ARM 与 FPGA 经 SPI 通信

调试报告

目录

1 SPI 背景知识

SPI（串行外围总线，Serial Peripheral Interface）电路是与其他外部设备以主机/从机模式进行同步串行数据链路。它还支持处理器与外部处理器之间连接和通信。

从本质上来说，SPI 使用一个移位寄存器串行传送若干数据位到其他 SPI 设备。在数据传输期间，有一个设备称为“主机”，用来控制传输数据流，包括时钟和片选信号；还有其他设备作为“从机”，用来移位和传输数据。主机使用 NSS 信号对从机进行片选，如果有多个从机存在，那么由主机分别单独

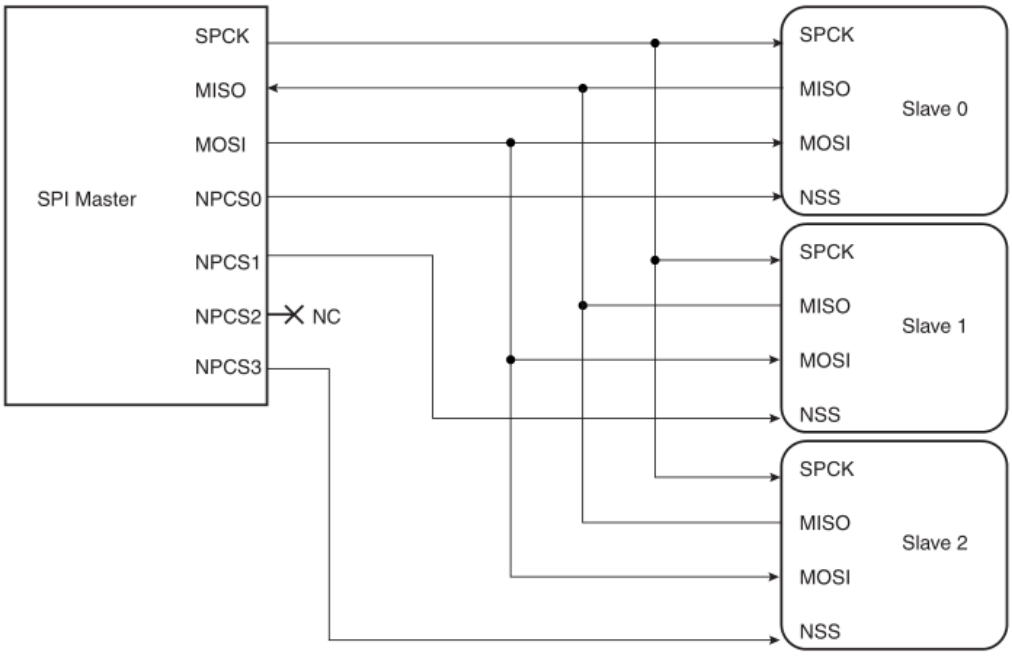


图 1 SPI 通信信号和接线图

对从机产生片选信号 NPCS。

SPI 系统包括两根数据线和两根控制线组成：

•主发从收（MOSI）：这条数据线提供从主机的移位寄存器输出到从机的输入之间的数据流。

•主收从发（MISO）：这条数据线提供从从机输出到主机输入的数据流，这里可能有不止一个从机发送数据。

•串行时钟（SPCK）：这条控制线是由主机驱动的，并且控制数据位传输流。主机

可以设置为不同波特率的数据传输速度。这条 SPCK 线一个周期传输一位数据。

•从机片选（NSS）：这条控制线允许对从机进行硬件上的开/关。

SPI 有四种不同的工作模式，MODE0/MODE1/MODE2/MODE3，其中使用的最为广泛的是 MODE0 和 MODE3 方式。通过对应寄存器配置来决定，如表。

表格 1 SPI **四种工作模式及其寄存器配置**

SPI 模式	CPOL	NCPHA	工作模式
MODE0	0	1	空闲时低电平，下降沿采样
MODE1	0	0	空闲时低电平，上升沿采样
MODE2	1	1	空闲时高电平，下降沿采样
MODE3	1	0	空闲时高电平，上升沿采样

