Name: ..... Student ID: .....
Name: ..... Student ID: .....

Laboratory 07
Introduction to Serial Communication

1. Introducing Serial Data Communication

In serial data transfer, one bit of the data is transmitted at a time, along a single interconnection, accompanied by as many other interconnections as are needed to define bit timing, word framing, and other control data. While slower than parallel transfer, the small number of wires needed is a huge advantage, in terms of pcb tracks, interconnecting wires, and IC pins.

The receiver know when each bit begins and ends, and how does it know when each word begins and ends, by sending a clock signal alongside the data, with one clock pulse per data bit. The data is *synchronised* to the clock. This idea, called synchronous serial communication.

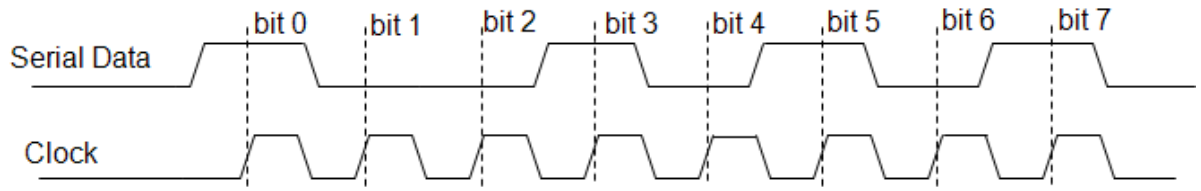


Fig.1. Synchronous serial data

2. A simple serial link

Node 1 is designated *Master*; it controls what's going on, as it controls the clock. The *Slave* is similar to the Master, but receives the clock signal from the Master.

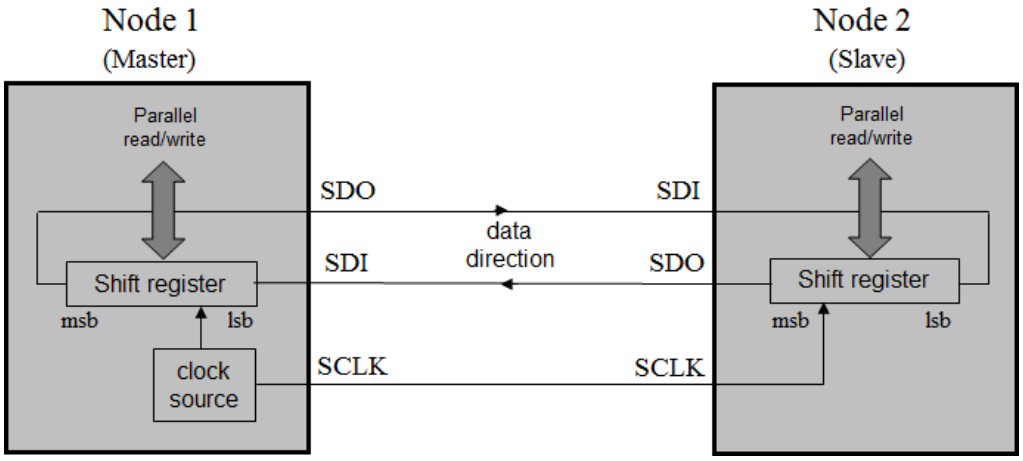


Fig. 2. A Simple Serial Link

3. Serial Peripheral Interface (SPI)

An SPI network with one Master and multiple slaves can be constructed as shown.

