

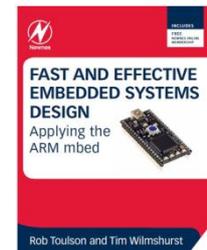
Embedded Systems Design Course

Applying the mbed microcontroller

Serial communications with SPI

These course notes are written by R.Toulson (Anglia Ruskin University) and T.Wilmshurst (University of Derby). (c) ARM 2012

These course notes accompany the textbook “Fast and effective embedded system design : Applying the ARM mbed”



Serial communications with SPI

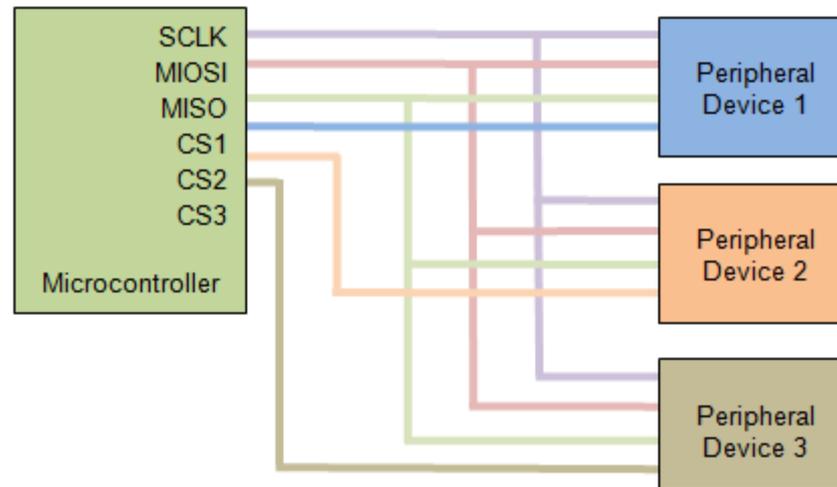
- Introducing SPI
- Evaluating simple SPI communications
- SPI on the mbed
- Evaluating the ADXL345 SPI accelerometer
- Interfacing the ADXL345 with the mbed
- Interfacing the Nokia 6610 display with the mbed
- Interfacing multiple devices on a single SPI bus
- Digital spirit level design challenge

Introducing SPI

- Serial Peripheral Interface Bus (SPI) is a serial data protocol which operates with a master/slave relationship
- When the master initiates communication and selects a slave device, data can be transferred in either or both directions simultaneously
- A master must send a byte to receive a byte - transmitting a dummy byte is a typical way to make a read transaction

Introducing SPI

- The SPI protocol uses four signals: clock (SCLK); master data output, slave data input (MOSI); master data input, slave data output (MISO); and slave chip select (CS)
- When multiple slave devices exist, the master must output a unique CS signal for each slave



Evaluating simple SPI communications

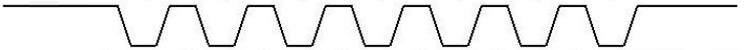
- SPI is suited to transferring data streams, for example data communication between microprocessors or data transfer from an analogue-to-digital converter
- It can achieve higher data rates than I2C, as it doesn't send address or control information, and doesn't include an acknowledge bit for every byte
- SPI data communication is ideal for use in digital signal processing applications, where data is regularly exchanged between the master and slave

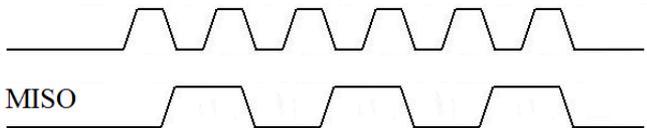
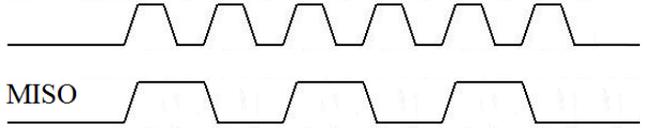
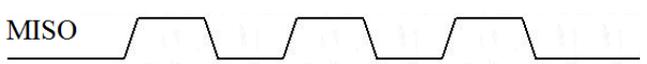
Evaluating simple SPI communications

- A SPI data transfer is initiated by the master device as follows:
 - The master first configures the SPI clock (SCLK) to be a frequency supported by the recipient slave device (up to 70 MHz)
 - The master sets the chip select signal (CS) of the intended recipient slave to 0 V
 - The master then starts the clock pulses on SCLK to indicate that data is to be transferred
 - The master simultaneously sends data as consecutive bits on MOSI
 - The number of bits in each data frame can be configured, but is usually between 4 and 16 bits
 - The slave returns data in the same manner on MISO