图 2 说明了 SPI 的具体工作过程和时序关系。

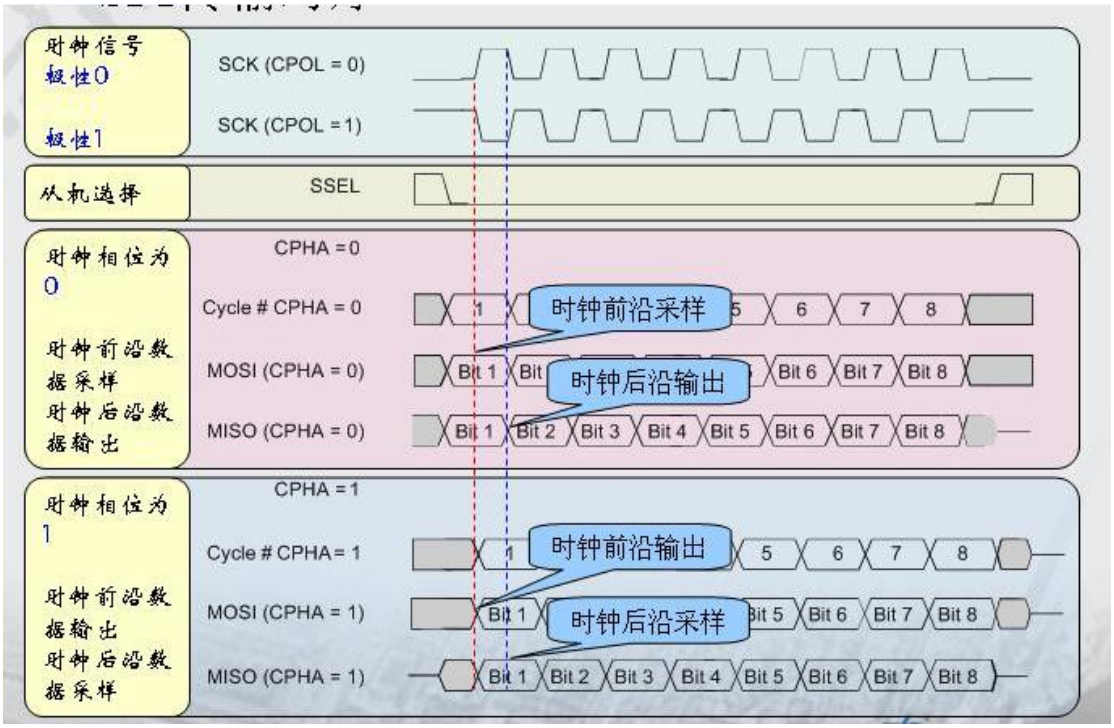


图 2 SPI 工作模式时序关系图

2 ARM 的 SPI 功能设计

2.1 管脚分配

对照图 1，通过查阅底板和核心板电路原理图可知 ARM 和 FPGA 管脚的分配情况和经过插接件的连接情况，如下表：

表格 2 SPI **功能管脚分配表**

序号	功能	ARM 管脚号/用途	FPGA 管脚号/用途	ARM 信号类型	FPGA 信号类型
1	MISO 主收从发	179/ **PA0**/SPI0_MISO/MCDB0	Y22	输入	输出
2	MOSI 主发从收	180/ **PA1**/SPI0_MOSI/MCCDB	W21	输出	输入
3	SPCK 时钟	181/ **PA2**/SPI0_SPCK	W22	输出	输入
4	CS 从机片选	182/ **PA3**/SPI0_NPCS0/MCDB3（低电平有效）	W20	输出	输入

2.2 与其他组件的依赖性

2.2.1 I/O 线

面向外部设备的管脚多路复用的 PIO 线。编程者必须首先对 PIO 控制器进行编程，为 SPI 管脚分配它们的外围功能。

2.2.2 能量管理

SPI 可能通过 PMC 来控制时钟，因此编程者应该首先配置 PMC 以使能 SPI

外设时钟。

2.2.3 中断

SPI 也可以设置不同类的中断，但在本工程中未涉及。

2.3 SPI 寄存器详解

AT91SAM9XE 中与 SPI 通信相关的寄存器有 12 个，其中能进行写操作的有 4 个，下面分别对这四个寄存器的参数做详细解释。

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	LASTXFER
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
SWRST	–	–	–	–	–	SPIDIS	SPIEN

2.3.1 SPI Control Register（SPI_CR）

SPIEN：SPI Enable

0，无效用

1，允许 SPI 发送和接收数据（SPI 使能）

SPIDS：SPI Disable

0， 无效用

1， 禁止 SPI 发送和接收数据（SPI 禁用）

SWRST：SPI Software Reset

0，无效用

2， 软件重置 SPI

软件重置之后，SPI 将处于从机模式。

LASTXFER：Last Transfer

0，无效用

1，在一次传输完成之后，当前的片选信号 NPCS 将变成无效。

31	30	29	28	27	26	25	24
DLYBCS							
23	22	21	20	19	18	17	16
–	–	–	–	PCS			
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
LLB	–	0	MODFDIS	–	PCSDEC	PS	MSTR

2.3.2 Mode Register（SPI_MR）

- MSTR: Master/Slave Mode

  0 ，SPI 处于从机模式

  1 ，SPI 处于主机模式

- PS: Peripheral Select

  0，混合片选

  1，变化片选

- PCSDEC: Chip Select Decode

  0，片选信号直接连接外部设备

  1，4 根片选线连接 4~8 位译码器

- MODFDIS: Mode Fault Detection

  0，模式故障识别使能

  1，模式故障识别禁用

- LLB: Local Loopback Enable

  0，Local loopback path 禁用

  1，Local loopback path 使能

- PCS: Peripheral Chip Select

  这个字段仅用于 PS=0 时

  如果 PCSDEC=0：

  PCS = xxx0 NPCS[3:0] = 1110 NPCS0 信号线连接从机被选中

  PCS = xx01 NPCS[3:0] = 1101 NPCS1 信号线连接从机被选中

  PCS = x011 NPCS[3:0] = 1011 NPCS2 信号线连接从机被选中

PCS = 0111　NPCS[3:0] = 0111　NPCS3 信号线连接从机被选中

PCS = 1111　　没有从机被选中

(x = don’t care)