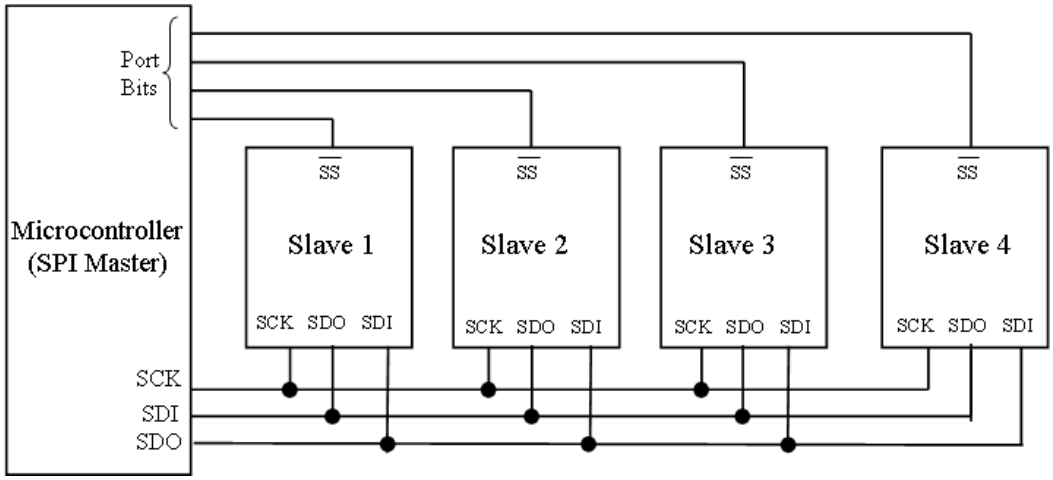


Fig. 3. Serial Peripheral Interface (SPI)

4.1 SPI on the LPC1769: Master

The LPC1769 has two SPI ports, each can be configured as Master or Slave. The API summary for SPI Master is shown. On the LPC1769, as with many SPI devices, the same pin is used for SDI if in Master mode, or SDO if Slave. Hence this pin gets to be called MISO, Master in, Slave out. Its partner pin is MOSI.

Summary of API for SPI Master

Functions	Usage
SPI	Create a SPI master connected to the specified pins
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

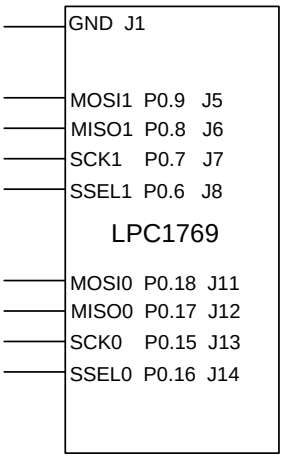
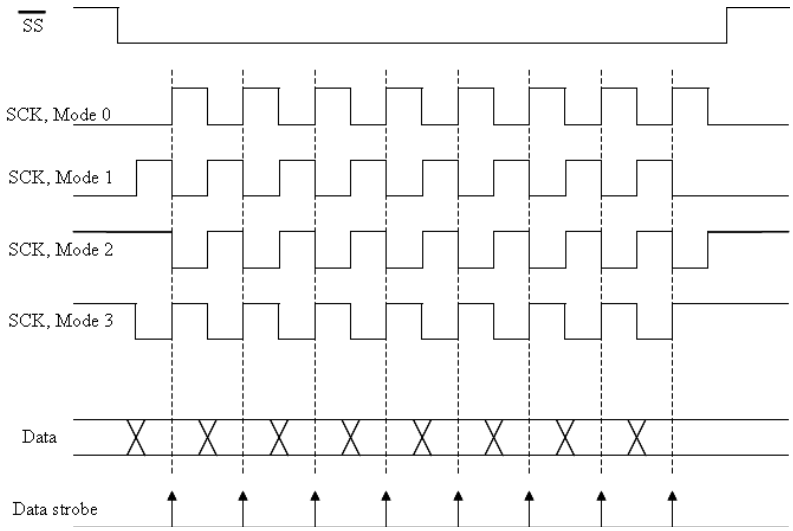


Fig. 4. LPC1769 Xpresso Board Pin out for SPI

4.2. Clocking Mode

The mode is a feature of SPI which allows choice of which clock edge is used to clock data into the shift register (indicated as "Data strobe" in the diagram), and whether the clock idles high or low. For most applications the default mode, i.e. Mode 0, is acceptable.



