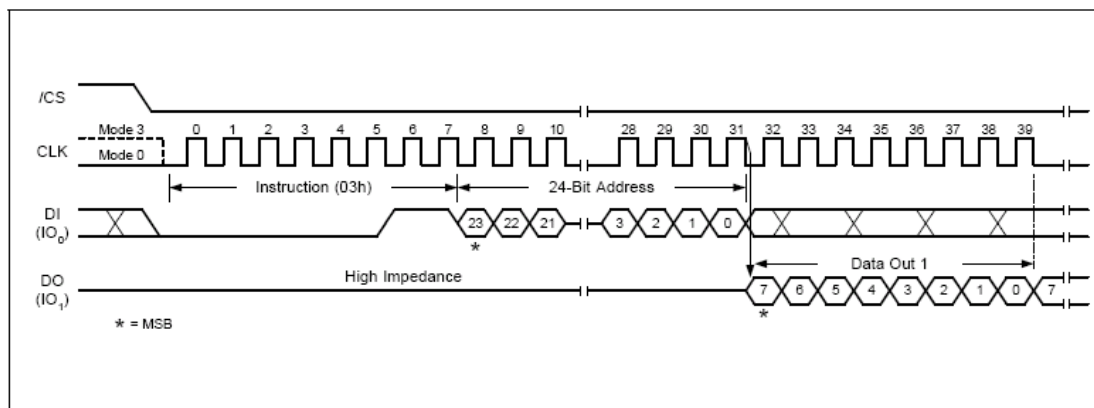
位	名称	属性	描述	复位值
15:0	RSD	R/W	接收数据（rxdata）采样延迟，用于控制外部串行输入数据的采样延迟，设定的值即为 rxdata 信号采样延迟的 ssi_clk 数目。如果 RSD 的设定值大于 移位寄存器的深度（SSI_RX_DLY_SR_DEPTH），那么相当于没有延迟。 注：该信号有效的前提是 IP 配置的时候选择了生成采样延迟逻辑，即参 数 SSI_HAS_RX_SAMPLE_DELAY=1.	0x0

设置接收数据采样延迟的原因是，SSI 控制器先会把命令的操作码，然后是地址和数据输出到串行总线，首先不清楚发出去的数据需要多少个时钟周期，其次不清楚外部芯片返回的数据延迟，这时就需要接受采样延迟的加入来控制真正的数据的收集时刻。