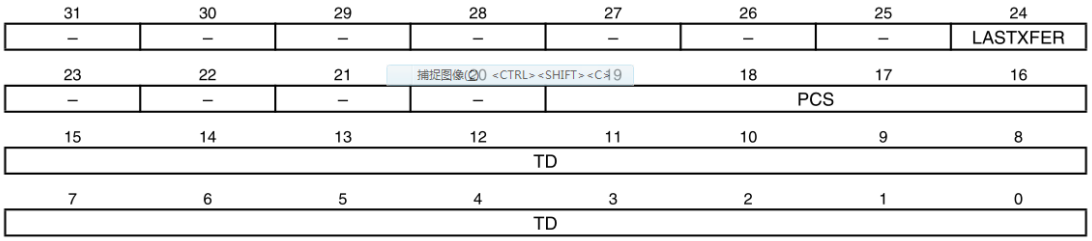
如果 PCSDE=1：

PCS[3:0] output signal = PCS

- DLYBCS:Delay Between Chip Select

这个字段定义了从一个片选信号 NPCS 闲置到另一个片选信号 NPCS 激活之间的延迟。DLYBCS 延迟时间保证了片选信号之间无重叠。在 DLYBCS 小于等于 6 时，延迟时间默认为 6 个主时钟 MCK 周期。在其余情况下，具体计算公式如下：

$$\text{Delay Between Chip Select} = \frac{DLYBCS}{MCK} \quad (1)$$



2.3.3 Transmit Data Register（SPI_TDR）

- TD:Transmit Data

将要被传输的数据被存储在这个寄存器里，将要被传输的信息必须在写入寄存器之前被编码成正确的格式。

- PCS:Perripheral Chip Select

这个字段仅用于 SPI_MR_PS=1 时

如果 PCSDEC=0：

PCS = xxx0　NPCS[3:0] = 1110　NPCS0 信号线连接从机被选中

PCS = xx01　NPCS[3:0] = 1101　NPCS1 信号线连接从机被选中

PCS = x011　NPCS[3:0] = 1011　NPCS2 信号线连接从机被选中

PCS = 0111　NPCS[3:0] = 0111　NPCS3 信号线连接从机被选中

PCS = 1111　　没有从机被选中

(x = don’t care)

如果 PCSDE=1：

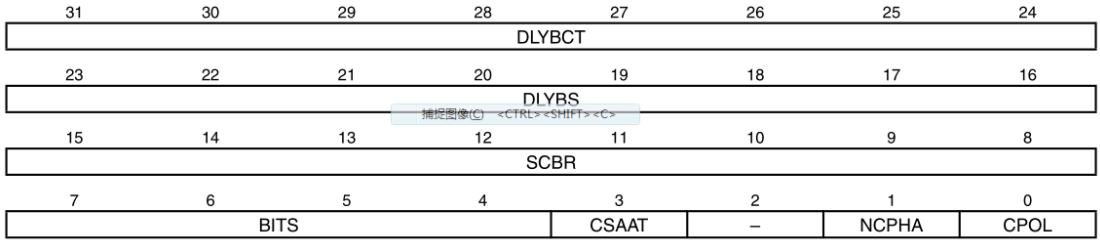
PCS[3:0] output signal = PCS

- LASTTXFER:Last Transfer

这个字段仅用于 SPI_MR_PS=1 时

0，无效用

1，当前的 NPCS 在 TD 中的数据被传输后将被解除绑定



2.3.4 Chip Select Register 0（SPI_CSR0）

- CPOL:Clock Polarity 时钟极性

0，闲置状态下时钟为逻辑低电平

1，闲置状态下时钟为逻辑高电平

CPOL 用来确定时钟信号 SPCK 的闲置状态，是低电平还是高电平。它通常和 NCPHA 一块使用，来确定主机和从机之间的时钟和数据流（工作模式）。

- NCPHA：Clock Phase 时钟相位

0，在 SPCK 时钟信号上升沿改变信号，在下降沿捕捉信号

1，在 SPCK 时钟信号上升沿捕捉信号，在下降沿改变信号

- CSAAT:Chip Select Active After Transfer

0，外部设备片选信号线在最后一次传输结束后立刻拉高电平（片选无效）

1，外部设备片选信号线在最后一次传输结束后不拉高电平，一直保留到要对新的片选开始新的传输时。

- BITS:Bits per Transfer

这个字段定义了每次传输的数据位数，未定义的保留值不使用。

表格 3 BITS 字段设置值

BITS	Bits Per Transfer
0000	8
0001	9
0010	10
0011	11
0100	12
0101	13
0110	14
0111	15
1000	16
1001/1010/1011/1100/1101/1110/1111	保留值