Evaluating simple SPI communications

- The master must also configure the clock's polarity (CPOL) and phase (CPHA) as follows:

CPOL		
0	Clock active high (off =0)	SCLK 
1	Clock active low (off=1)	SCLK 

CPHA		
0	Clock out of phase with data (CPOL = 0)	SCLK  MOSI / MISO 
1	Clock in phase with data (CPOL = 0)	SCLK  MOSI / MISO 

Evaluating simple SPI communications

- We therefore have four SPI Modes as follows:

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

- In general, SPI devices are designed to operate in one of the four modes and this will be described in the device datasheet

SPI on the mbed

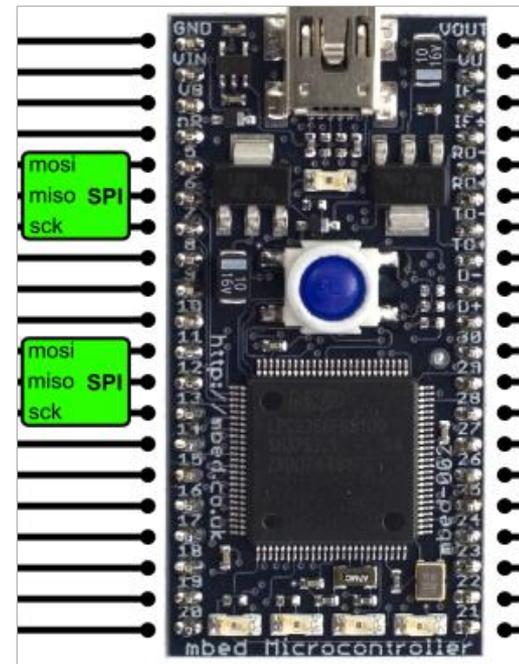
The mbed SPI library functions are shown in the table below:

SPI	A SPI Master, used for communicating with SPI slave devices
Functions	Usage
SPI	Create a SPI master connected to the specified pins format
format	Configure the data transmission mode and data length
frequency	Set the SPI bus clock frequency
write	Write to the SPI Slave and return the response

Note: this table is for the SPI master library. There is also a SPI slave library which is used for communicating with a SPI Master device. The SPISlave library is not covered in these slides.

SPI on the mbed

- The SPI Interface can be used on pins p5/p6/p7 and p11/p12/p13
- Default settings of the SPI interface on the mbed:
 - Default clock frequency of 1 MHz
 - Default data length of 8 bits
 - Default mode of 0



Evaluating the ADXL345 SPI accelerometer

- Configuration and data register details are given in the ADXL345 datasheet http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf
- The accelerometer is actually analogue in nature, measuring acceleration on 3 axes.
- This kind of accelerometer has an internal capacitor mounted in the plane of each axis. Acceleration causes the capacitor plates to move, hence changing the output voltage proportionally to the acceleration or force.
- The ADXL345 accelerometer converts the analogue voltage fluctuations to digital and outputs this over a digital communication protocol.
- The ADXL345 can be configured to communicate in SPI and I²C modes.

Evaluating the ADXL345 SPI accelerometer

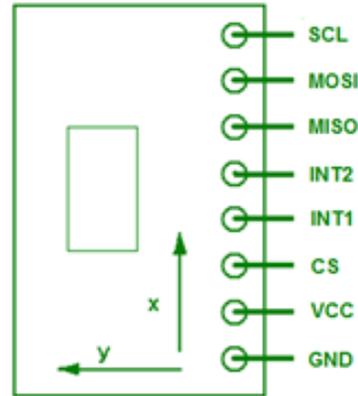
- To configure the accelerometer for SPI communication we need to:
 - Set the SPI frequency to 1-5 MHz and SPI Mode 3
 - Set the Data Format by writing to register 0x31; a data byte of 0x0B will configure the accelerometer data to be in the range $\pm 16g$ and to a resolution of 0.004g/LSB (1g is the value of acceleration due to earth's gravity, i.e. $9.81ms^{-2}$)
 - Set the device into 'measure' mode by sending a data value 0x08 to the Power Control register (address 0x2D)
- When writing to the ADXL345 we must follow the following sequential procedure
 - Set CS low
 - Send register address byte
 - Send data byte
 - Set CS high