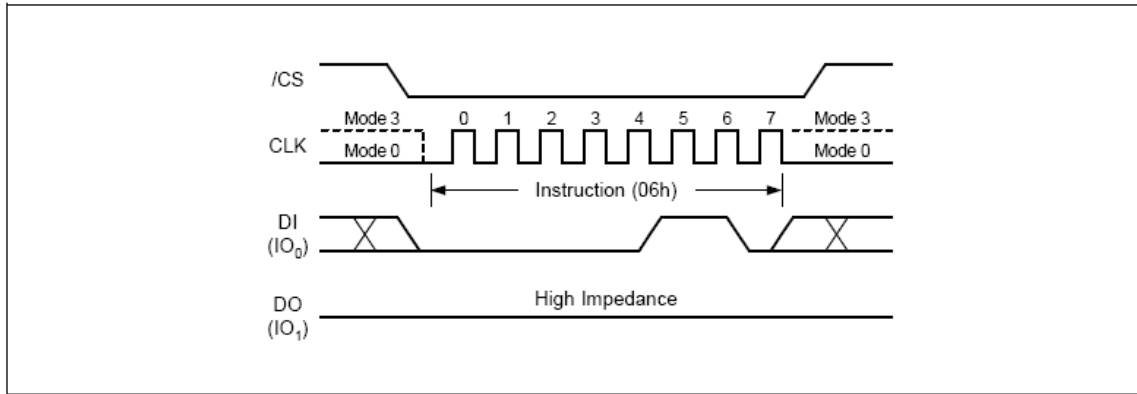
4.5 工作流程

4.5.1 Flash 访问命令

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

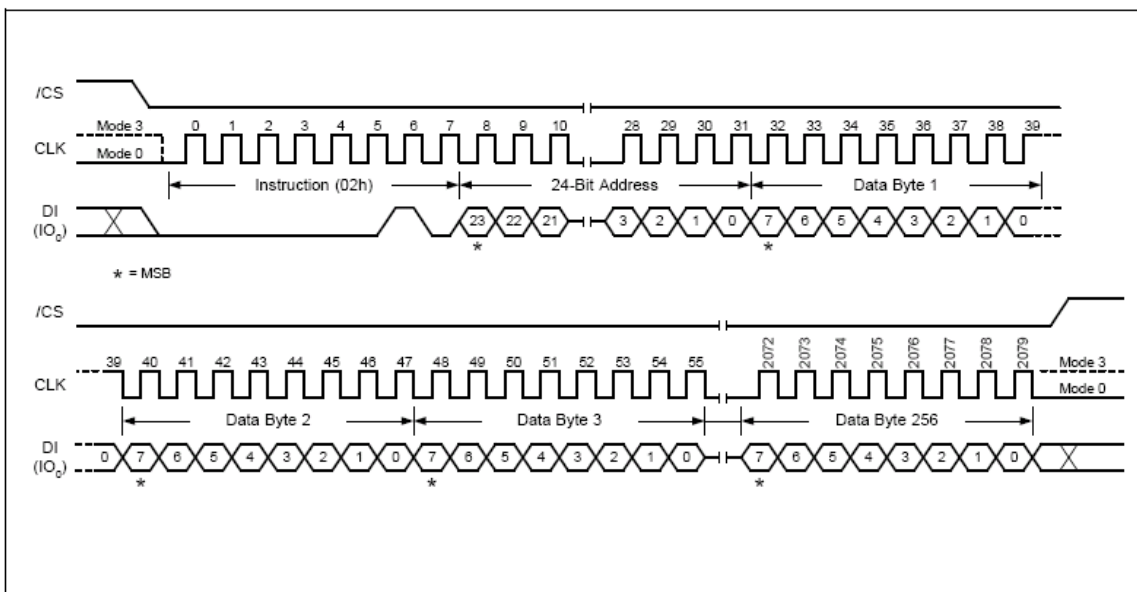


读数据命令序列（Read）



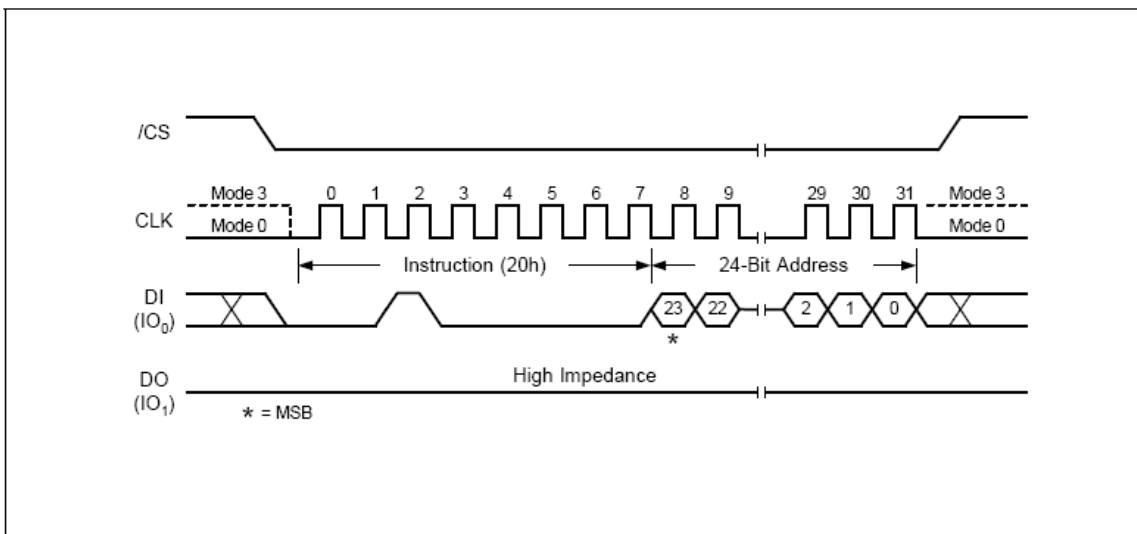
写使能命令序列 (Write Enable)

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



页编程命令序列 (Page Program)

支持一次进行 1~256 字节的数据编程。

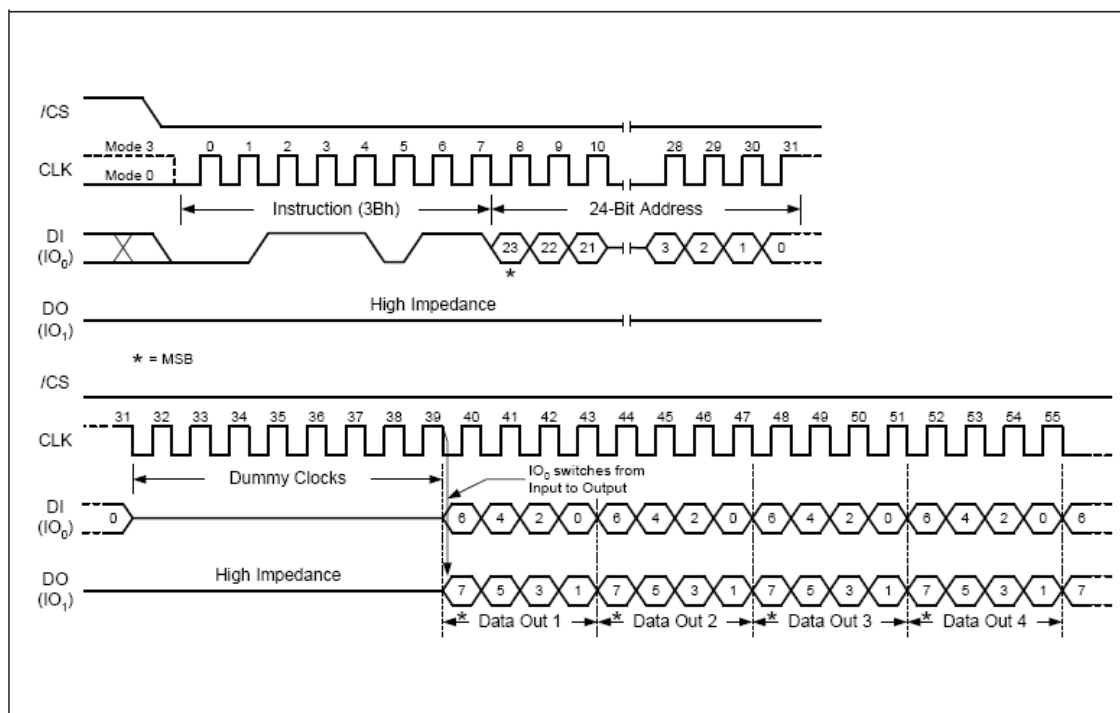


扇区擦除命令序列 (Sector Erase)

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

再以一个 xilinx 的 flash 模型为例，Read 命令操作码和序列同 winbond 芯片一致，但是写操作有不同，不需要执行写使能，首先执行命令 Buffer 1 Write，把要编程的数据写入 flash 芯片的一个临时缓冲，然后执行命令 Buffer 1 To Main Memory Page Prog With Erase，把临时缓冲中的数据真正写入存储器，并且写入前自动进行擦除操作。

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



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

4.5.2 编程流程

按照 SSI 控制器操作模式的描述，启动编程过程的软件寄存器读写过程如下：

序号	操作
1	disable SSI 控制器，写'h8 地址 SSIENR 寄存器值为 0x0
2	配置 TMOD 为 transmit only 模式，写'h0 地址 CTRLR0 寄存器值为 0x1c7，配置 SCPH 和 SCPOL 也为 1，有的芯片只支持 MODE3，即前面两个值都为“1”
3	配置波特率，写'h14 地址 BAUD 寄存器值为 2
4	配置发送 FIFO 阈值，写'h18 地址 TXFTLR 寄存器值为 0x4
5	配置接收 FIFO 阈值，写'h1c 地址 RXFTLR 寄存器值为 0x4
6	使能 slave 设备，写'h10 地址 SER 寄存器值为 0x1

7	enable SSI 控制器，写'h8 地址 SSIENR 寄存器值为 0x1
8	发送写使能命令，写'h60 地址 DR 寄存器值为 0x6
9	写'h10 地址 SER 寄存器值为 0x0
10	发送擦除命令操作码，写'h60 地址 DR 寄存器值为 0x20
11	发送擦除命令高段地址[23:16]，写'h64 地址 DR 寄存器
12	发送擦除命令中段地址[15:8]，写'h68 地址 DR 寄存器
13	发送擦除命令低段地址[7:0]，写'h6c 地址 DR 寄存器
14	写'h10 地址 SER 寄存器值为 0x1
等待擦除完毕	
15	发送写使能命令，写'h60 地址 DR 寄存器值为 0x6
16	写'h10 地址 SER 寄存器值为 0x0
17	发送编程命令操作码，写'h60 地址 DR 寄存器值为 0x2
18	发送编程命令高段地址[23:16]，写'h64 地址 DR 寄存器
19	发送编程命令中段地址[15:8]，写'h68 地址 DR 寄存器
20	发送编程命令低段地址[7:0]，写'h6c 地址 DR 寄存器
.....	从'h70 地址寄存器开始到'hec，再从'h60 到'hec 地址寄存器，依次写入希望编程的数据
	写'h10 地址 SER 寄存器值为 0x1
等待编程数据写入结束	