- SCBR:Serial Clock Baud Rate 串行时钟波特率

在主机模式下，SPI 接口使用一个分频器，从主时钟 MCK 中分出来一个 SPCK 时钟，SPI 时钟波特率是由 SCBR 决定的，可以分 1~255 档，具体计算公式如下：

$$\text{SPCK baudrate} = \frac{MCK}{SPI\_CSR.SCBR} \quad (2)$$

禁止将 SCBR 写 0。根据公式（2）表明，分母是 0 的情况下是不能计算的。在重置时，SCBR 被清零，编程者在调试前必须配置 SCBR。

- DLYBS:Delay Before SPCK

这个字段定义了从片选信号 NPCS 有效到时钟信号 SPCK 有效的延迟。

当 DLYBS 等于 0 时，以上两者之间的延迟等于时钟信号 SPCK 周期的一半。在其余情况下，具体计算公式如下：

$$\text{Delay before SPCK} = \frac{DLYBS}{MCK} \quad (3)$$

- DLYBCT:Delay Between Consecutive Transfers

这个字段定义了对同一外部设备的两次连续传输之间的延迟。这个延迟通常在每次传输后和片选信号改变前需要使用。

当 DLYBCT 等于 0 时，两次连续传输之间没有插入延迟。具体计算公式如下：

$$\text{Delay Between Consecutive Transfers} = \frac{DLYBS}{MCK} \quad (4)$$

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	-	-

2.3.5 Peripheral Clock Enable Register（PMC_PCER）

- PIDx:Peripheral Clock x Enable

  0 无效用

  1 使能对应外设时钟

2.4 SPI 寄存器配置

寄存器配置所要实现的目标是：

2.4.1 管脚复用

ARM 的 SPI 功能使用了 PA0/PA1/PA2/PA3 管脚，因此需要把这些管脚配制成相应的工作模式，即要把 PIOA 的工作模式初始化为外部函数 1。

2.4.2 SPI 使能

把 SPI 功能使能（打开总开关），**SPI_CR. SPIEN=1，SPI_CR. SPIDS=0**。并且 ARM 处于主机模式，即 **SPI_MR.MSTR=1**。

2.4.3 时钟

首先，要利用 **PMC_PCER** 将 SPI 的时钟使能，因 Peripheral ID=12，Peripheral Mnemonic=SPI0 ， Peripheral name=serial peripheral interface 0 ， 因 此 **PMC_PCER.PID12=1**。

ARM 的时钟 MCK（master clock）为 90MHz，而 FPGA 中定义的时钟频率

为 25M，要保证 SPI 的正常工作，需要将主机 SPI 的时钟频率（SPCK baudrate）定义为小于 25MHz，由公式（2）可知，$SPI_CSR.SCBR$要大于等于 4，这里定义 **SPI_CSR0.SCBR=5=0x04**。

2.4.4 片选

串行外部设备是由 NPCS0/NPCS1/NPCS2/NPCS3 信号来声明的。在没有声明的情况下，每次传输前和传输后，NPCS 信号都是高电平；只有在传输过程中，NPCS 才是低电平有效。

片选可以通过两种途径：

a.混合片选：SPI 只能与 1 个从机传输数据

b.变化片选：SPI 可以与多个从机传输数据

在本项目中，只有 FPGA 一个从机，因此这里选用混合片选的方式。对于混合片选的方式，只需要把 SPI_MR.PS 写 0。在这种情况下，当下的外部设备由 SPI_MR.PCS 字段定义，而 SPI_TDR.PCS 字段失去了效用。

结合硬件电路图可知，NPCS0 用作片选信号线。传输数据时，将 NPCS0 变成低电平 0，即代表片选选中了 FPGA 作为从机。**SPI_MR.PS=0，SPI_MR.PCSDEC=0，SPI_MR.PCS= xxx0**（这里设定为 1110）。

