SpartanMC

Serial Peripheral Interface Bus (SPI)

Table of Contents

List of Figures

List of Tables

Serial Peripheral Interface Bus (SPI)

The SPI is a SpartanMC peripheral device for serial communication using the SPI-bus. The SPI enables a bit stream of 8 bits to be shifted in and out of the component at programmable speed. The peripheral can be connected with up to 15 SPI-slaves which shares 3 Wires: SCK (serial clock), MOSI (master out slave in) and MISO (master in slave out). Each slave requires one slave-select-signal to activate the slave and connect it to the other wires.

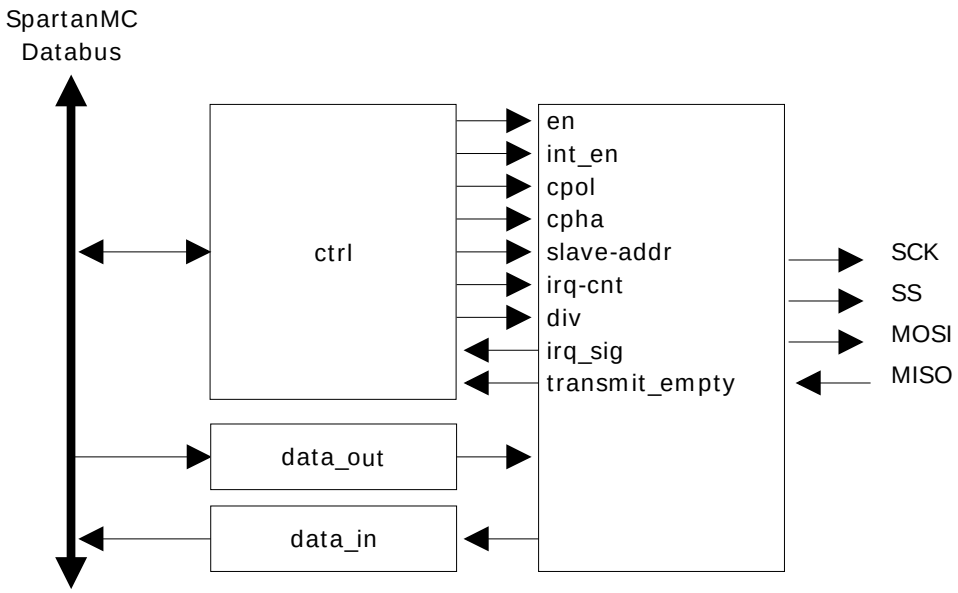

Figure 1: SPI block diagram

1. Communication

To start a transfer to a slave, the master has to clear the select signal for this slave. After this the SPI-Master can generate the clock signal and shift data to the slave. During each SPI clock cycle, one bit is sent to the slave and one bit is received from the slave. The polarity of the clock is controlled by the CPOL bit in the control register. The phase of the data is controlled by a bit called CPHA.