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

对 SER 寄存器反复进行写 0 和写 1 操作的原因是：如果系统接口的速率很低，那么如果在 DR 寄存器的 instruction、address、data 没全部布好的情况下就打开 slave enable，可能会导致发送数据错误。所以等待所有的数据都写到 DR 寄存器后，再打开 slave enable 开始发送控制和数据，虽然性能会稍差，但是保证了正确性。而且 IP 生成的时候可以配置发送和接收 FIFO 的数量分别是最大值 -256，这样可以循环不断的往 0x60~0xec 编址的数据寄存器发送数据，写到 DR 寄存器的数据会进入 256 个条目的发送 FIFO，不会存在数据覆盖的问题。

4.5.3 EEPROM 读流程

序号	操作
1	disable SSI 控制器，写'h8 地址 SSIENR 寄存器值为 0x0
2	配置 TMOD 为 eeprom read 模式，写'h0 地址 CTRLR0 寄存器值为 0x3c7，配置 SCPH 和 SCPOL 也为 1，有的芯片只支持 MODE3，即前面两个值都为“1”
3	配置波特率，写'h14 地址 BAUD 寄存器值为 2
4	配置发送 FIFO 阈值，写'h18 地址 TXFTLR 寄存器值为 0x4
5	配置接收 FIFO 阈值，写'h1c 地址 RXFTLR 寄存器值为 0x4
6	配置接收数据帧数目 NDF，写'h4 地址 CTRLR1 寄存器值为 ndf，接收的数据帧的数目从发送完 8-bit 操作码和 16-bit 地址后，间隔一个 8bit 串行数据收集时间后

	就开始增加。ndf 的具体设置值见下面的详细说明。
7	配置接收采样延迟，写'hf0 地址 RSD 寄存器值为 0x0 或 0x10，如果随命令后发送 16-bit 地址，则写入 0x10，如果发送 24-bit 地址，则写入 0x0，原因见下面的说明。
8	不使能 slave 设备，写'h10 地址 SER 寄存器值为 0x0
9	enable SSI 控制器，写'h8 地址 SSIENR 寄存器值为 0x1
10	发送读数据命令操作码，写'h60 地址 DR 寄存器值为 0x3
11	发送读数据命令高段地址[23:16]，写'h64 地址 DR 寄存器
12	发送读数据命令中段地址[15:8]，写'h68 地址 DR 寄存器
13	发送读数据命令低段地址[7:0]，写'h6c 地址 DR 寄存器
14	使能 slave 设备，写'h10 地址 SER 寄存器值为 0x1
等待读响应数据返回	
15	读地址'h34 的原生中断状态寄存器的[4]位，判断是否为“1”，如果为 1，则代表接收 FIFO 已经收到接收阈值数目以上的接收数据，可以启动系统总线读操作，连续的读'h60~'hec 地址数据寄存器的接收数据，直到该位始终保持为“0”，代表已经读完 ndf 或 ndf-1 个数据帧。

从 SPI 总线上接收输入数据写入接收缓冲：i_load_rx_buf 有效是写入条件

DW_wb_ssi_shift.v

```
assign i_load_rx_buf = (ssi_en == 1'b0) ? 1'b0 : (~SSI_HAS_RX_SAMPLE_DELAY==0 ||
rx_sample_dly==0 || rx_sample_dly > `SSI_RX_DLY_SR_DEPTH) ? load_rx_buf :
|((1'b1<<(rx_sample_dly-1)) & load_rx_buf_dly_sr);
```

DW_wb_ssi_mstfsm.v

```
assign load_rx_buf = (tmod == 2'b10 || tmod == 2'b00 || mwire ||
(tmod == 2'b11 && ((scph == 1'b1 && spi1_control == 1'b0) || (scph == 1'b0 && spi0_control ==
1'b0)))) ? (c_state==MWPOP && (sclk_re_int || baud2) && srl_test)||((c_state==PUSH_RX &&
sclk_re_int)|| (c_state==TOGGLE_SS && (sclk_re_int || baud2) && (ssp || mwire && srl_test))||
(c_state==HOLD_MASK && (sclk_re_int || baud2) && (!mwire || mwire && srl_test))||((c_state==MASK_SCLK && !spi0 && sclk_re_int && !mwire) : 1'b0;
```

EEPROM read 模式因为确定传输的前 3 个字节分别是 opcode, addr[15:8], addr[7:0]，所以 tmod==2'b11 加入了条件 scph 和 spi1_control（发送缓冲非空）的判断，即确保控制器把 3 个字节的数据放到 SPI 总线上才会使能 FIFO 接收，但是还需要 8 个 ssi_clk 才能把第一个字节的输入数据存下来，所以还是需要加入 rx_sample_dly（设置成 16，因为波特率为 2，即控制器的 ssi_clk 是输出给 flash 的时钟频率的 2 倍）。NDF 的值同样也需要加 1。前面这样设置的前提是 flash 只支持 16 位地址，如果是支持 24 位地址，那么 eeprom read 的命令依然是 1 个字节的 opcode+3 个字节的地址，这样就不需要设置 rx_sample_dly，或写成 0。

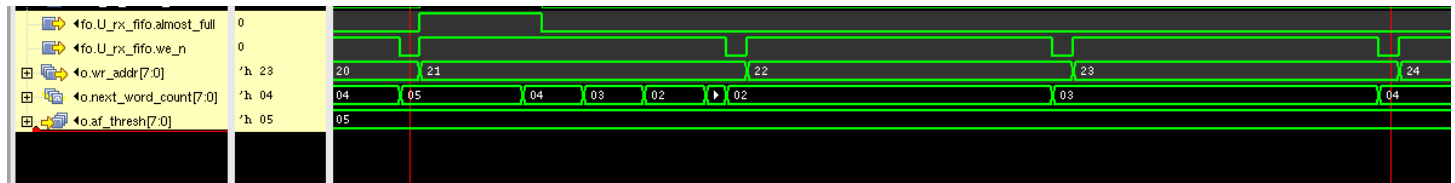
其他的非 EEPROM read 模式，由于输出数据的字节不定，没有办法进行延迟判断，所以都是在发出第一个字节的数据后就置接收有效。所以每次需要根据发送的控制字节数（操作码和地址）的不同修改 rx_sample_dly 的值（设置成 64，在 eeprom read 的基础上需要增加 3 字节的地址）。

关于 NDF 值的设置说明如下：

- 1) 如果是 24 位地址的 flash：写 DR (0x60~0x6c)，发送 1 个字节读命令+3 个字节地址，不需要配置 rx_sample_dly；写 DR (0x60~0x68)，发送 1 个字节读命令+2 个字节地址，需要配置 rx_sample_dly，因为 flash 应该是按照接收完 24 位地址才会返回响应数据。