2.4.5 模式

本项目中使用 MODE3（空闲时时钟高电平，上升沿采样数据，下降沿改变数据），即 **SPI_CSR0.CPOL=1，SPI_CSR0.NCPHA=0**。

2.4.6 传输位数

本项目中使用一次传输 8 位，即 **SPI_CSR0.BITS=0000**。

2.5 SPI 寄存器编程

在 ARM 工程中，在 spi.c 文件中编写函数，实现对 2.3 节和 2.4 节中 SPI

相关寄存器的定义和写操作，如对寄存器的写操作函数编码如下：

```
#include "AT91SAM9XE512.h"

//write register function definetion

void SPI_WRITE_CR(unsigned int enable_1,

                  unsigned int software_reset_1,

                  unsigned int last_transfer_1)

{

   AT91C_BASE_SPI0->SPI_CR = (AT91C_SPI_SPIEN & (enable_1))

                  | (AT91C_SPI_SWRST & (software_reset_1<< 7))

                  | (AT91C_SPI_LASTXFER & (last_transfer_1<< 24));

}

void SPI_WRITE_MR(

                  unsigned int peripheral_select_1,

                  unsigned int chip_select_decode_1,

                  unsigned int mode_fault_detection_1,

                  unsigned int local_loopback_enable_1,

                  unsigned int peripheral_chip_select_4,

                  unsigned int delay_between_chip_selects_8)

{

   AT91C_BASE_SPI0->SPI_MR = (AT91C_SPI_MSTR & 1)

                  | (AT91C_SPI_PS & (peripheral_select_1 >> 1))

                  | (AT91C_SPI_PCSDEC & (chip_select_decode_1 >> 2))

                  | (AT91C_SPI_FDIV & (0x01 >> 3))

                  | (AT91C_SPI_MODFDIS & (mode_fault_detection_1 >> 4))

                  | (AT91C_SPI_LLB & (local_loopback_enable_1 >> 7))

                  |(AT91C_SPI_PCS                    &((unsigned

int)peripheral_chip_select_4 >> 16))

                  |          (AT91C_SPI_DLYBCS          &((unsigned

int)delay_between_chip_selects_8 >> 24));
```

```
}

void SPI_WRITE_TDR(unsigned int transmint_data_16)
{
   AT91C_BASE_SPI0->SPI_TDR = (AT91C_SPI_TD &(transmint_data_16 >> 0));
}

void SPI_WRITE_CSR(unsigned int delay_before_spck_8,
                                    unsigned int delay_between_consecutive_Transfers_8)
{
      AT91C_BASE_SPI0->SPI_CSR[0] =(AT91C_SPI_CPOL & (1)) //?
                                    |(AT91C_SPI_NCPHA & (0x00 << 1))   //空闲搞电平，上升沿采样
                                    | (AT91C_SPI_CSAAT & (0x00 << 3))
                                    | (AT91C_SPI_BITS & (0x00<< 4))   //一次传输几位
                                    | (AT91C_SPI_SCBR & (0x2d << 8)) //0X04=配置成低于25MHz
                                    |          (AT91C_SPI_DLYBCT          & (delay_between_consecutive_Transfers_8 << 24))
                                    | (AT91C_SPI_DLYBS & (delay_before_spck_8 << 16)) ;

}
void SPI_Write(unsigned short data)
{
      // Discard contents of RDR register
      //volatile unsigned int discard = spi->SPI_RDR;
      unsigned int discard = AT91C_BASE_SPI0->SPI_RDR;
      // Send data
      while ((AT91C_BASE_SPI0->SPI_SR & AT91C_SPI_TXEMPTY) == 0);
```

```
        AT91C_BASE_SPI0->SPI_TDR = data |((0x1) << 16);
        discard = AT91C_BASE_SPI0->SPI_RDR;
        while ((AT91C_BASE_SPI0->SPI_SR & AT91C_SPI_TDRE) == 0);
}
```

在 spi.h 中，添加 spi.c 中定义的函数，以供外部程序调用这些函数，编码如下：

```
#include "global.h"
//#include "FPGADebugConfig.h"
#include "at91sam9xe_out.h"
#ifndef _SPI_H_
#define _SPI_H_
/// Calculate the PCS field value given the chip select NPCS value
#define SPI_PCS(npcs)           ((~(1 << npcs) & 0xF) << 16)
extern void SPI_WRITE_CR(unsigned int enable_1,
                         unsigned int software_reset_1,
                         unsigned int last_transfer_1);
extern void   SPI_WRITE_MR(
                         unsigned int peripheral_select_1,
                         unsigned int chip_select_decode_1,
                         //unsigned int clock_selection_1,
                         unsigned int mode_fault_detection_1,
                         unsigned int local_loopback_enable_1,
                         unsigned int peripheral_chip_select_4,
                         unsigned int delay_between_chip_selects_8);
extern void SPI_WRITE_TDR(unsigned int transmint_data_16);
extern void SPI_WRITE_CSR(unsigned int delay_before_spck_8,
                          unsigned int delay_between_consecutive_Transfers_8);
extern void SPI_Write(unsigned short data);
```

```
extern void ISR_Spi0(void);
#endif
```

2.6 主程序功能与编程

预定的 spi 功能是，在 app.c 中编写代码，实现如下功能：

① PIO 管脚模式配置
② 确定时钟
③ 寄存器写入
④ SPI 数据传输

编程如下：

```
void    main (void)
{
        static const Pin pinsin = PINS_SPI_IN;
        static const Pin pinsout = PINS_SPI_OUT;
    DisableInt((unsigned char)AT91C_ALL_INT);
        AT91C_BASE_PMC->PMC_PCER = (1 << AT91C_ID_SYS);
/* Ensure clocks for the SMC are enabled*/
        AT91C_BASE_PMC->PMC_PCER = (1 << AT91C_ID_PIOA);
/* Enable PIOA clock*/
        PIOInit();
        PIO_Configure(&pinsin, 1);
        PIO_Configure(&pinsout, 1);
         //delay(321454);
          SPI_WRITE_CR(1,0,0);
          AT91C_BASE_PMC->PMC_PCER=(0 << 12);
          //delay(321454);
       SPI_WRITE_MR(1,1,0,0,0x0e,0x06);
        SPI_WRITE_CSR(0x00,0x01);
        //delay(321454);
```

```
            AT91C_BASE_SPI0->SPI_PTCR =
                                        AT91C_PDC_RXTDIS | AT91C_PDC_TXTDIS;
    while(1)
    {
            SPI_Write(0xaa);
            SPI_Write(0x55);
            SPI_Write(0xa4);
            SPI_Write(0x00);
            SPI_Write(0x02);
    }
```