Evaluating the ADXL345 SPI accelerometer

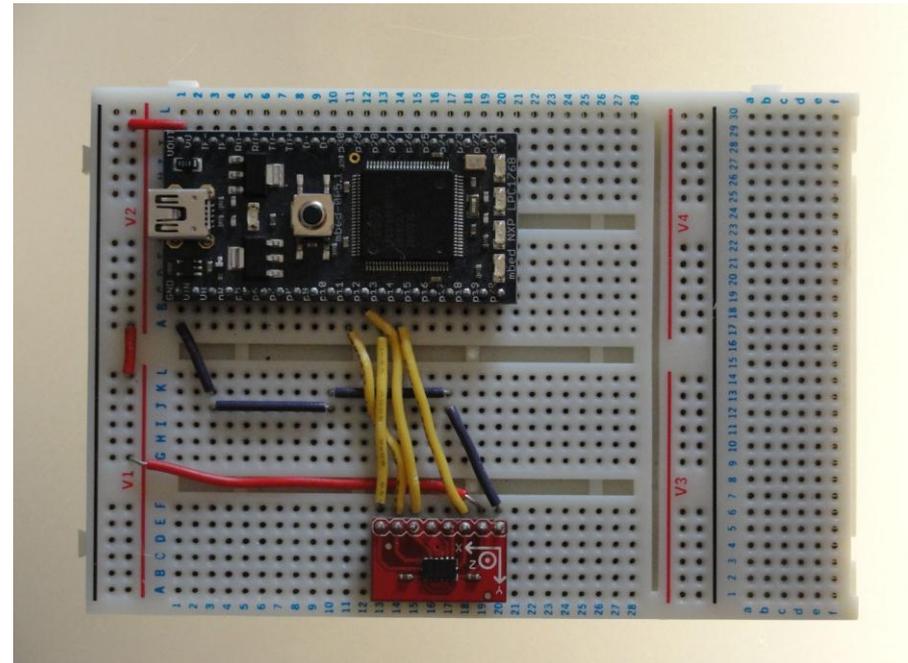
- To read 3-axis data back from the ADXL345 we must:
 - Set the read/write bit high (bit 7 of the address byte)
 - Set the multibyte-data flag high (bit 6 of the address byte)
 - Set CS low
 - Send the configured address byte for register 0x32
 - Read 6 bytes of data back from the ADXL345 by writing 6 dummy bytes of 0x00
 - Set CS high
- The 6 returned data bytes contain the most significant and least significant bytes for each of the three measurement axes (x, y and z).
- We therefore need to convert the data to floating point 'g' values by combining the relevant data bytes and multiplying by the configured data resolution.

Interfacing the ADXL345 SPI accelerometer with the mbed

- The ADXL345 can be connected to the mbed as shown:



ADXL345 signal name	mbed pin
<u>Vcc</u>	40
<u>Gnd</u>	1
SCL	13
MOSI	11
MISO	12
CS	14



Interfacing the ADXL345 SPI accelerometer with the mbed

- Exercise 1: Configure the ADXL345 accelerometer to continuously output 3-axis data to the terminal screen.
 - Set up the SPI accelerometer as described in the previous slides
 - Initiate a data read by setting CS high and writing the configured address value
 - Use a 6 element 'char' buffer to read the raw data back from the accelerometer
 - Combine the relevant most significant and least significant bytes into three two-complement, 16-bit integer values (to keep things simple, you can make use of the `16int_t` data type for this to function correctly)
 - Convert the 16-bit data to floating point by multiplying each axis reading by the configured data resolution
 - Output and display data to a host terminal application

```

// solution for SPI Exercise 1
#include "mbed.h"
SPI acc(p11,p12,p13);           // setup SPI interface on pins 11,12,13
DigitalOut cs(p14);           // use pin 14 as chip select
Serial pc(USBTX, USBRX);      // setup USB interface to host terminal
char buffer[6];               // raw data array type char
signed short data[3];         // acc data is signed 16 bit from -32,768 to +32,767
float x, y, z;                // floating point data
int main() {
    cs=1;
    acc.format(8,3);           // 8 bit data, Mode 3
    acc.frequency(2000000);    // 2MHz clock rate
    cs=0;
    acc.write(0x31);           // data format register
    acc.write(0x0B);           // format +/-16g, 0.004g/LSB
    cs=1;
    cs=0;
    acc.write(0x2D);           // power ctrl register
    acc.write(0x08);           // measure mode
    cs=1;
    while (1) {                // infinite loop
        wait(0.2);
        cs=0;
        acc.write(0x80|0x40|0x32); // RW bit high, MB bit high, plus address
        for (int i = 0;i<=5;i++) {
            buffer[i]=acc.write(0x00); // read back 6 data bytes
        }
        cs=1;
        data[0] = buffer[1]<<8 | buffer[0]; // combine MSB and LSB
        data[1] = buffer[3]<<8 | buffer[2];
        data[2] = buffer[5]<<8 | buffer[4];
        x=0.004*data[0]; y=0.004*data[1]; z=0.004*data[2]; // convert to floating point
        pc.printf("x = %+1.2fg\t y = %+1.2fg\t z = %+1.2fg\n\r", x, y,z); //print to screen
    }
}