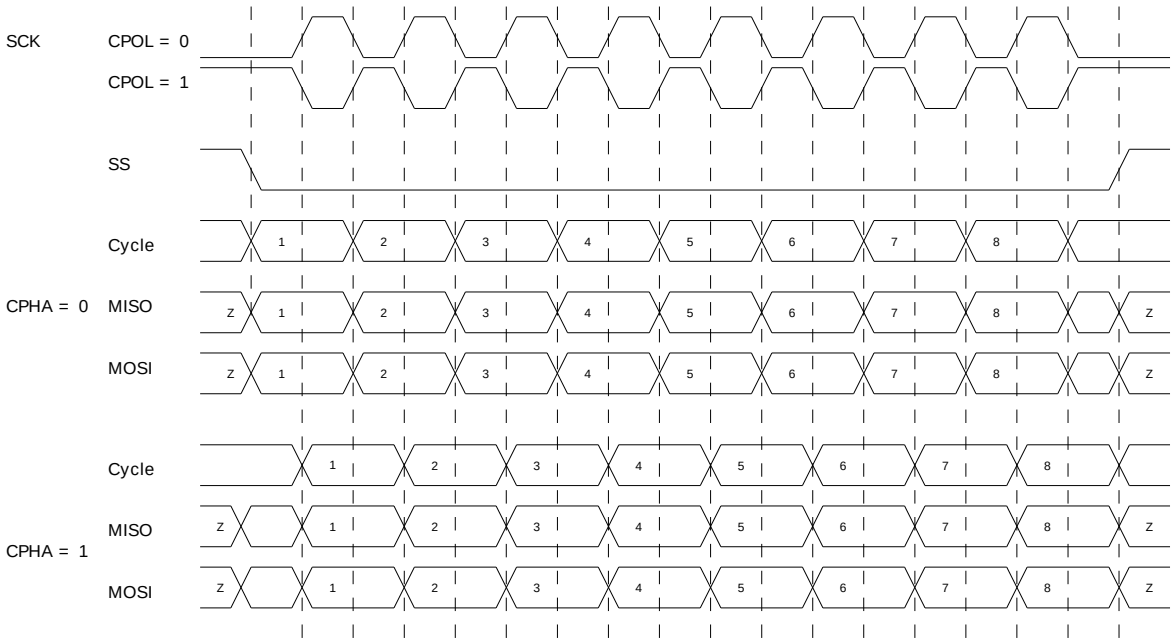

Figure 2: SPI frame

2. Module parameters

Table 1: SPI module parameters

Parameter	Default Value	Descripton
SPI_SS	0x1	Number of generated slave-select-signals

3. Peripheral Registers

3.1. SPI Register Description

The SPI peripheral provides three 18 bit registers which are mapped to the SpartanMC address space.

Table 2: SPI registers

Offset	Name	Access	Description
0	spi_control	read/ write	Contains the current SPI setting e.g. clock divider, CPOL, CPHA interrupt settings and status.
1	spi_data_out	read/ write	Register for outgoing data.
2	spi_data_in	read	Register for incoming data.

3.2. SPI Control Register

Table 3: SPI control register layout

Bit	Name	Access	Def	Description
0	EN	r/w	0	Enable the Controller.
1	IRQ_EN	r/w	0	Enable sending IRQs.
2	CPOL	r/w	0	The base value of the clock (at CPOL=0 the base value is zero).
3	CPHA	r/w	0	A value of CPHA=0 means sample on the leading (first) clock edge, while CPHA=1 means sample on the trailing (second) clock edge, regardless of whether that clock edge is rising or falling.
4-7	Slave	r/w	0	Decimal number of the slave to communicate with. 0 = 0000 = deactivates all slave-select-signals 1 = 0001 = activates the first slave-select-signal 2 = 0010 = activates the second slave-select-signal .... 15 = 1111 = activates the 15th slave-select-signal
8	transmit empty	r	0	If set to one the transmit register is empty.
9	IRQ_Sig	r	0	Interrupt-Signal goes to one when finishing a transmission. It will be cleared on a read access to the data_in register.
10-12	Bitcnt	r/w	000	Field named bitcount. 0 = 000 = 8 Bit Word

Bit	Name	Access	Def	Description
				1 = 001 = 7 Bit Word 2 = 010 = 6 Bit Word .... 7 = 111 = 1 Bit Word
13-15	clk_div	r/w	111	Sets the clock divider. Output clk = (spi_clk) / (4 * devider_value) 0 = 000 = 2^0 1 = 001 = 2^1 2 = 010 = 2^2 .... 7 = 111 = 2^7
16	ss_on	r	0	0 = all slave-select-signals deactivate
17	ss_set	r	1	0 = slave-select-signals not ready

Table 3: SPI control register layout

3.3. SPI C-Header spi.h for Register Description

```
#ifndef __SPI_H
#define __SPI_H

#include <peripherals/spi_master.h>
#include <peripherals/spi_slave.h>

// the following lines is kept for compatibility with older
// projects and may be removed in the future

// master and slave regs are identical, so we just pick one to
// emulate the old spi_t type
typedef spi_slave_regs_t spi_t;

// end compatibility section

#include <bitmagic.h>

unsigned char spi_readwrite(spi_t *spi, unsigned char data);
void spi_write(spi_t *spi, unsigned char data);
void spi_activate(spi_t *spi,unsigned int device);
void spi_deactivate(spi_t *spi);
void spi_enable(spi_t *spi);
void spi_disable(spi_t *spi);
void spi_enable_irq(spi_t *spi);
void spi_disable_irq(spi_t *spi);
```

```
void spi_set_spol(spi_t *spi, unsigned int cpol);
void spi_set_cpah(spi_t *spi, unsigned int cpah);
void spi_set_bitcnt(spi_t *spi, unsigned int bitcnt);
void spi_set_div(spi_t *spi, unsigned int div);

#endif
```