3 FPGA 的 SPI 功能设计

3.1 概述

在 FPGA 芯片中，需要设计专用于 SPI 通信的驱动模块以及相应的功能辅助模块，这些模块需要实现的功能主要包括以下三点：

(1) 实现 ARM 与 FPGA 通过 SPI 接口进行字节流传输（ARM 作主机，FPGA 作从机）；

(2) 在(1)实现的基础上，自定义字节流的数据格式规范（用于解析字节流中的地址和数据信息）；

(3) 在(2)实现的基础上，SPI 通信驱动模块需要向 CPU_TOP 模块提供数据接口（统一参照已经开发完成的并口通信方式，以便后续的接口管理开发工作），各功能模块之间的结构关系如图 3 所示，

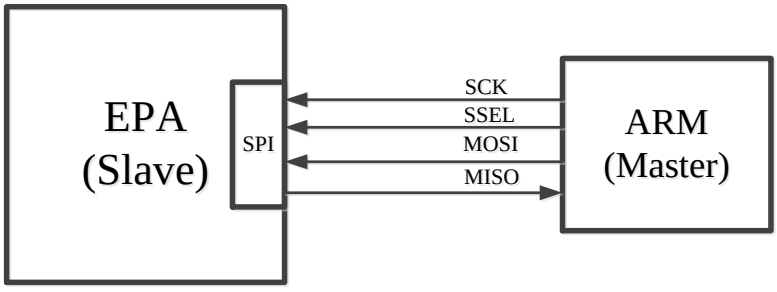


图 3　ARM 与 FPGA 之间的 SPI 通信实现方式

FPGA 侧的问题在于，SPI 模块在接收到 ARM 的配置信息后，也需要将配置信息写入 DPRAM 相应的地址区域中。为了实现写 DPRAM 的操作，需要定义 SPI 字节流的数据格式，具体来说就是如何区分 SPI 数据流中的地址信息与数据信息，如何确定 SPI 数据流的开始与终止，以及 SPI 模块如何向上层提供这些信息的接口，具体来说，就是如何将 SPI 的接口形式转化成并口的接口形式。

3.2 数据格式设计

目的是为了约束 SPI 模块按照规定的数据格式解析 SPI 字节流，方便将解析出的信息分类，提供给上层模块使用。

类似于串口通信在接收到 0xAA、0x55 之后才认为通信开始，本文档暂时设计 SPI 字节流的数据格式如图所示，

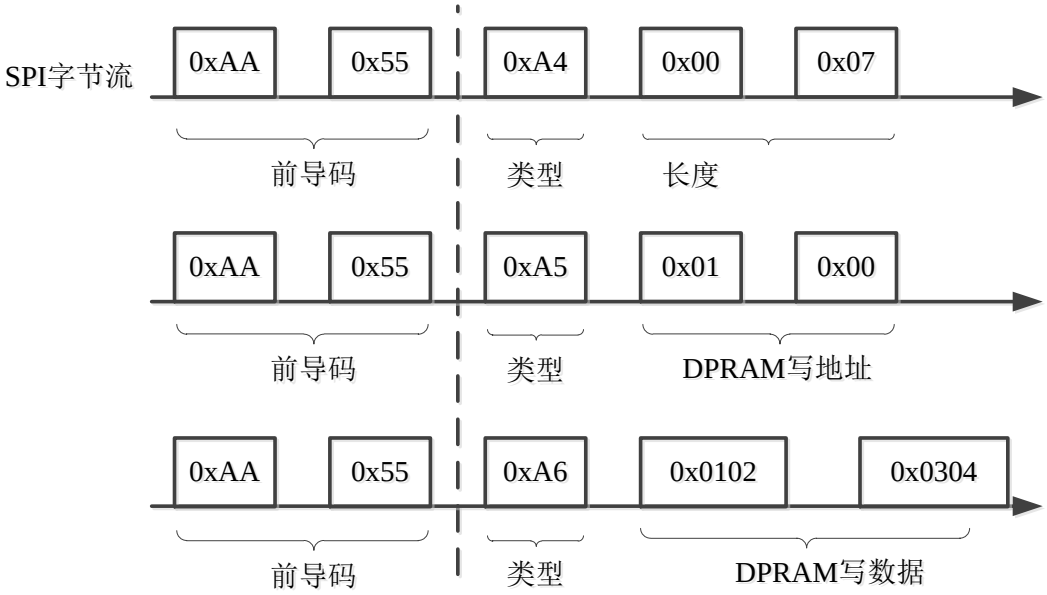


图 4　SPI 字节流数据格式定义示意图

图 4 中各个字段的定义如下：

(1) 前导码：标识 SPI 字节流的开始（如果为 0xAA、0x55，引导 SPI STATE MACHINE 进入下一状态）；

(2) 类型：标识“类型”字段后面的数据是长度信息、写地址信息还是写数据信息（0xA4 表示长度，0XA5 表示写地址，0XA6 表示写数据）；

(3) 长度：标识写入 DPRAM 数据的长度（考虑到以太网报文最大长度 1518B，

因此**长度字段使用 2B 字节**）；

(4) DPRAM 写地址：标识 DPRAM 写数据的起始位置（与并口操作 DPRAM 一致，共 10 位地址线，因此 **DPRAM 写地址使用 2B 字节**）；