Mode	Polarity	Phase
0	0	0
1	0	1
2	1	0
3	1	1

Fig. 5. SPI Clocking Mode

4.3. Simple SPI Master Program

Example 1

```
// Sets up the LPC1769 as SPI master, and continuously sends a single byte

#include "mbed.h"

SPI ser_port(p0_18, p0_17, p0_15);        // mosi0, miso0, sclk0
char switch_word;                    //word we will send
int main() {
  ser_port.format(8,0); // Setup the SPI for 8 bit data, Mode 0 operation
  ser_port.frequency(1000000);       // Clock frequency is 1MHz
  while (1){
    switch_word=0xA1;                //set up word to be transmitted
    ser_port.write(switch_word);     //send switch_word
    wait_us(50);
  }
}
```

4.4. Creating an SPI data link: Master

We will now develop two programs; one master and one slave, and get two Xpresso boards to communicate. Each will have two switches and two light-emitting diodes (LEDs); the aim will be to get the switches of the master to control the LEDs on the slave, and vice versa. To simplify initialization of GPIO pin, we use mbed function **DigitalOut** and **DigitalIn** which are summarized as follows

Summary of API for Digital Input/output

Functions	Usage
DigitalOut	Create a DigitalOut connected to the specified pin
DigitalIn	Create a DigitalIn connected to the specified pin

Example 2

```
/* Sets the LPC1769 up as Master, and exchanges data with a slave, sending
its own switch positions, and displaying those of the slave.*/

#include " mbed.h"
SPI ser_port(p0_18, p0_17, p0_15);    // mosi, miso, sclk
DigitalOut red_led(p0_2);   //red led
DigitalOut green_led(p0_3); //green led
DigitalOut cs(p14);         //this acts as "slave select"
DigitalIn  switch_ip1(p0_9);
DigitalIn  switch_ip2(p0_8);
char switch_word ;         //word we will send
char recd_val;             //value return from slave
int main() {
  while (1){
    //Default settings for SPI Master chosen
    //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
    cs = 0;                                //select slave
    recd_val=ser_port.write(switch_word); //send switch_word and receive
data
    cs = 1;
    wait(0.01);
//set leds according to incoming word from slave
    red_led=0;              //preset both to 0
    green_led=0;
    recd_val=recd_val&0x03; //AND out unwanted bits
    if (recd_val==1)
      red_led=1;
    if (recd_val==2)
      green_led=1;
    if (recd_val==3){
      red_led=1;
      green_led=1;
    }
  }
}
```

