SPI on Raspberry Pi

ELECTGON
www.electgon.com
ma_ext@gmx.net

27.05.2018

Contents

1 Background

An SPI communication scheme is a full-duplex data link, using four wires. The master initiates the transaction by pulling the Slave Select (SS) wire low. A Serial Clock (SCLK) line, driven by the master, provides a synchronous clock source. The master transmits data via the Master Out, Slave In (MOSI) line and receives data via the Master In, Slave Out (MISO) line.

A master can communicate with multiple slaves via a variety of techniques. In the most common configuration, each slave has an independent SS line but shares the SCLK, MISO, and MOSI lines with the other slaves. Each slave ignores the shared lines when its SS line is not pulled low.

SPI has four modes of operation, based on two parameters: clock polarity (CPOL) and clock phase (CPHA). Master and slave must use the same mode to communicate accurately. If CPOL is zero, then SCLK is normally low, and the first clock edge is a rising edge. If CPOL is one, SCLK is normally high, and the first clock edge is a falling edge. CPHA defines the data alignment.

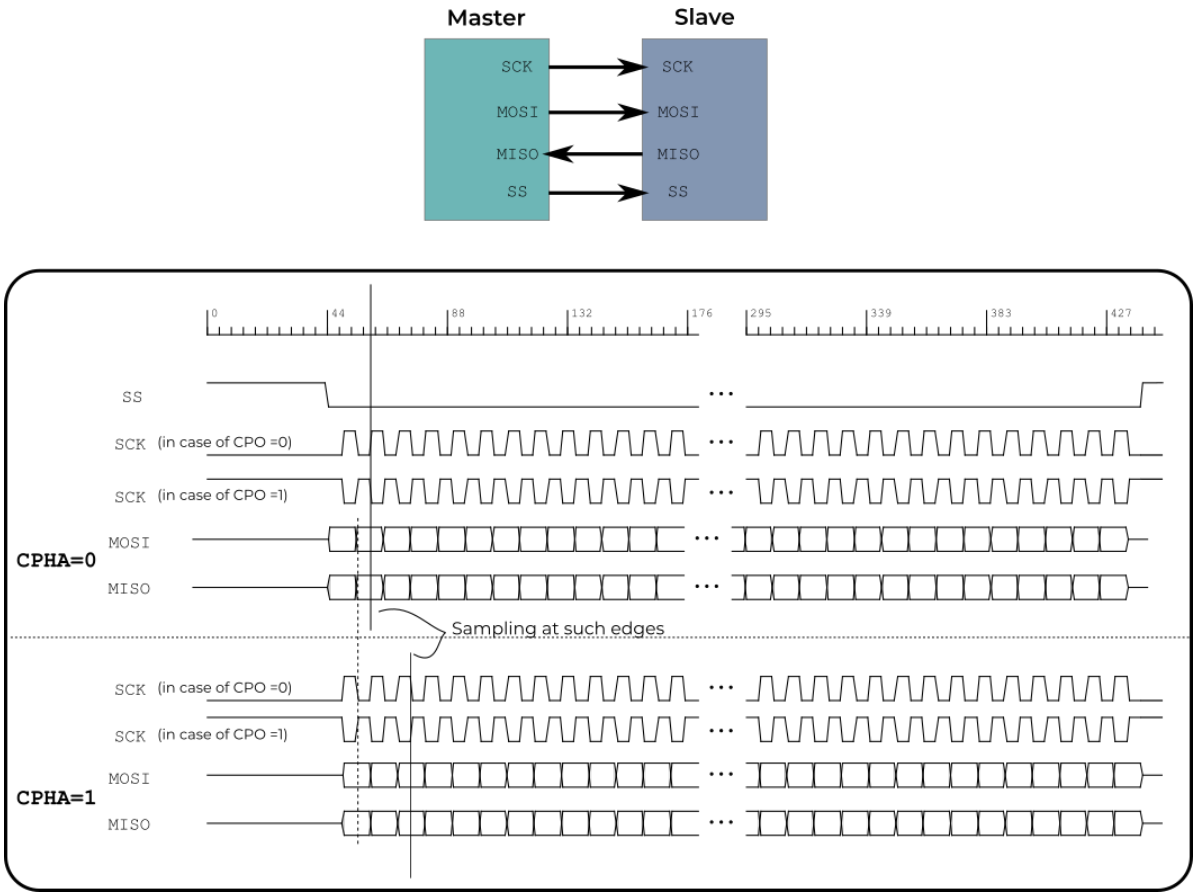

Figure 1: SPI Timing Diagram [2]

2 SPI Peripherals in Raspberry

Raspberry Pi is equipped with one SPI bus that has two chip selects. To work with SPI driver of the Raspberry Pi, it has to be enabled first as by default it is disabled. To enable it use

raspi-config.

```
$raspi-config
```

You can check on Raspberry Pi this SPI device by typing

```
$ls /dev/*spi*
```

Raspberry Pi should respond with

```
/dev/spidev0.0  /dev/spidev0.1
```

which indicates that the SPI Interface supports up to 2 spi peripherals. After that we have to choose a library to use for controlling and commanding this SPI driver.

3 SPI Driver

Raspberry Pi website lists suggested libraries written mainly in C. if working with python or Tcl or Java is needed, you can use Wiring Pi which is written in C but has some extensions (wrappers) in other languages.

