18. SPI communication

Some sensors implement SPI (Serial Peripheral Interface) protocol for data transfer. An example of communication between a microcontroller and an accelerometer sensor using the SPI interface will be demonstrated in this example.

The SPI protocol basically defines a bus with four wires (four signals) and a common ground. There is one master device controlling the activity on the bus, and one slave device. The slave is active only when one of the signals, Slave Select (SS) enables it. This signal is always provided by the master. There can be more than one slave connected to the SPI bus, but each slave requires its own Slave Select signal, see Fig. 1. The data gets transferred serially bit-by-bit. There are basically two signals to carry information, one from master to slave (MOSI, Master Output Slave Input, driven by master), and one for the opposite direction (MISO, Master Input Slave Output, driven by slave). The last signal SCLK (Serial CLocK) assures the time synchronization between master and slave, and is always driven by master. There are streamlined versions of the SPI bus using only one signal to transfer data, but the direction of data must be reversed on request; we will not use this kind of data transfer.

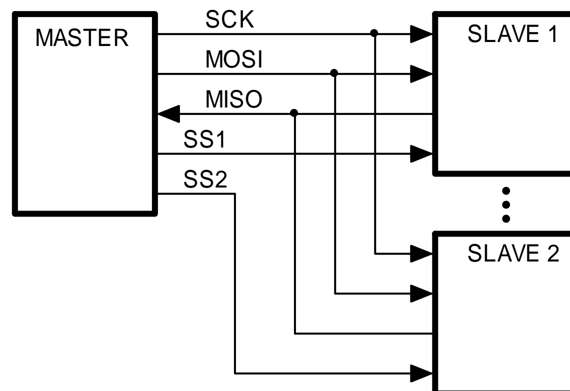

Fig. 1: The SPI bus can be used to transfer data between the master and a slave using four wires.

The speed of data transfer is higher than with I^2C bus, since the slave is selected using a hardware signal and there is no need to transfer the address of the slave. However, this results in multiple Slave Select signals where more than one slave is connected to the bus. The speed of transfer is typically higher also due to the outputs which should be able to force signals low or high, contrary to the open-drain outputs used at I^2C which can force signals low only. The logic levels are defined by the power supply of the devices connected to the SPI bus, and are 3.5V in our case. The actual speed of transmission conforms to the slowest device on the bus, as with I^2C bus.

The SPI protocol is far less strict that the I2C protocol also due to the fact that it was implemented first by several different companies and standardized only later. Variants of clock polarities, edge synchronizations, and even number of bits per transfer are used, and the designer should adopt its hardware to the SPI devices used. The microcontroller used here can implement only some possible variants of the standard, and the variant implemented in the accelerometer used is not one of them. To avoid the problem a simple “bit-banging” function combining writing to and reading from a slave

will be prepared to implement SPI communication as understood by the accelerometer. The accelerometer LIS3LV02DL is used.

The timing diagram of the required signals is given in Fig. 2, the writing being shown in the upper half. Slave select signal must first be forced low by the master, then a series of 2 times eight clock pulses are issued by the master at the SCK signal. After this the signal SCK is first returned high, followed by the signal slave select SS. The value of the signal MOSI is clocked into the slave on positive edge of the clock signal SCK, and the MOSI signal can change either before or after the clocking edge, see slave device data sheet for setup and hold times. The first eight bits start with a bit to define either writing to slave (low) or reading from slave (high); this bit is low in our case. The next bit is fixed to zero, followed by six bits of address. This is the address to be used by the slave device to select one of the internal locations for writing, not the address of the slave on the bus, as with I^2C bus. Here, with SPI bus, the device is selected using the Slave Select signal! Next eight bits are simply the byte as it is supposed to be written into the slave. In the diagram address 20_h is selected for writing, and value 40_h is written. The signal MISO is not important during the writing, and is not shown.