(5) DPRAM 写数据：写入 DPRAM 的数据（**单位是双字 DWORD**，个数取决于(3)中的长度）；

根据上面的数据格式定义，ARM 开发板能够根据图 5 所示的流程，实现通过 SPI 接口对 FPGA 芯片的数据交互。

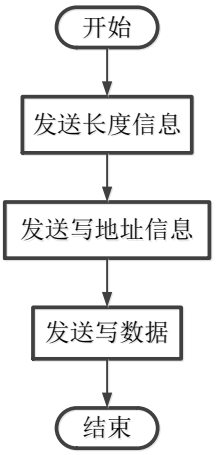


图 5　ARM 开发板与 FPGA 芯片数据交互流程图

3.3 SPI 驱动模块输入输出设计

SPI 驱动模块的输入输出接口、内部逻辑模块示意如图 6 所示，

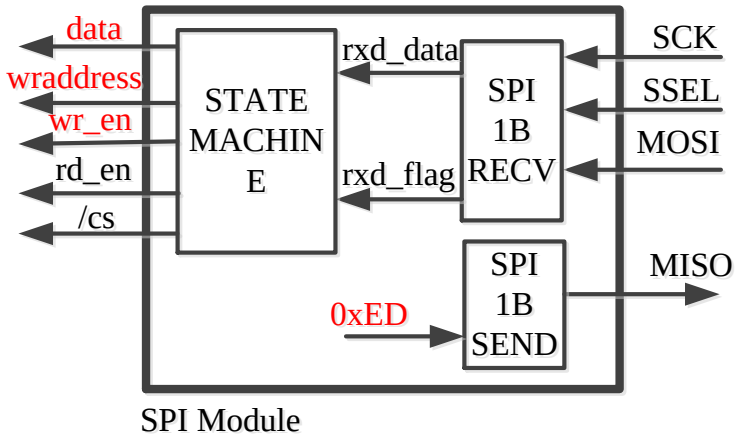


图 6　SPI 驱动模块输入输出接口及内部原理示意图

3.4 STATE MACHINE 模块设计说明

STATE MACHINE 模块的**功能**是按照图 5 和图 6 设计的数据交互格式与流程，

解析 SPI 字节流中的信息，分类输出。

(1) 设计 STATE MACHINE 有 10 个状态：IDLE，RECV_PRENUM1，RECV_PRENUM2，RECV_TYPE，RECV_LENGTH1，RECV_LENGTH2，RECV_ADDRESS1，RECV_ADDRESS2，RECV_DATA1，RECV_DATA2，状态跳转时刻在 rxd_flag 高电平（rxd_flag_h 上升沿时刻）；

(2) 设计 4 个标志位：len_Type_recved，addr_Type_recved，data_Type_recved，data_start，作为状态跳转条件；data_len 作为记录长度类型值；

4 联合调试

4.1 联合调试过程

在 ARM 的集成开发环境 IAR 中，编译、下载并运行 uCOS 工程，通过设置断点的方式，观察 ARM 寄存器相关的实时数值是否与期望相符。在 FPGA 的集成开发环境 QUARTUS 中，变异、下载并运行 FPGA_SPI_SLAVE 工程，通过 SignalTap 工具对 FPGA 引脚和内部程序中的主要信号流进行监视，观察时序波形是否与期望相符。

4.2 联合调试结果

在 app.c 中编写 SPI 数据发送语句，使 ARM 通过 SPI 接口向 FPGA 发送 5B 数据（aa,55,a4,00,02）。

```
SPI_Write(0xaa);
SPI_Write(0x55);
SPI_Write(0xa4);
SPI_Write(0x00);
SPI_Write(0x02);
```