For our demo here spidev library is recommended to be used as this library is provided by default in linux operating systems. The following example (Source Link is pointed to at beginning) shows how to use spidev to command the spi driver of Raspberry Pi.

```
/*
 This Code is based on the following link
 http://www.raspberry-projects.com/pi/programming-in-c/spi/using-the-spi-
    interface
 */


#include <fcntl.h>            //Needed for SPI port
#include <sys/ioctl.h>        //Needed for SPI port
#include <linux/spi/spidev.h> //Needed for SPI port
#include <unistd.h>           //Needed for SPI port
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <iostream>
#include <unistd.h>
#include <cstring>

int spi_cs0_fd;           //file descriptor for the SPI device
int spi_cs1_fd;           //file descriptor for the SPI device
unsigned char spi_mode;
unsigned char spi_bitsPerWord;
unsigned int spi_speed;

//***********************************
//***********************************
//********** SPI OPEN PORT **********
//***********************************
```

```
//**********************************
//spi_device    0=CS0, 1=CS1

int SpiOpenPort (int spi_device) {
    int status_value = -1;
    int *spi_cs_fd;

//----- SET SPI MODE -----
//SPI_MODE_0 (0,0)  CPOL = 0, CPHA = 0,
//Input data on rising edge, output data on falling edge

//SPI_MODE_1 (0,1)  CPOL = 0, CPHA = 1,
//Input data on falling edge, output data on rising edge

//SPI_MODE_2 (1,0)  CPOL = 1, CPHA = 0,
//Input data on falling edge, output data on rising edge

//SPI_MODE_3 (1,1)  CPOL = 1, CPHA = 1,
//Input data on rising edge, output data on falling edge

spi_mode = SPI_MODE_0;

//----- SET BITS PER WORD -----
spi_bitsPerWord = 8;

//----- SET SPI BUS SPEED -----
spi_speed = 1000000;      //1000000 = 1MHz (1uS per bit)

if (spi_device)
    spi_cs_fd = &spi_cs1_fd;
else
    spi_cs_fd = &spi_cs0_fd;

if (spi_device)
    *spi_cs_fd = open(std::string("/dev/spidev0.1").c_str(), O_RDWR);
else
    *spi_cs_fd = open(std::string("/dev/spidev0.0").c_str(), O_RDWR);

 if (*spi_cs_fd < 0)     {
        perror("Error - Could not open SPI device");
        exit(1);
    }

status_value = ioctl(*spi_cs_fd, SPI_IOC_WR_MODE, &spi_mode);
if(status_value < 0)     {
    perror("Could not set SPIMode (WR)...ioctl fail");
    exit(1);
}

status_value = ioctl(*spi_cs_fd, SPI_IOC_RD_MODE, &spi_mode);
if(status_value < 0)     {
    perror("Could not set SPIMode (RD)...ioctl fail");
```

```
    exit(1);
}

status_value = ioctl(*spi_cs_fd, SPI_IOC_WR_BITS_PER_WORD,  &spi_bitsPerWord);
if(status_value < 0)     {
    perror("Could not set SPI bitsPerWord (WR)...ioctl fail");
    exit(1);
}

status_value = ioctl(*spi_cs_fd, SPI_IOC_RD_BITS_PER_WORD,  &spi_bitsPerWord);
if(status_value < 0)     {
    perror("Could not set SPI bitsPerWord(RD)...ioctl fail");
    exit(1);
}

status_value = ioctl(*spi_cs_fd, SPI_IOC_WR_MAX_SPEED_HZ, &spi_speed);
if(status_value < 0)     {
    perror("Could not set SPI speed (WR)...ioctl fail");
    exit(1);
}

status_value = ioctl(*spi_cs_fd, SPI_IOC_RD_MAX_SPEED_HZ, &spi_speed);
if(status_value < 0)     {
    perror("Could not set SPI speed (RD)...ioctl fail");
    exit(1);
}

return(status_value);
}

//***********************************
//***********************************
//********** SPI CLOSE PORT **********
//***********************************
//***********************************

int SpiClosePort (int spi_device) {
  int status_value = -1;
  int *spi_cs_fd;

  if (spi_device)
      spi_cs_fd = &spi_cs1_fd;
  else
      spi_cs_fd = &spi_cs0_fd;

  status_value = close(*spi_cs_fd);

  if(status_value < 0)     {
      perror("Error - Could not close SPI device");
      exit(1);
  }
  return(status_value);
```

```
}

//****************************************
//****************************************
//********** SPI WRITE & READ DATA **********
//****************************************
//****************************************
//data Bytes to write. Content is overwritten with bytes read.

int SpiWriteAndRead (int spi_device, unsigned char *data, int length) {
  struct spi_ioc_transfer spi[length];
  int i = 0;
  int retVal = -1;
  int *spi_cs_fd;

  if (spi_device)
      spi_cs_fd = &spi_cs1_fd;
  else
      spi_cs_fd = &spi_cs0_fd;

  //one spi transfer for each byte
  for (i = 0 ; i < length ; i++) {
      memset(&spi[i], 0, sizeof (spi[i]));
      spi[i].tx_buf = (unsigned long)(data + i); // transmit from "data"
      spi[i].rx_buf = (unsigned long)(data + i); // receive  into "data"
      spi[i].len = sizeof(*(data + i));
      spi[i].delay_usecs = 0;
      spi[i].speed_hz = spi_speed;
      spi[i].bits_per_word = spi_bitsPerWord;
      spi[i].cs_change = 0;
  }

  retVal = ioctl(*spi_cs_fd, SPI_IOC_MESSAGE(length), &spi);

  if(retVal < 0) {
      perror("Error - Problem transmitting spi data..ioctl");
      exit(1);
  }
  return retVal;
}
```

Bibliography

[1] https://www.raspberrypi.org/

[2] https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

[3] http://www.raspberry-projects.com/pi/programming-in-c/spi/using-the-spi-interface