3.4. SPI C-Header spi_master.h for Register Description

```
#ifndef __SPI_MASTER_H
#define __SPI_MASTER_H

#define SPI_MASTER_CTRL_EN           0x00001
#define SPI_MASTER_CTRL_INT_EN       0x00002
#define SPI_MASTER_CTRL_CPOL       0x00004
#define SPI_MASTER_CTRL_CPHA       0x00008
#define SPI_MASTER_CTRL_SLAVE       0x000F0
#define SPI_MASTER_CTRL_TRANS_EMPTY 0x00100
#define SPI_MASTER_CTRL_INT           0x00200
#define SPI_MASTER_CTRL_BITCNT       0x01C00
#define SPI_MASTER_CTRL_DIV           0x0E000
#define SPI_MASTER_CTRL_SS_ON       0x10000
#define SPI_MASTER_CTRL_SS_SET       0x20000

typedef volatile struct {
   volatile unsigned int spi_control;   // (r/w)
   volatile unsigned int spi_data_out; // (r/w) (reset-int)
   volatile unsigned int spi_data_in;   // (r) (reset-int)
} spi_master_regs_t;

#endif
```

3.5. SPI C-Header spi_slave.h for Register Description

```
#ifndef __SPI_SLAVE_H
#define __SPI_SLAVE_H

#define SPI_SLAVE_CTRL_EN       0x00001
#define SPI_SLAVE_CTRL_INT_EN   0x00002
#define SPI_SLAVE_CTRL_CPOL   0x00004
#define SPI_SLAVE_CTRL_CPHA   0x00008
#define SPI_SLAVE_CTRL_DONE   0x00100
#define SPI_SLAVE_CTRL_INT       0x00200
#define SPI_SLAVE_CTRL_BITCNT   0x01C00
```

```
typedef volatile struct {
   volatile unsigned int spi_control;   // (r/w)
   volatile unsigned int spi_data_out; // (r/w) (reset-int)
   volatile unsigned int spi_data_in;   // (r) (reset-int)
} spi_slave_regs_t;

#endif
```

3.6. SPI Sample Application

This sample application reads the Circuit-ID from an M25P32 Flash EPROM via SPI. The application was implemented on an Xilinx ML507 evaluation board.

```
#include <system/peripherals.h>
#include <uart.h>
#include <stdio.h>
#include <spi.h>
#include "m25p32.h"


void main() {
   unsigned int i;
   stdio_uart_open(UART_LIGHT_0);
   printf("\r\nHello SPI_Sample:");

   printf(\r\nEnable the SPI-Core:");
   SET(SPI_MASTER_0->spi_control, SPI_MASTER_CTRL_EN);

   printf("\r\nPower-Up the connected SPI-Flash:");
   spi_activate(SPI_MASTER_0,1);
   spi_readwrite(SPI_MASTER_0,0xAB);
   spi_deactivate(SPI_MASTER_0);

   printf("\r\nRead ID of the SPI-Flash:\r\n");
   unsigned int id[4];
   m25p32_read_id(SPI_MASTER_0, &id[0]);

   for(i=0;i<3;i++) {
       printf("ID %u : %x\r\n",i,id[i]);
   }
   UNSET(SPI_MASTER_0->spi_control, SPI_MASTER_CTRL_EN);
   while(1);
}

void m25p32_read_id(spi_t* spi, unsigned int* data) {
   unsigned int i;
   spi_activate(spi,1);
```

```
   spi_readwrite(spi,M25P32_RDID);
   for (i = 0; i< 3; i++){
       data[i] = spi_readwrite(spi,0);
   }
   spi_deactivate(spi);
}
```

The output is send via serial connection to a host PC. Therefore an UART peripheral is required in the SoC. ID 0 = 020 specify the menufacturer type (ST), ID 1 = 020 specify device type and ID 2 = 016 indicates the memory capacity.

```
SpMC loader v20120927

Hello SPI_Sample:
Enable the SPI-Core:
Power-Up the connected SPI-Flash:
Read ID of the SPI-Flash:
ID 0 : 20
ID 1 : 20
ID 2 : 16
```