2) 如果是16位地址的flash: 写DR (0x60~0x68), 发送1个字节读命令+2个字节地址, 也不需要配置rx_sample_dly, ndf设置值为期望值 (和手册不一致); 写DR (0x60~0x6c), 发送1个字节读命令+3个字节地址, 应该会漏掉第一个字节的数据。

因为控制器确定是不是准备接收第一个返回字节数据的条件是判断发送FIFO的数据都发送完, 1个字节的命令+2/3个字节的地址, 在发送完之后过8个ssi_clk才会把收到的第一个字节数据写入接收FIFO。所以16位地址的flash如果用24位地址访问, 会丢掉第一个字节数据。



假设ndf=35, 即期望接收36个字节的数据, rx_threshold接收FIFO阈值设置成4, 控制器最后一个写入存储器的数据是地址0x23条目的数据, 0x0~0x23正好是36个字节的数据没有问题。但是rx fifo full的生成条件如下: DW_wb_ssi_bcm06.v

```
assign next_almost_full = (next_word_count >= af_thresh_val[ADDR_WIDTH-1:0])? 1'b1 :
    next_full;
```

就是说, 即使最后4个字节的数据已经写入了FIFO, 但是next_word_count等于4, 没有变成threshold+1=5的值, full信号就会一直无效。而next_word_count再加1的条件是必须还有一个FIFO条目写入。

所以ndf设置值应该是期望值, 而不是期望值减1, 造成的结果是多收下一个字节数据, 但是可以不读出。否则最后4个字节数据无法读出。

而且上面的判断还涉及一个threshold的值, 如果期望读出的字节数不是threshold的整数倍, 也要在ndf的值上加x到整数倍。这样才能满足full的条件。

4.5.4 正常工作读写流程

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

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

而transmit & receive模式接收数据的数目是按照发送FIFO的有效数目来确定, 只有发送FIFO持续的有数据发到串行总线上, 才会有持续的sclk_out, 以此来控制接收数据的帧数。即发送完操作码和地址后, 还需要持续的发送期望接收帧数目的数据帧到串行总线上。接收FIFO写入的条目数和发送FIFO的发送条目数相同, 即控制器真正收到的数据帧数=期望数据帧数+1个操作码+3个地址码, 即期望数+4。非期望数据的忽略通过采样延迟配置成0x40即可把前面的64拍ssi_clk(4*8=32个sclk_out时钟周期)时钟周期的等待接收数据时间去掉。

考虑写操作的执行时间比较长, 而且读操作和写操作的传输模式也不同, 需要分别配置, 所以尽量避免读写交叉, 减少切换时间。

5 GPIO 接口

实现系统可见的 GPIO 接口，实现 2 组，每组各 8 根输入/输出线。采用 Synopsys 的 GPIO IP 实现需要的功能（额外增加按位写的操作）。DW_apb_gpio 控制输出数据和 I/O pad 的方向，也可以使用寄存器读回 I/O pad 的数据。有两种方法可以实现输入输出数据和控制信号源的选择，通过 APB 总线访问的软件控制方法，和通过辅助硬件接口信号的硬件控制方法。

DW_apb_gpio 可以最多配置出 4 个 port，每个 port 可分别配置成 8bit~32bit 的数据线，安全 IO 芯片的设计只需要配置 2 个 port，portA 和 portB，每个 port 配置出 8 根线，采用软件控制的方法实现输入输出数据和方向的控制。

GPIO 在功能使用上希望可以按位控制，而不是现在的按字节控制，但是由于 AHB/APB 总线的特性，最小粒度也只能做到按字节控制写操作。可以增加对应 8 位数据的屏蔽码，也按照数据写入数据寄存器的高位，GPIO 内部根据 8 位数据和 8 位屏蔽码进行对应位的修改。寄存器定义见 5.2 节，修改 portA 和 portB 的 dr 寄存器的定义，除了数据增加屏蔽码，并且数据和屏蔽码同拍写入。

修改 DW_apb_gpio_apbif.v 文件，int_gpio_swporta_dr 和 int_gpio_swportb_dr 的生成逻辑需要进行修改，按照 pwwdata[15:8] 的每一位的值分别对 pwwdata[7:0] 的值是否写入新值还是保持原值进行二选一。

输入输出方向控制不需要进行按位写，方向属性只会在初始化时静态配置好，正常运行过程中不再需要重新配置。复位时可以都配置成输出方向，在自引导加载中可以配置输入输出方向。

2 组 8 位的 GPIO 信号用途，板级会利用 GPIO 进行一些控制，安全 IO 芯片也可以利用 GPIO 输出管脚把一些状态监测等关键信号拉到板级进行点灯，考虑 GPIO 管脚复用。

5.1 IP 配置参数

参数名称	参数说明	设定值
顶层参数		
Add encoded parameters	是否需要将 GPIO 配置信息登记到寄存器 0：不登记 1：登记配置信息到 gpio_config_reg 寄存器	1
APB Data Bus Width	APB 数据总线宽度	32 位
Number of Ports	GPIO 分组端口数	2 设置 Port A 和 PortB
DW_apb_gpio ID	是否设置 GPIO ID 寄存器 0：不设置 1：设置	1
DW_apb_gpio ID Width	GPIO ID 寄存器宽度	32 位
DW_apb_gpio ID Value	GPIO ID 值	32'h0
port A 参数		
Port A Width	Port A 数据宽度	8 位
Port A Auxiliary H/W	Port A 是否有硬件 Aux 电源支持 0：没有 Aux 电源 1：有 Aux 电源	0

Port A controlled from single source?	GPIO 控制器模式选择（前提是有 Aux 电源） 0: 只支持软件控制 1: 软/硬件控制都支持	0 不关心
Port A Default Direction	Port A 输入/输出方向默认值 0: 输入 1: 输出	0
Port A Default Mode	如果支持两种模式控制，默认控制模式选择（前提是有 AUX 电源） 0: 软件控制 1: 硬件控制	0 不关心
Port A Debounce Logic	是否需要 debounce 逻辑消除外部输入中断信号的毛刺，需要外部额外输入一个慢时钟 dbclk，小于一个 dbclk 时钟周期的中断信号都会被过滤掉 0: 不需要 1: 需要	0 因为还需要外部额外输入一个时钟，这个逻辑不是很必要
Port A Interrupt	Port A 的信号是否可以作为中断源使用 0: 不做中断源 1: 做中断源	1
Port A Interrupt Polarity	如果做中断源使用 Port A 中断极性 0: 低有效 1: 高有效	1
Single Interrupt Generated?	Port A 可以出一根组合的中断信号，还是每个中断类型单独一根信号 0: 出多根信号 1: 出一根信号	1
Port A Read Back Data Synchronization	从输入信号 gpio_ext_porta 读入的数据是否做同步以消除亚稳态 0: 不做同步 1: 做同步	1
Port A Software Register Reset Value	Port A 8 根信号复位初始值	8'h0
Port A Interrupt Synchronization	Port A 信号作为中断信号使用时，中断是否在 GPIO 内部做同步（用 pclk_intr 时钟打两拍） 0: 不需要 1: 需要	1
port B 参数		
Port B Width	Port B 数据宽度	8 位
Port B Auxiliary H/W	Port B 是否有硬件 Aux 电源支持 0: 没有 Aux 电源 1: 有 Aux 电源	0
Port B controlled from single source?	GPIO 控制器模式选择（前提是有 Aux 电源） 0: 只支持软件控制 1: 软/硬件控制都支持	0 不关心
Port B Default Direction	Port B 输入/输出方向默认值 0: 输入 1: 输出	0
Port B Default Mode	如果支持两种模式控制，默认控制	0