```

Interfacing the ADXL345 SPI accelerometer with the mbed

- Exercise 2: Experiment with advanced configuration parameters for the ADXL345.
 - For example, the following configuration code will set the data rate to measure at 3200 Hz:

```
cs=0;
acc.write(0x2C);      // data rate register
acc.write(0x0F);      // set to 3200Hz
cs=1;
```

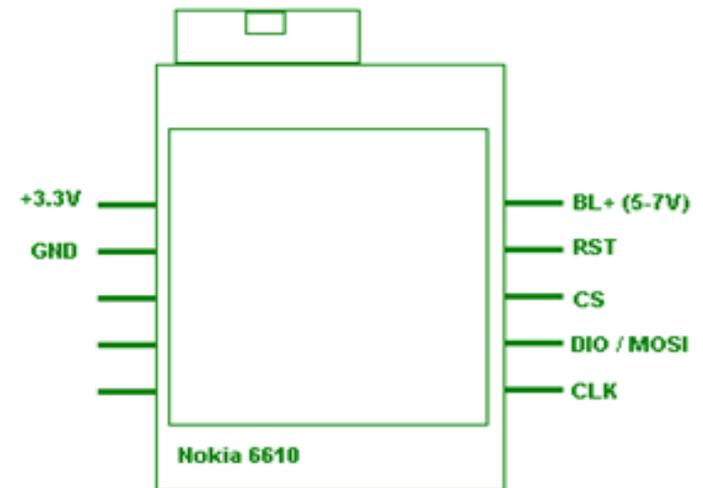
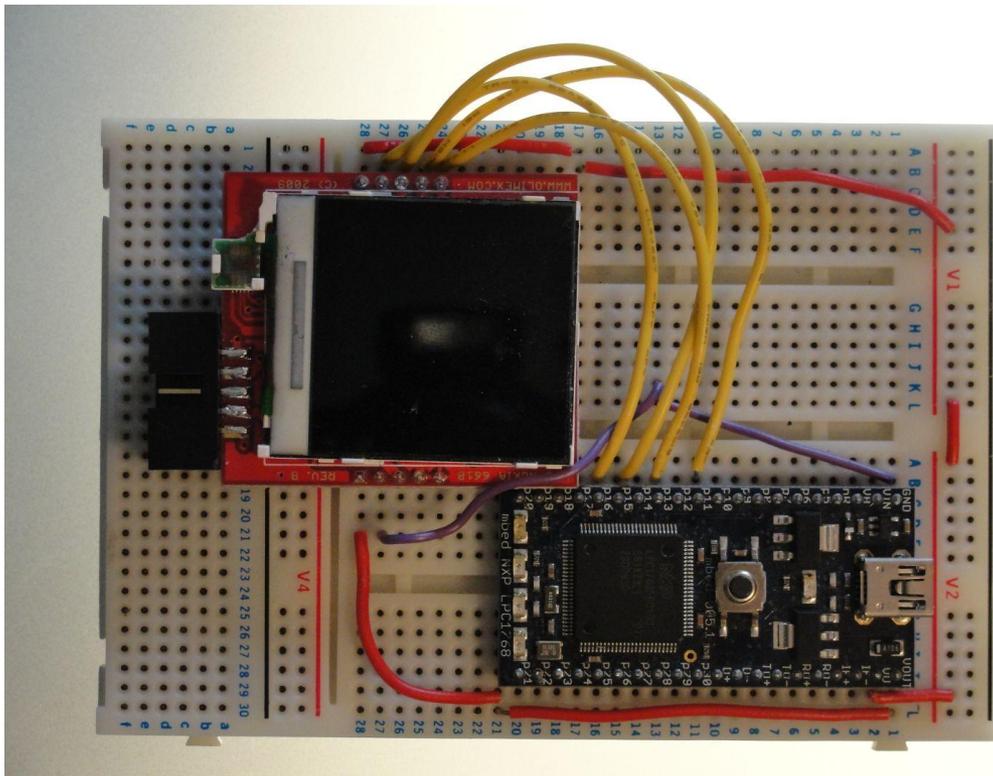
- Similarly, the following configuration will set the measurement range to $\pm 2g$:

```
cs=0; // active low
acc.write(0x31);      // data format register
acc.write(0x08);      // format +/-2g, 0.004g/LSB
cs=1;
```