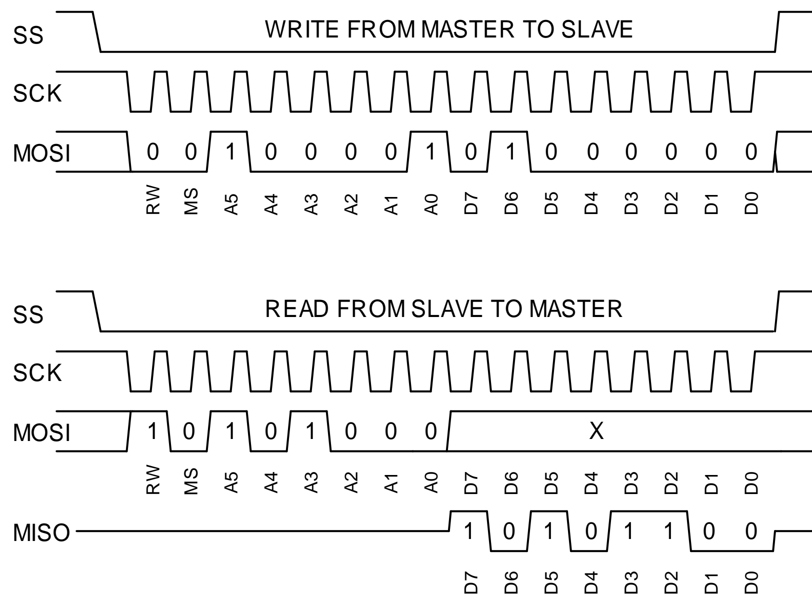

Figure 2: SPI bus signals during the writing into the slave (top), and reading from the slave (bottom)

The timing diagram for reading is given in the same figure, bottom half. The Slave Select signal and the 16 rising edges of the clock are the same as for writing, signals MOSI and MISO are different. The master first sends a command to read from the slave by forcing the first bit of the first byte at MOSI signal high, and then pulling the same signal low during the second bit. Next come six bits of address within the slave, here 28_h. After the first byte the MOSI signal is not important anymore. It can be left floating as shown in the figure, but it can also have any other logic value; tha value will be ignored by the slave. However, clock pulses are still coming, and the slave now drives the MISO signal returning the byte to be read, shown as AC_h. This sequence should be read by the microcontroller and combined into a byte.

Both sequences for writing and reading are equal with respect to Slave Select and clock signals, and similar with respect of first byte sent over the MOSI signal. It seems rational to prepare one single function for sending and receiving a byte, and to interpret return values of this function in a proper way. When writing to slave we can ignore the value returned over the MISO signal, when

reading from slave we can send a dummy value instead of eight data bits over the MOSI signal. The complete listing of the function is given in Fig. 3.

The combined 16 bits to be sent over the MOSI signal are the argument AandD of the function.

```
char SPItransaction (int AandD)  {                  // about 7us
#define MOSI 0x8000
#define MISO 0x4000
#define SCK  0x2000
#define SEL  0x1000
int Ret = 0;
  GPIOB->ODR &= ~SEL;                               // SPI select
  for (char k = 0; k<16; k++)    {                  // for 16 bits
    if (AandD & 0x8000) GPIOB->ODR |=  MOSI;        // send address & data
    else                GPIOB->ODR &= ~MOSI;        //
    AandD <<= 1;                                    // next bit of address & data
    GPIOB->ODR &= ~SCK;                             // SCL lo
    for (int i = 0; i<10; i++) {};                  // waste 150ns
    Ret <<= 1;                                      // shift bits read right
    GPIOB->ODR |=  SCK;                             // SCL hi
    if (GPIOB->IDR & MISO) Ret++;                   // read bit and add to string
  };
  GPIOB->ODR |=  SEL;                               // SPI done
  return ((char)(Ret & 0xff));                      // return result
}
```

Figure 3: A listing of the function to write or read a byte of data using SPI bus

The function returns a “char” value as received over the MISO signal during the transmission. The function starts with some definitions of bits and the declaration and initialization of a local variable Ret. Next the Slave Select signal is pulsed low and kept as such until the end of the function.

Sixteen clock pulses are needed to transmit or receive a byte, so the function continues by a “for” loop to be repeated 16 times. In the loop the bit 15 of the argument is first checked, and then the signal MOSI is set to the value of this bit. Next the value of the argument is shifted left for one bit preparing the argument for the next loop, and the clock pulse is generated by pulling the signal SCK first low, and then high again. Some delay is added to adapt to the speed of the slave device. The last thing to do within the loop is to read the value of the signal MISO and shift-accumulate bits in the variable Ret.