	模式选择（前提是有 AUX 电源） 0：软件控制 1：硬件控制	不关心
Port B Read Back Data Synchronization	从输入信号 gpio_ext_portb 读入的数据是否做同步以消除亚稳态 0：不做同步 1：做同步	1
Port B Software Register Reset Value	Port B 8 根信号复位初始值	8'h0

5.2 接口信号

信号	方向	含义
与 APB 总线接口信号		
pclk	输入	总线时钟
presetn	输入	总线复位
psel	输入	总线选择
paddr[31:0]	输入	总线地址
pdata[31:0]	输入	总线写数据
pwrite	输入	总线读写标志
penable	输入	数据使能
prdata[31:0]	输出	总线读数据
片外接口信号		
gpio_porta_dr[7:0]	输出	GPIO 端口 A 输出数据
gpio_porta_ddr[7:0]	输出	GPIO 端口 A 输入/输出方向
gpio_ext_porta[7:0]	输入	GPIO 端口 A 输入数据
gpio_portb_dr[7:0]	输出	GPIO 端口 B 输出数据
gpio_portb_ddr[7:0]	输出	GPIO 端口 B 输入/输出方向
gpio_ext_portb[7:0]	输入	GPIO 端口 B 输入数据
其他信号		
pclk_intr	输入	GPIO Port A 的信号作为中断使用时，中断信号可以利用 GPIO 自带的逻辑同步到系统时钟 pclk_intr。如果 GPIO 的核心工作时钟 pclk 被门控关闭，系统仍需要接收中断就使用 pclk_intr。通常如果只有当 pclk 工作时控制器才检测中断，就可以把 pclk 和 pclk_intr 这两个时钟接到一个源上。
gpio_intr_flag	输出	GPIO 组合中断输出
gpio_intrclk_en	输出	中断触发类型指示：0-电平出发 1-边沿触发

说明：只有 Port A 的信号可以做中断线使用，Port B 不能做中断线使用。

5.3 寄存器定义

offset[4:0]	Description	有效位数	Default	Type
0x0	gpio_swporta_dr: Port A 数据寄存器 [15:8]: mask 1-对应位需要修改 0-对应位不需要修改 [7:0] : data	16 bits	0x0	RW
0x4	gpio_swporta_ddr: Port A 数据方向寄存器 0— Input 1— Output	8 bits	0x0	RW
0x8	gpio_swporta_ctl: PortA 数据源寄存器 0— 软件控制 1— 硬件控制	8 bits	0x0	RW
0xc	gpio_swportb_dr: Port B 数据寄存器 [15:8]: mask 1-对应位需要修改 0-对应位不需要修改 [7:0] : data	16 bits	0x0	RW
0x10	gpio_swportb_ddr: Port B 数据方向寄存器	8 bits	0x0	RW
0x14	gpio_swportb_ctl: Port B 数据源寄存器	8 bits	0x0	RW
0x30	gpio_inten: 配置 GPIO Port A 的 8 根线的中断使能, 按位配置 0-配置 Port A 的一个 bit 是正常 GPIO 信号 1-配置 Port A 的一个 bit 是中断信号	8 bits	0x0	RW
0x34	gpio_intmask: 按位配置 0-不屏蔽中断位 1-屏蔽中断位	8 bits	0x0	RW
0x38	gpio_inttype_level: 按位配置 0-中断信号是电平触发 1-中断信号是边沿触发	8 bits	0x0	RW
0x3c	gpio_int_polarity: 按位配置 0-中断信号是低有效 1-中断信号是高有效	8 bits	0x0	RW
0x40	gpio_intstatus: Port A 的中断状态	8 bits	0x0	R
0x44	gpio_raw_intstatus: Port A 的原始中断状态, 即不关心是否 mask 的状态	8 bits	0x0	R
0x4c	gpio_porta_eoi: 按位控制 0-不清除中断 1-清除中断	8 bits	0x0	W
0x50	gpio_ext_porta: Port A 外部端口寄存器 Port A 如果配置成 Input, 读取信号 gpio_ext_porta 的数据; 如果 Port A 的数据方向是输出, 则读取 Port A 的数据寄存器的数据。	8 bits	0x0	R
0x54	gpio_ext_portb: Port B 外部端口寄存器	8 bits	0x0	R
0x60	gpio_ls_sync: 是否把电平敏感的中断信号和 pclk_intr 同步 0-不和 pclk_intr 时钟进行同步 1-进行时钟同步	1 bit	0x0	RW
0x74	gpio_config_reg1: 可以读取生成 IP 时选择的配置参数 [31:21]: 保留	32 bits	和 IP 生成配置值相关	R

	[20:16]: ENCODED_ID_WIDTH [15]: GPIO_ID [14]: ADD_ENCODED_PARAMS [13]: DEBOUNCE [12]: PORTA_INTR [11]: HW_PORTD [10]: HW_PORTC [9]: HW_PORTB [8]: HW_PORTA [7]: PORTD_SINGLE_CTL [6]: PORTC_SINGLE_CTL [5]: PORTB_SINGLE_CTL [4]: PORTA_SINGLE_CTL [3:2]: NUM_PORTS [1:0]: APB_DATA_WIDTH			
--	---	--	--	--

如果利用 GPIO Port A 的 4 根线作为板级的 4 个中断输入，可以配置中断信号的电平极性，也不影响 Port A 的其余 4 根线仍作为正常的 GPIO 使用。IP 生成的时候可以选择 4 个中断信号分别拉到 GPIO 顶层，也可以选择使用中断信号组合的一根信号拉到 GPIO 顶层。

如果选择组合输出的方式，当有中断产生的时候，GPIO 不负责进行仲裁，直接输出一根中断有效（高有效或低有效，在 IP 生成时可以配置极性），经过 PAB 桥的中断模块提交到系统，驱动会最终查询到 GPIO 的中断状态寄存器，根据中断状态寄存器的登记进行处理，4 个中断源的处理优先级可能会有问题。GPIO 提交的中断可能是 int0 的请求源产生的，但是进行中断状态查询的时候 int1 或 int2 的中断登记位置起，无法保证公平。驱动在清中断源的时候应该不知道先清哪个。当 int0~int3 的中断都被清除的时候，顶层的中断信号会变为无效。驱动在清掉 1 个中断源的时候就可以把 PAB 桥的 MSI 提交开关清掉。