- A number of other features including tap detection, freefall detection and threshold exceedance interrupts can also be configured with the ADXL345

Interfacing the Nokia 6610 LCD display with the mbed

- The mobile screen has 10 pins and can be connected to the mbed as shown:



Nokia 6610 Pin	Mbed Pin
+3.3V	40
GND	1
BL+	39
RST	16
CS	15
MOSI	11
CLK	13

Interfacing the Nokia 6610 LCD display with the mbed

- Exercise 3: Display text typed into the terminal application on to the LCD screen. Use the # key to clear the screen of the text you have written.
 - You'll need to define the following screen_setup function to set a background colour and clear the screen :

```
void screen_setup(void) {           // define a function called screen_setup
    lcd.background(0x0000FF);       // set the background colour
    lcd.cls();                       // clear the screen
}
```

- You'll also need to import the MobileLCD library from <http://mbed.co.uk/projects/cookbook/svn/MobileLCD/tests/MobileLCD>
- Note also that the cursor can be moved to a chosen position to allow you to choose where to display data, for example:

```
lcd.locate(3,1);                    // move cursor to row 1 column 3
```

```

// solution for SPI Exercise 3
#include "mbed.h"
#include "MobileLCD.h"           //include the Nokia display library

MobileLCD lcd(p11, p12, p13, p15, p16); //mosi,miso,clk,cs,rst
Serial pc(USBTX, USBRX);        // host terminal comms setup
char c;                          // char variable for keyboard input
void screen_setup(void);        // function proptotype

int main() {
    pc.printf("\n\rType something to be displayed:\n\r");
    screen_setup();             // call the screen setup function

    while(1){
        c = pc.getc();          // c = character input from computer keyboard
        wait(0.001);
        if (c=='#'){            // perform the following if "#" is pressed
            screen_setup();     // call the screen setup function
            lcd.locate(0,0);     // move the cursor back to row 0 column 0
        }
        else{
            lcd.printf("%c",c);  // print character on the LCD screen
            pc.printf("%c",c);   // print character on the terminal screen
        }
    }

void screen_setup(void) {      // define a function called screen_setup
    lcd.background(0x0000FF);  // set the background colour
    lcd.cls();                 // clear the screen
}

```

Interfacing the Nokia 6610 LCD display with the mbed

- Exercise 4: Experiment with the fill and pixel commands to draw on the LCD Display. For example:

- Use the following fill functions in order to fill some areas of the LCD display:

```
lcd.fill(2, 51, 128, 10, 0x00FF00); //fill an area between the defined pixels
lcd.fill(50, 1, 10, 128, 0xFF0000);
```

- Draw on the screen pixel by pixel. The following loop will create the function for a sine wave and print this to the screen:

```
for(int i=0; i<130; i++) {
    //draw a black pixel at the location described by the following formula
    lcd.pixel(i, 80 + sin((float)i / 5.0)*10, 0x000000);
}
```

- Note the use of colour as a single 24-bit value for red, green and blue (8-bits each)

0xFF0000 = red 0x00FF00 = green 0x0000FF = blue 0x000000 = black 0xFFFFFFFF = white

Interfacing multiple devices on a single SPI bus

- Exercise 5: You can use a single SPI data bus to control the ADXL345 accelerometer and the Nokia 6610 display at the same time. Write a program so that the x, y and z data appear and update on the LCD display, rather than on a host pc.
- You will need a separate slave chip select signal for each device.

Digital spirit level design challenge

- Exercise 6: Design, build and test a digital spirit level based on the mbed microcontroller, the ADXL345 accelerometer and the Nokia 6610 display
- You may wish to consider the following:
 - Design your display to show a pixel or image moving around the LCD screen with respect to the orientation of the accelerometer
 - Improve your display output to include accurate measurements of the 2-plane orientation angles in degrees from the horizontal (i.e. horizontal = 0°)
 - You may need to include some calibration or filtering techniques to ensure smooth and accurate functionality

Summary

- Introducing SPI
- Evaluating simple SPI communications
- SPI on the mbed
- Evaluating the ADXL345 SPI accelerometer
- Interfacing the ADXL345 with the mbed
- Interfacing the Nokia 6610 display with the mbed
- Interfacing multiple devices on a single SPI bus
- Digital spirit level design challenge