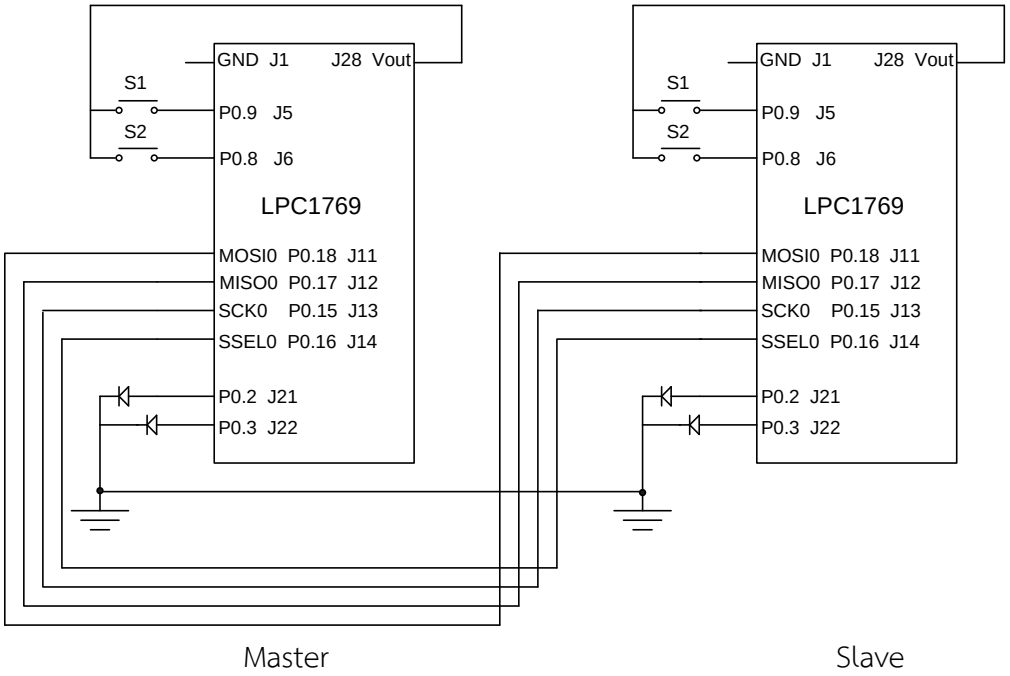


Fig. 6. SPI Master Slave connection

4.5. SPI on the LPC1769: Slave

It is almost the mirror image of the Master program, with small but key differences. It also declares variables **switch_word** and **recd_val.** The Slave program configures its **switch_word** just like the Master. While the Master initiates a transmission when it wishes, the Slave must wait. The mbed library does this with the **receive( )** function. This returns 1 if data has been received, and 0 otherwise. If data has been received from the Master, then data has also been sent from Slave to Master.

Functions	Usage
SPISlave	Create a SPI slave connected to the specified pins
format	Configure the data transmission format
frequency	Set the SPI bus clock frequency
receive	Polls the SPI to see if data has been received
read	Retrieve data from receive buffer as slave
reply	Fill the transmission buffer with the value to be written out as slave on the next received message from the master.

Example 3

```
/* Sets the LPC1769 up as Slave, and exchanges data with a Master, sending
its own switch positions, and displaying those of the Master. as SPI slave.
*/
#include "mbed.h"
SPISlave ser_port(p0_18,p0_17,p0_15,p0_16); // mosi, miso, sclk, ssel
DigitalOut red_led(p_2);            //red led
DigitalOut green_led(p0_3);          //green led
DigitalIn  switch_ip1(p_9);
DigitalIn  switch_ip2(p_8);
char switch_word ;                  //word we will send
char recd_val;                      //value received from master
int main() {
  //default formatting applied
  while(1) {
    //set up switch_word from switches that are pressed
    switch_word=0xa0;             //set up a recognisable output pattern
    if (switch_ip1==1)
      switch_word=switch_word|0x01;
    if (switch_ip2==1)
      switch_word=switch_word|0x02;

    if(ser_port.receive()) {      //test if data transfer has occurred
      recd_val = ser_port.read();   // Read byte from master
      ser_port.reply(switch_word);  // Make this the next reply
    }
    //now set leds according to received word
    ...
    (continues as in Example 2 Program)
    ...
  }
}
```

4.6. SPI Limitation

The SPI standard is extremely effective. The electronic hardware is simple and therefore cheap, and data can be transferred rapidly. There are disadvantages.

- There is no acknowledgement from the receiver, so in a simple system the Master cannot be sure that data has been received.
- There is no addressing. In a system where there are multiple slaves, a separate |SS line must be run to each Slave, as seen earlier. Therefore we begin to lose the advantage that serial communications should give us, i.e. a limited number of interconnect lines.
- There is no error-checking. Suppose some electromagnetic interference was experienced in a long data link, data or clock would be corrupted, but the system would have no way of detecting this, or correcting for it.

Overall SPI could be graded as: simple, convenient and low-cost, but not appropriate for complex or high reliability systems.

5. Inter-Integrated Circuit bus (I^2C)

I^2C aims to resolve some of the perceived weaknesses of SPI. I^2C is a serial data protocol which operates with a master/slave relationship.

- I^2C only uses two physical wires, called serial data (SDA) and serial clock (SCL). This means that data only travels in one direction at a time.
- Any node can only pull down the SCL or SDA line to Logic 0; it cannot force the line up to Logic 1. This role is played by a single pull-up resistor connected to each line.