在 SignalTap 中抓的波形如图 7 所示，图中黑色框标注的就是 FPGA 采样得到的数据，显示收发一致，通过测试。

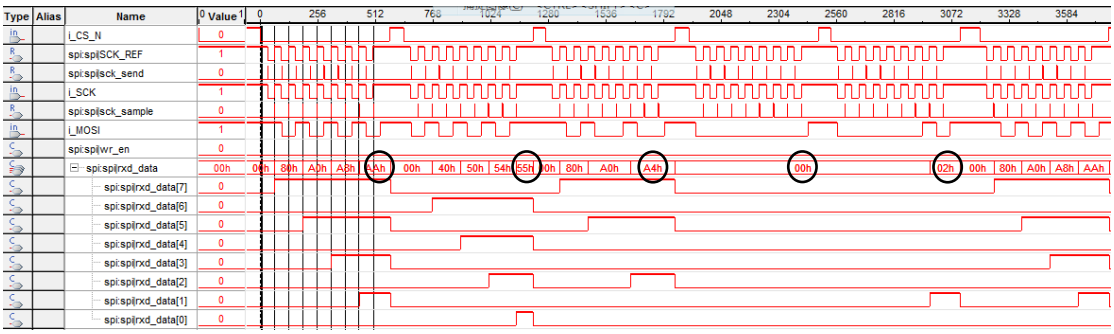


图 7 联合调试时序图

图 8 中显示的是，发送一字节数据 0xaa 时的时序图的放大细节。

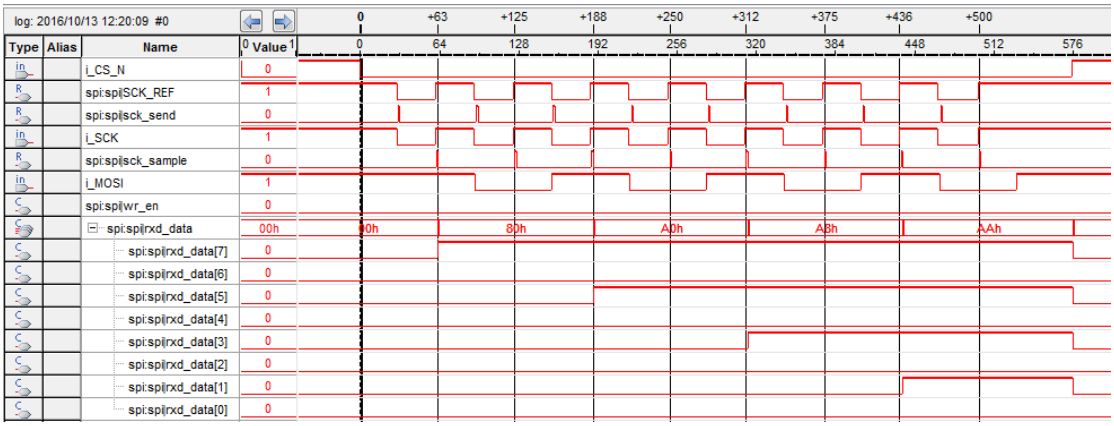


图 8 联合调试时序细节图

5 问题

5.1 CS 引脚上拉电阻问题

经过广泛查阅资料，使用 ARM9 作为主机进行 SPI 通信时，通常会在 CS 线上加一个 470k 的上拉电阻接到 3.3v 上，这样是为了两个处理器之间的电平转换。本项目中遇到的最大的问题在于，电路板中并没有设计这个上拉电阻，只能想办法以 FPGA 输入输出管脚内部的 weak 上拉电阻替代。

具体过程如下：

1. 在菜单 Assignments 中选择 Pin Planner；

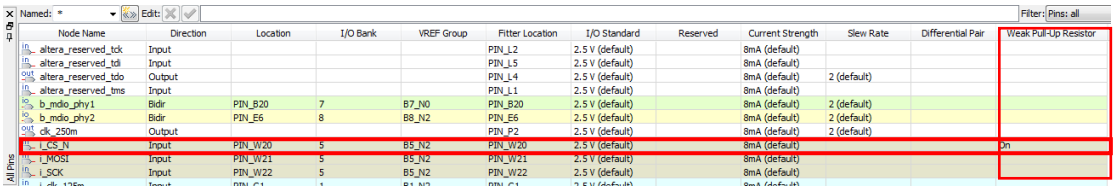
2. 在弹出的 Pin Planner 界面的 All Pins 区域里点击鼠标右键，找到 Customize Columns；

3. 在弹出的 Customize Columns 对话框的左列表框选择 Weak Pull-Up

Resistor，再点击，把 Weak Pull-Up Resistor 添加到右列表框，这样在 Pin Planner 的 All Pins 区域里就有一列 Weak Pull-Up Resistor 的设置项；

4. 再把需要上拉电阻的 Pin 在其对应的 Weak Pull-Up Resistor 列的位置双极鼠标左键，就会弹出一个 Off/On 的选项，选上 On 就可以了；

5. 重新综合布局布线生成新的下载文件即可生效。此上拉电阻式弱上拉，Altera 没有下拉电阻选项。



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Weak Pull-Up Resistor
altera_reserved_tck	Input				PIN_L2	2.5 V (default)		8mA (default)			
altera_reserved_tdi	Input				PIN_L5	2.5 V (default)		8mA (default)			
altera_reserved_tdo	Output				PIN_L4	2.5 V (default)		8mA (default)	2 (default)		
altera_reserved_tms	Input				PIN_L1	2.5 V (default)		8mA (default)			
b_mdio_phy1	Bidir	PIN_B20	7	B7_N0	PIN_B20	2.5 V (default)		8mA (default)	2 (default)		
b_mdio_phy2	Bidir	PIN_E6	8	B8_N2	PIN_E6	2.5 V (default)		8mA (default)	2 (default)		
clk_250m	Output				PIN_P2	2.5 V (default)		8mA (default)	2 (default)		
i_CS_N	Input	PIN_W20	5	B5_N2	PIN_W20	2.5 V (default)		8mA (default)			On
i_MOSI	Input	PIN_W21	5	B5_N2	PIN_W21	2.5 V (default)		8mA (default)			
i_SCK	Input	PIN_W22	5	B5_N2	PIN_W22	2.5 V (default)		8mA (default)			