如果选择非组合输出的方式，GPIO 的顶层会产生 8 根中断线输出信号，当然在连接的时候根据中断的配置，只接 4 根中断线，另外 4 根中断线输出悬空不接即可。并且 int0~int3 的输出中断信号极性同样是根据 IP 生成时的配置参数确定，同时为高有效或同时为低有效，也就是说 GPIO 控制器会对输入的中断信号进行极性处理。即使是不同电平的中断输入，也是相同电平的中断输出。但是如果选择非组合输出的方式，意义就不大了，同样 4 根中断信号拉到 PAB 模块，还是需要根据中断优先级进行仲裁然后登记仲裁成功的中断源，没有节省设计工作量。

5.4 软件控制实现

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

通过写寄存器 gpio_swportx_ddr 控制外部 I/O pad 的方向，对该寄存器的写操作会映射到 GPIO 的输出信号 gpio_portx_ddr，该信号控制了外部 I/O pad 的方向。通过写寄存器 gpio_swportx_dr 驱动 I/O pad 的输出缓冲。从输入信号 gpio_ext_portx 可以获得外部的输入数据，但是前提是取决于 gpio_ext_portx 是被配置成输入还是输出：

- 配置成输入——读取输入信号 gpio_ext_portx 的数据；
 - 配置成输出——读取 Portx 的数据寄存器。
- gpio_ext_portx 寄存器是只读的，不能通过 APB 总线进行写操作。

6 UART 接口

实现标准的串口，内部寄存器定义如下：

保留原来 BMC 通过串口对 SPI Flash 进行编程的功能。

与其
的接
如

Mem IO 地址	wishbone 内部地址	位宽	属性	功能说明
0x3F8	0x1000-0000	8	R	Receiver Buffer
0x3F8	0x1000-0000	8	W	Transmitter Holding Register (THR)
0x3F9	0x1000-0001	8	RW	Interrupt Enable
0x3FA	0x1000-0002	8	R	Interrupt Identification
0x3FA	0x1000-0002	8	W	FIFO Control
0x3FB	0x1000-0003	8	RW	Line Control Register
0x3FC	0x1000-0004	8	W	Modem Control
0x3FD	0x1000-0005	8	R	Line Status
0x3FE	0x1000-0006	8	R	Modem Status

UART
它模块
口信号
下：

信号	属性	含义
wb_clk_i	输入	时钟
wb_rst_i	输入	复位
baud_tick	输入	波特率
与 wishbone 交叉开关的接口信号		
wb_dat_i[31:0]	输入	读数据
wb_dat_o[31:0]	输出	写数据
wb_ack_o	输出	周期结束响应
wb_adr_i[4:0]	输入	总线地址
wb_cyc_i	输入	总线请求/继续标志

wb_sel_i[3:0]	输入	字节使能
wb_stb_i	输入	请求开始总线周期
wb_we_i	输入	写使能
wb_err_o	输出	
与中断模块的接口信号		
int_o	输出	中断请求
与 IO 的接口信号		
srx_pad_i	输入	串行输入数据
stx_pad_o	输出	串行输出数据
rts_pad_o	输出	发送请求
cts_pad_i	输入	允许发送
dtr_pad_o	输出	数据终端准备好
dsr_pad_i	输入	数据通信设备准备好
ri_pad_i	输入	振铃指示
dcd_pad_i	输入	载波检测

7 PS/2 接口

键盘和鼠标如果通过 MSI 中断方式发给 CPU 时，MSI 中断信息中包含了区分键盘还是鼠标的信息，软件只需要读取 60h 寄存器中的数据即可，完成 60h 寄存器的读操作后，PS2 接口自动清除对应的中断。

如果采用 INTx 中断方式时，软件首先需要读取 64h 寄存器中的值来区分是键盘还是鼠标发出了中断，然后再读取 60h 寄存器中的值，之后 PS2 接口自动清除对应的中断。

PS2 接口设备时钟的最高频率是 33K，一般工作在 10KHz~20KHz。如果核心工作时钟 100MHz，分频系数为 500（devide_reg），能够得到 5 微秒的计时值（200KHz）。

PS2 的内部寄存器定义如下：

LegacyIO 地址	Wishbone 内部地址	位宽	属性	功能说明
60H	0x0000_0060	8	RW	数据缓冲寄存器
64H	0x0000_0064	8	RW	状态和控制寄存器
68H	0x0000_0068	12	RW,0x1f4	配置 5 微秒的分频系数

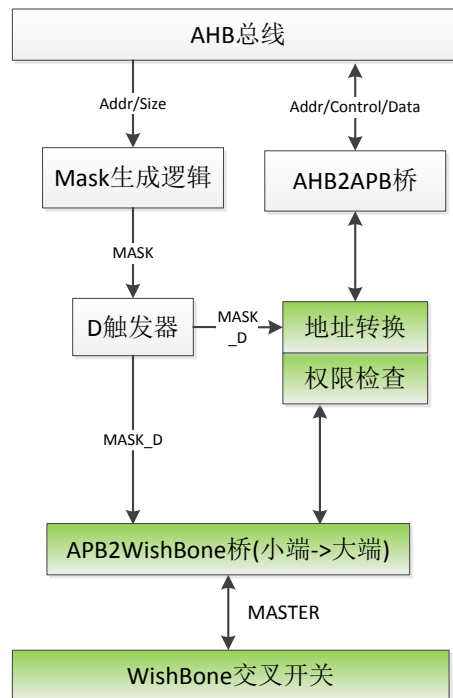
PS2 与其它模块的接口信号如下：

信号	属性	含义
与 wishbone 交叉开关的接口信号		
wb_dat_i[31:0]	输入	读数据
wb_dat_o[31:0]	输出	写数据
wb_ack_o	输出	周期结束响应
wb_adr_i[31:0]	输入	总线地址
wb_cyc_i	输入	总线请求/继续标志
wb_sel_i[3:0]	输入	字节使能
wb_stb_i	输入	请求开始总线周期

wb_we_i	输入	写使能
wb_clk_i	输入	
wb_rst_i	输入	
wb_err_o	输出	
与键盘的接口信号		
ps2_kbd_clk_pad_i	输出	
ps2_kbd_data_pad_i	输入	
ps2_kbd_clk_pad_o	输出	
ps2_kbd_data_pad_o	输出	
ps2_kbd_clk_pad_oe_o	输出	
ps2_kbd_data_pad_oe_o	输出	
与 AUX 的接口信号		
ps2_aux_clk_pad_i	输入	
ps2_aux_data_pad_i	输入	
ps2_aux_clk_pad_o	输出	
ps2_aux_data_pad_o	输出	
ps2_aux_clk_pad_oe_o	输出	
ps2_aux_data_pad_oe_o	输出	
其它		
wb_int_o	输出	