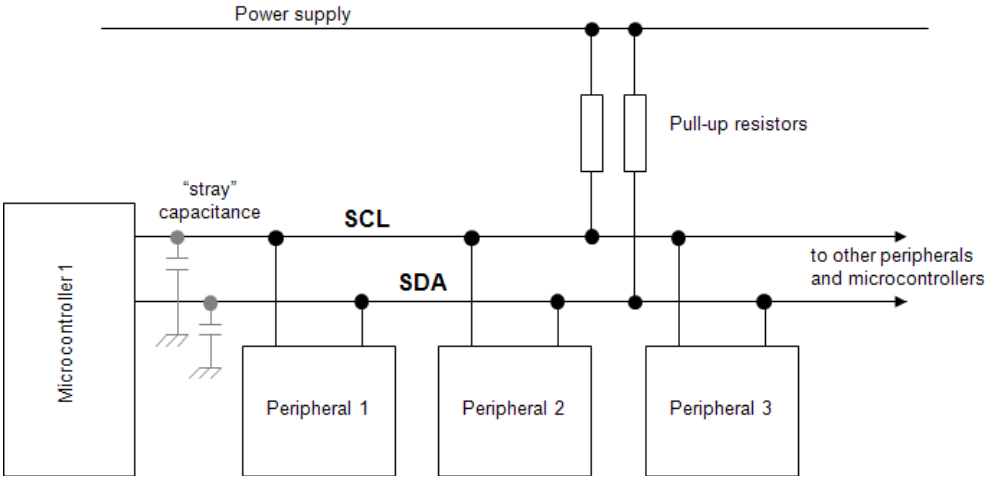


Fig.7. I2C Bus Connection

I^2C has a built-in addressing scheme, which simplifies the task of linking multiple devices together. In general, the device that initiates communication is termed the 'master'. A device being addressed by the master is called a 'slave'. Each I^2C-compatible slave device has a predefined device address. The slaves are therefore responsible for monitoring the bus and responding only to data and commands associate with their own address. This addressing method, however, limits the number of identical slave devices that can exist on a single I^2C bus, as each device must have a unique address. For some devices, only a few of the address bits are user configurable.

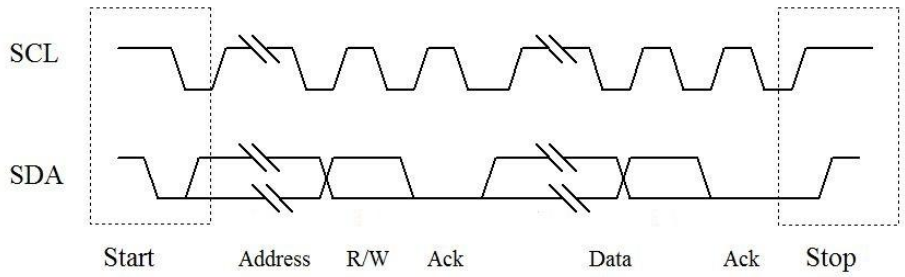


Fig.8. Simple I^2C communications

A data transfer is made up of the Master signalling a **Start Condition**, followed by one or two bytes containing address and control information. The Start condition is defined by a high to low transition of SDA when SCL is high. A low to high transition of SDA while SCL is high defines a **Stop condition**. One SCL clock pulse is generated for each SDA data bit, and data may only change when the clock is low.

The byte following the Start condition is made up of **seven address bits**, and one data direction bit (Read/Write). All data transferred is in units of one byte, with no limit on the number of bytes transferred in one message. Each byte must be followed by a 1-bit acknowledge from the receiver, during which time the transmitter relinquishes SDA control.

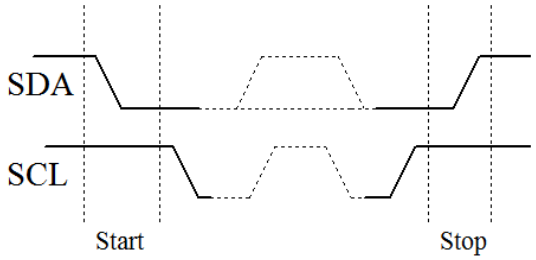


Fig.9.1 Start Condition and Stop Condition

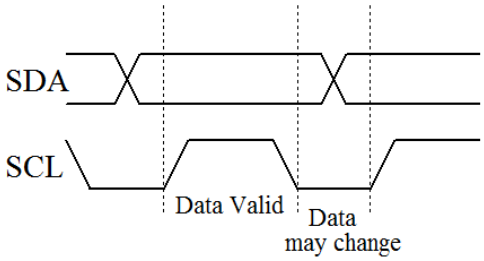


Fig.9.2. Data stable during clock high

LPC1769 has three I^2C ports however only two ports are connected to Xpresso board as shown in Figure 10. Library Master and Slave functions are shown in the Tables below. These are more complex than SPI.

