# **Embedded Systems Essentials with Arm: Get Practical with Hardware**

# Module 1

KV4: Using the Mbed API in Synchronous Serial Communication

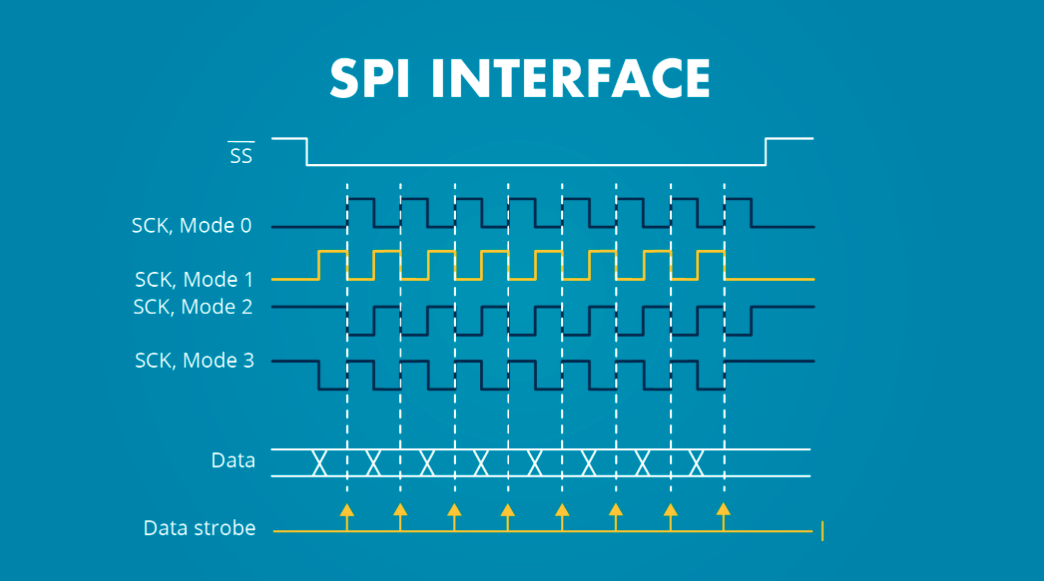
This video looks at how Mbed devices can be configured as SPI (serial peripheral) master or slave. More often you want your device to be a master because this allows you to control the communication. With other devices, you don’t have the ability to control the communication. These are some, but not all, of the functions that the Mbed API provides to the SPI peripheral as a master:

|  |  |
| --- | --- |
| **Function Name​** | **Description ​** |
| **SPI** (PinName mosi, PinName miso, PinName sclk, PinName \_unused=NC)​ | Create a SPI master connected to the specified pins​ |
| void **format** (int bits, int mode)​ | Configure the data transmission format (Mode default = 0)​ |
| void **frequency** (int hz)​ | Set the spi bus clock frequency (default 1 MHz)​ |
| virtual int **write** (int value)​ | Write to the SPI Slave and return the response​ |

The SPI Interface can be used to write data words out from the SPI port and return the data from the SPI slave.​ It’s possible to configure the SPI clock frequency and format to suit your specific needs. In addition to the frequency, the master can also configure the mode. ​

If you don’t need to set up a specific configuration, you can use the default settings of the SPI interface: 1MHz, 8-bit, Mode 0.​

The mode is a feature of SPI which allows you to choose which clock edge is used to clock data into the shift register—this is indicated as “Data strobe” in the diagram. Mode determines whether it’s the rising edge or the falling edge which clocks the data.



In this diagram Mode 0 is the rising edge and Mode 1 is the falling edge.

This example shows an SPI Master being configured. A single byte is sent to the selected slave, which then responds with the requested ID byte.

|  |
| --- |
| #include "mbed.h"​  ​  SPI spi(*SPI\_MOSI, SPI\_MISO, SPI\_SCLK*); ​  DigitalOut cs(SPI\_CS);​  ​  int main() {​  cs = 1; // Start with Slave deselected​  // Setup the spi for 8 bit data, high steady state clock,​  // second edge capture, with a 1MHz clock rate​  spi.format(8,3);​  spi.frequency(1000000);​  cs = 0; // Select the Slave by setting chip select low​  // Send 0x8f, the command to read the WHOAMI register​  spi.write(0x8F);​  // Send a dummy byte to receive the contents of the WHOAMI register​  int whoami = spi.write(0x00);​  printf("WHOAMI register = 0x%X\n", whoami);​  cs = 1; // Deselect the device​  } |

We’re defining the SPI master here with the SPI terminology and what pins it connects to. Enter the pin numbers according to the SPI port on the device you chose. We’ve also created a digital out, which we’ve labelled CS, for Chip Select.

We start with CS at 1. When it’s at 1, the slave is not selected.

We set up the format and the communication frequency. Then we move on to CS at 0 —now we’re selecting the slave. We use the spi.write command to send a command that this particular slave will recognise. We do another spi.write. What we‘re sending is not of interest to the slave, but that spi.write command triggers a data exchange, the Master receives the data word and stores the whoami register contents which we print to the screen. We then set CS to 1 and we’ve completed the communication.

This is an example of a device we can use with SPI. 

This is a three-axis accelerometer placed on a breakout board. It has MOSI, MISO and serial clock pins which give it SPI capability. The other pins shown extend its capability beyond SPI.

Mbed also has good capability for application with UARTs. Mbed can set up the standard characteristics such as baud rate, data length, the use of parity, number of stop bits. If you don’t want to add these in manually, you use the default settings for the mbed microcontroller, described as 9600 8N1. This translates to nine thousand, six hundred bits per second, 8 data bits, no parity, 1 stop bit.

This example shows the start of a program which configures a serial port and then creates a word which it transmits.

|  |
| --- |
| #include "mbed.h"​  Serial async\_port(p9, p10); //set up TX and RX on pins 9 and 10​  DigitalOut red\_led(p25); //red led​  DigitalOut green\_led(p26); //green led​  DigitalIn switch\_ip1(p5);​  DigitalIn switch\_ip2(p6);​  char switch\_word ; //the word we will send​  char recd\_val; //the received value ​  ​  int main() {​  async\_port.baud(19200); //set baud rate to 19200​  //accept default format, of 8 bits, no parity​  while (1){​  //Set up the word to be sent, by testing switch inputs​  switch\_word=0xa0; //set up a recognisable output pattern​  if (switch\_ip1==1)​  switch\_word=switch\_word|0x01; //OR in lsb​  if (switch\_ip2==1)​  switch\_word=switch\_word|0x02; //OR in next lsb​  async\_port.putc(switch\_word); //transmit switch\_word​  if (async\_port.readable()==1) //is there a character to be read?​  recd\_val=async\_port.getc(); //if yes, then read it​  ... |

It then tests if any data has been returned to it. In the remainder of the program, which is not shown, it configures its LED display according to the data it receives.

**Visit arm.com for more information.**