In order to write to the slave device the above function should be called, and the combined address and data bytes are to be passed as the argument. The listing of the function “SPIwrite” to do this is shown in Fig. 4, top. Only six bits of argument “Adr” are used (this assures the first two of the string of 16 bits bits are low), they are shifted eight bits to the left, and then byte of data is added. The return value is ignored.

In order to read from the slave device the above function is called in a similar way (“SPIread”) but most significant of the 16 bits is set to high by adding 8000_h. This causes reading from the slave, so the returned value is used here and returned to the calling program.

```
void SPIwrite(int Adr, int Data)   {
  SPItransaction(((Adr & 0x3f) << 8) + Data);               // prepare 16 bits
}

char SPIread(int Adr)              {
  char Ret = SPItransaction(((Adr & 0x3f) << 8) + 0x8000);   // 16 bits
  return (Ret);                                             // return result
}
```

Figure 4: A listing of functions to utilize the above function for writing and reading

The complete program to read data from the accelerometer, two axes's, is given in Fig. 5. The program starts with the initialization of ports where the slave device is connected, and continues with the initialization of the slave device and the LCD screen. Then the execution of the program enters the endless loop, where the two bytes representing the readout from each axis of the accelerometer are combined (see datasheet on the accelerometer for details and registers) into a 16-bit result and sent to the screen. Some delay is added to ease the reading of results from the LCD screen.

As the initialization is concerned, the SPI bus is available at the connector K485. This is connected to port B (MOSI signal at bit 15, MISO signal at bit 14, SCK signal at bit 13, and SS signal at bit 12). This port must first be activated by enabling the clock for port B, then bits 15, 13, and 12 must be defined as outputs. This is dobe in two steps; on reset of the microcontroller the bit 15 is defined as alternate function for programming purposes, and this must be disabled by clearing MS bit of the register MODE, then all three bits of the port can be made outputs. The final step of the initilization is to set Slave Select and clock signal to high initially.

As the initialization of the accelerometer chip is concerned, the relevan info is available in the datasheet of the accelerometer. Here we only state that the accelerometer chip must be powered-up by writing $C7_h$ to addres 20_h, and then the internal updating of results should be defined by writing to address 21_h, as well as filtering disabled by writing to address 22_h for the purpose of this experiment. Other requirements may be suitable for other purposes.

```
#include "stm32f4xx.h"
#include "LCD2x16.c"

void main ()  {

  // ports init: 15-MOSI, 14-MISO, 13-SCK, 12-SEL
  RCC->AHB1ENR  |=  0x00000002;         // Enable clock for GPIOB
  GPIOB->MODER  &= ~0x80000000;         // PB 15 => input pin, no AF!
  GPIOB->MODER  |=  0x45000000;         // PB 15,13,12 => outputs
  GPIOB->ODR    |=  0xc000;             // SEL & SCK -> hi

  // Init accelerometer chip
  SPIwrite(0x20, 0xc7);                 // CR1: power-up, all axes
  SPIwrite(0x21, 0x40);                 // CR2: Block data update & 12 bit mode
  SPIwrite(0x22, 0x00);                 // CR3: no filtering

  // Init LCD & prepare display
  LCD_init();                           // Init LCD
  LCD_string("x=       mg", 0x00);      // prepare 1st row
  LCD_string("y=       mg", 0x40);      // prepare 2nd row

  // endless loop
  while (1) {
    short retX = SPIread(0x28) + (SPIread(0x29) << 8);  // x axis, both bytes
    LCD_sInt16(retX, 0x02, 1);                          // display result
    short retY = SPIread(0x2a) + (SPIread(0x2b) << 8);  // y axis, both bytes
    LCD_sInt16(retY, 0x42, 1);                          // display result
    for (int i=0; i<1000000; i++)       {};             // waste time, ~30ms
  };
}
```

Figure 5: A listing of the program to read data from sensor using the SPI bus