Library for Master and Slave functions

Functions	Usage
I2C	Create an I^2C Master interface, connected to the specified pins
frequency	Set the frequency of the I^2C interface
read	Read from an I^2C slave
write	Write to an I^2C slave
start	Creates a start condition on the I^2C bus
stop	Creates a stop condition on the I^2C bus

Function	Usage
I2CSlave	Create an I^2C Slave interface, connected to the specified pins.
frequency	Set the frequency of the I^2C interface
receive	Checks to see if this I^2C Slave has been addressed.
read	Read from an I^2C master.

write	Write to an I^{2}C master.
address	Sets the I^{2}C slave address.
stop	Reset the I^{2}C slave back into the known ready receiving state.

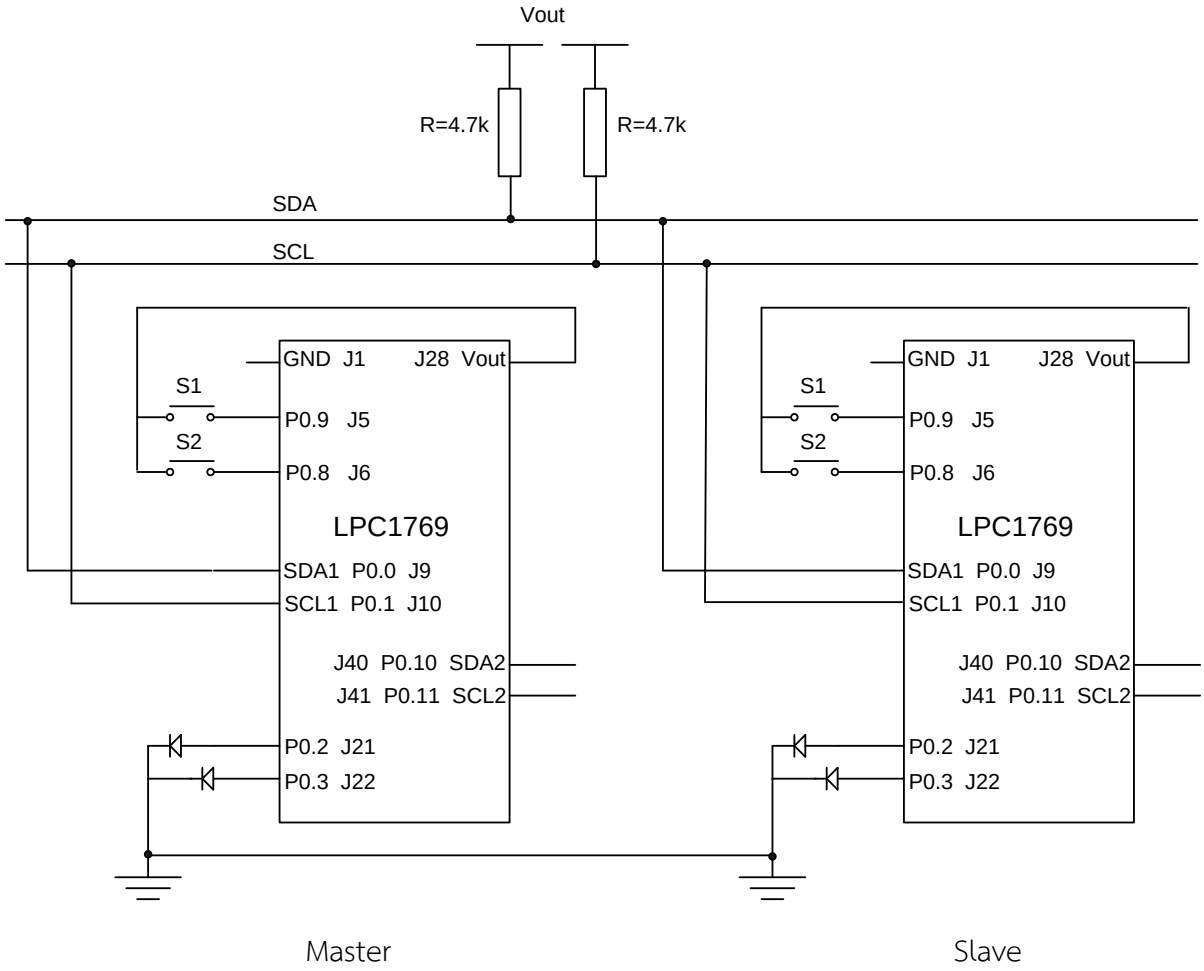


Fig.10. I2C Master slave connection

5.1 Setting up an I^{2}C LPC1769 to LPC1769 Data Link

Setting up the I^{2}C Data Link (Master)

Example 4

```
/* I2C Master, transfers switch state to second LPC1769 acting as slave,
and displays state of slave's switches on its leds.*/

#include "mbed.h"

I2C i2c_port(p0_0, p0_1);        // Configure a serial port, pins 9 and 10
are sda,scl

DigitalOut red_led(p0_2);   //red led
```

```
DigitalOut green_led(p0_3); //green led
DigitalIn  switch_ip1(p0_9); //input switch
DigitalIn  switch_ip2(p0_8);

char switch_word ;        //word we will send
char recd_val;            //value return from slave
const int addr = 0x52;    // define the I2C slave address, an arbitrary
even number

int main() {
  while(1) {
    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
    //send a single byte of data, in correct I2C package
    i2c_port.start();          //force a start condition
    i2c_port.write(addr);      //send the address
    i2c_port.write(switch_word); //send one byte of data, ie switch_word
    i2c_port.stop();           //force a stop condition
    wait(0.002);
    //receive a single byte of data, in correct I2C package