8 APB2WB 桥

APB2WB 桥将系统的 APB 请求转换为 wishbone 请求，并将 wishbone 响应转换成 APB 的响应，同时需要进行 APB 小端数据格式和 wishbone 大端数据格式的转换。对于 APB 访问的地址需要按照 wishbone 交叉开关各个端口定义的地址范围进行映射转换。另外，还需要对 APB 访问地址进行权限检查，只有落在 PS2、UART 和 SPI 的有效寄存器编制内的访问且 mask 符合要求才是有效的访问，可以继续转发；否则是非法的访问，不继续转发，直接返回错误响应，如果是读请求返回 0xffffffff 的数据。



合法的访问地址和粒度，以及地址空间转换如下表所示：

设备	寄存器	mask[3:0]	AMBA 空间地址	WISHBONE 空间地址
KEYBOARD	数据缓冲寄存器	‘b1, 只支持字节粒度访问	0x10_0060	0x60
	状态和控制寄存器	‘b1, 只支持字节粒度访问	0x10_0064	0x64
MOUSE	数据缓冲寄存器	‘b1, 只支持字节粒度访问	0x10_2060	0x60
	状态和控制寄存器	‘b1, 只支持字节粒度访问	0x10_2064	0x64
UART	RBR	‘b1, 只支持字节粒度访问	0x10_43F8	0x1000_03F8
	THR	‘b1, 只支持字节粒度访问	0x10_43F8	0x1000_03F8
	IER	‘b1, 只支持字节粒度访问	0x10_43F9	0x1000_03F9

	IIR	‘b1，只支持字节粒度访问	0x10_43FA	0x1000_03FA
	FCR	‘b1，只支持字节粒度访问	0x10_43FA	0x1000_03FA
	LCR	‘b1，只支持字节粒度访问	0x10_43FB	0x1000_03FB
	MCR	‘b1，只支持字节粒度访问	0x10_43FC	0x1000_03FC
	LSR	‘b1，只支持字节粒度访问	0x10_43FD	0x1000_03FD
	MSR	‘b1，只支持字节粒度访问	0x10_43FE	0x1000_03FE
SPI	具体寄存器参照 4.4 节	’b1111	0x10_E000	0x2000_0000

9 WB2AHB 桥

因为 32 位地址的高 4 位用于 wishbone 交叉开关路由，只有低 28 位可以使用，所以对显存和 LPC 控制器的维护访问需要进行地址高位的转换，显存和 LPC 的维护地址和 AMBA 总线地址映射关系如下：

维护地址	AMBA 总线地址	说明
0x0800_0000~0x0fff_ffff	{1'b1, 4'b 显存偏移配置值, addr[26:0]}	128MB
0x0100_0000~0x0100_ffff	0x1000_0000~0x1000_ffff	64KB
0x0200_0000~0x03ff_ffff	0x2000_0000~0x21ff_ffff	32MB 偏移值 LPC 地址偏移值={1'bx, 3'b000} [3]: 0-FWIO 低 32MB 1-FWIO 高 32MB [2:0]: 000~111 对应 MemIO 的 32MB 在 256MB 空间的偏移
	0x2200_0000~0x23ff_ffff	32MB, 偏移值={1'bx, 3'b001}
	0x2400_0000~0x25ff_ffff	32MB, 偏移值={1'bx, 3'b010}
	0x2600_0000~0x27ff_ffff	32MB, 偏移值={1'bx, 3'b011}
	0x2800_0000~0x29ff_ffff	32MB, 偏移值={1'bx, 3'b100}
	0x2a00_0000~0x2bff_ffff	32MB, 偏移值={1'bx, 3'b101}
	0x2c00_0000~0x2dff_ffff	32MB, 偏移值={1'bx, 3'b110}
	0x2e00_0000~0x2fff_ffff	32MB, 偏移值={1'bx, 3'b111}
0x0400_0000~0x05ff_ffff	0x3000_0000~0x31ff_ffff	32MB, 偏移值={1'b0, 3'bxxx}
	0x3200_0000~0x33ff_ffff	32MB, 偏移值={1'b1, 3'bxxx}
0x0600_0000~0x0600_ffff	0x4000_0000~0x4000_ffff	64KB

10 WB2APB 桥

需要考虑在不同的应用模式下，如 PSW 旁路，显示相关旁路，可管控设备不使能等情况下，确保对这些地址的访问不进行继续转发，并直接返回错误或正常响应。

wishbone 转 APB 信号关系如下：

Wishbone 信号	APB 信号	连接关系
wb_clk_i	PCLK	直连
wb_rst_i	PRESETn	反相连

wb_adr_o[31:0]	PADDR [31:0]	直连
wb_cyc_o & wb_stb_o & wb_adr_o[21:19]	PSELx	组合逻辑生成一个有效信号。当 PREADY 有效时，下一拍 PSELx 无效。
wb_cyc_o & wb_stb_o	PENABLE	推 一 拍 有 效 ， 当 PREADY 有效时，下一拍 PENABLE 无效。
wb_we_o	PWRITE	直连
wb_dat_o[31:0]	PWDATA[31:0]	big-endian (WB) little-edian (APB) 转换，支持 8、16、32 位读写
wb_ack_i	PREADY	直连后打一拍输出
wb_err_i	PSLVERR	直连后打一拍输出
wb_dat_i[31:0]	PRDATA[31:0]	big-endian (WB) little-edian (APB) 转换，之后打一拍输出
wb_sel_o [3:0]	——	支持 8、16、32 位粒度的读写
wb_tga_o[1:0]	——	不连

11 WISHBONE 交叉开关

wishbone 交叉开关是一个包含最多 8 个 master，16 个 slave 的全交叉开关，用地址的最高 4 位 [31:28] 作为 16 个 slave 的选择信号。交叉开关支持每个 slave 按 4 级优先权选择各个 master，相同优先级时采用轮转仲裁。

优先权寄存器堆放在 slave 15 的编址空间内。通过对地址 [31:28] 的判断为 0xf，就是对 slave 15 的访问，如果地址 [27:24] 的值为 0xf，则是对优先权寄存器堆的访问，而非真正的 slave 15。

wishbone 交叉开关端口分配如下：

Master Port	模块名	Slave Port	模块名	地址空间划分
M0	APB2WB 桥	S0	PS/2	0x0000_0000~0x0000_1FFF
M 1	JTAG 维护/调试/编程	S1	UART	0x1000_0000~0x1000_1FFF
M 2	复位自加载启动	S2	SPI	0x2000_0000~0x2000_1FFF
M 3		S3	WB2AHB	0x3000_0000~0x3FFF_FFFF
M 4		S4	WB2APB	0x4000_0000~0x4FFF_FFFF
M 5		S5		
M 6		S6		
M 7		S7		
		S8		
		S9		
		S10		
		S11		
		S12		
		S13		
		S14		
		S15	(交叉开关优先权配置)	0xFF00_0000~0xFFFF_FFFF

12 复位

下列信号是复位信号

复位信号名	方向	含义	有效电平
i_DCOK_H	输入	上电复位 OK	高
i_MTRST_N	输入	板级维护复位	低
i_APB_PRESET_N	输入	APB 总线复位	低
TRSTB	输入	JTAG 复位	低
o_apb_resetrn	输出	APB 专用维护通路复位	低
o_io_rst_n	输出	核心逻辑复位	低
o_pab_csr_rst_n	输出	PSW 和 PAB 的 CSR 寄存器复位	低
o_SICH_PWRGD	输出	ICH2 芯片上电 OK	高
dbg_rst	内部信号	TAP 控制器产生的复位	高
wb_subsystem_rst_n	内部信号	wishbone 总线复位	低
apb_mtrst_n	内部信号	自引导产生的维护复位	低
USBPHY_RESET[5:0]	输出	USB PHY 复位	高
SATAPHY_RESET[2:0]	输出	SATA PHY 复位	高
PCIEPHY_RESET[5:0]	输出	PCIE PHY 复位	高
GCLKPLL_RESET	输出	GCLK PLL 复位	高
PSWPLL_RESET	输出	PSW PLL 复位	高
GPUPLL_RESET	输出	GPU PLL 复位	高
MCPLL_RESET	输出	MC PLL 复位	高
DIVCLK_rst_n	输出	内部分频器复位	低
RefclkDiv_RstN	输出	参考时钟分频器复位	低

注：

- TAP 控制器根据 TRSTB 产生 dbg_rst 复位信号，用于复位 ADBG 模块，ADBG 模块同时还受 wishbone 总线复位影响。
- BOOTER 根据板级维护复位 i_MTRST_N 产生 apb_mtrst_n 复位信号复位 WB2APB 模块，WB2APB 模块产生 o_apb_resetrn 复位，来复位挂载在 WB2APB 的 APB 总线上的设备。

c) Wishbone 总线复位是 BOOTER 把上电复位同步后产生的。

各模块的时钟和复位需求如下：

模块名	ASIC 时钟频率	复位信号	是否有 异步交接
BAUD_PRESCALE	25M	wishbone 总线复位(上电复位)	
CLOCK_DIVIDE4	100M、25M		
ADBG	100M、10M	wishbone 总线复位、TAP 产生的 dbg_rst	有
TAP	10M	TRSTB	
PRG_SPI 编程	100M	wishbone 总线复位	
UART	100M	wishbone 总线复位	
WB 交叉开关	100M	wishbone 总线复位	
DW_wb_ssi	100M	wishbone 总线复位	有
PS2	100M	APB 总线复位	
AHB_MT	100M	APB 总线复位	
WB2AHB	100M	APB 总线复位	
WB2APB	100M	自引导根据板级维护复位产生的维护复位	
SICH_BOOTER	100M	板级维护复位、上电复位	

13 串口编程 Flash 芯片（prg_spi）

13.1 概述

串口编程逻辑是通过 uart 串口线对 spi flash 芯片进行编程的逻辑，通过板级跳线信号 UART_HDPLX 进行选择，如果 UART_HDPLX=0，则代表进行串口编程，uart 接口信号和 uart 控制器无法通信，spi flash 芯片和串口编程逻辑之间通信，flash 芯片和 spi 控制器也无法通信；如果 UART_HDPLX=1，则禁止串口编程，uart 控制器和 spi 控制器都可以正常工作。

13.2 编程命令包格式

串口编程命令包格式如下：

字节序号	字节内容
1	写长度（WLEN）
2	读长度（RLEN）
3	写数据 1
...	
W	写数据 W（最后一个）
W+1	
...	
W+R	

根据命令包格式的内容，每一个字节序号对应的内容都是 9 位的数据，第[7:0]是数据，第[8]位是低 8 位数据的奇偶校验值，采用的是偶校验（Even，有奇数个“1”，则 parity=1）。字节序号 1 对应写长度，即从字节序号 3 开始数起的写长度（写数据 1），字节序号 2 对应读长度，后面依次是写 Flash 命令+地址+数据。

对于读写长度如有下规定：

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

0~3 —— 保留，视为 0；

4 —— 2^4 ，长度为 16；

5 —— 2^5 ，长度为 32；

6 —— 2^6 ，长度为 64；

7 —— 2^7 ，长度为 128；

8~14 —— 2^8 ，长度为 256；

15 —— 长度为 0；

读写长度的低 4 位根据 RLEN[3:0]或 WLEN[3:0]的真实值确定，这样可以接受的读写长度为最大为 256+15 个字节。

13.3 编程工作流程

正常的工作流程是，首先可能需要对 flash 芯片进行擦除操作-sector erase，因为如果 flash 芯片已经有内容了，即不是初始的全“1”数据，那么新编程进去的数据就会和原来的数据与操作后存进 flash，导致并不是期望的数据存入。所以要先进行 write enable 操作，然后执行 sector erase，然后再执行 write enable，最后进行 page program 操作。

1) Write enable: 对应的命令包内容如下

字节序号	字节内容
1	'h101
2	'h000
3	'h006-write enable cmd

2) Sector erase: 对应的命令包内容如下

字节序号	字节内容
1	'h003 或'h104(取决于发出的地址位数，分别对应 16 位地址和 24 位地址，以 24 位地址为例)
2	000
3	'h120-sector erase cmd
4	地址[23:16]
5	地址[15:8]
6	地址[7:0]

3) Page program: 对应的命令包内容如下

字节序号	字节内容
1	'h10x(x=编程数据字节数+1 字节操作码+3 字节地址)
2	000
3	'h102-page program cmd
4	地址[23:16]
5	地址[15:8]
6	地址[7:0]
7	数据字节 0
8	数据字节 1
9	数据字节 2
10	数据字节 3
.....	

13.4 接口信号

串口编程逻辑的接口信号如下：

信号	属性	含义
时钟复位信号		
i_CORE_CLK	输入	时钟
i_rst_n	输入	复位
与串口接口信号		
txd	输出	串口编程输出
rx_d	输入	串口编程输入
rx_error	输出	串口接收错误指示
与 SPI 接口信号		
o_cs_n	输出	片选信号
o_clk	输入	spi 时钟
o_mosi	输出	spi 数据输出
i_miso	输入	spi 数据输入
其他信号		
tick	输入	波特率计数
UART_HDPLX	输入	编程选择, 0-串口编程模式, 1-正常串口工作