    i2c_port.start();
    i2c_port.write(addr|0x01);    //send address, with Read/Write bit set
//to Read
    recd_val=i2c_port.read(addr); //Read and save the received byte
    i2c_port.stop();              //force a stop condition
    //set leds according to word received from slave
    red_led=0;              //preset both to 0
    green_led=0;
    recd_val=recd_val&0x03; //AND out unwanted bits
    if (recd_val==1)
      red_led=1;
    if (recd_val==2)
      green_led=1;
    if (recd_val==3){
      red_led=1;
      green_led=1;
    }
    wait(0.004);
  }
}
```

Setting up the I^2C Data Link (Slave)

The slave program is similar to the SPI example, with SPI features replaced by I^2C. The I^2C slave just responds to calls from the Master. The slave port is defined with the LPC1769 utility **I2Cslave,** with **slave** chosen as the port name. Just within the **main** function the slave address is defined, the same 0x52 as in the Master program. The **receive( )** function tests if an I^2C transmission has been received. This returns a 0 if the Slave has not been addressed, a 1 if it has been addressed to read, and a 3 if addressed to write.

Example 5

```
/* I2C Slave, when called transfers switch state to LPC1769 acting as Mas-
ter, and displays state of Master's switches on its leds.   */

#include <mbed.h>

I2CSlave slave(p0_0, p0_1);  //Configure I2C slave

DigitalOut red_led(p0_2);     //red led
DigitalOut green_led(p0_3);   //green led
DigitalIn  switch_ip1(p0_9);
DigitalIn  switch_ip2(p0_8);

char switch_word ;    //word we will send
char recd_val;        //value received from master
int main() {
  slave.address(0x52);
    while (1) {
      //set up switch_word from switches that are pressed
      switch_word=0xa0;          //set up a recognisable output pattern
      if (switch_ip1==1)
        switch_word=switch_word|0x01;
      if (switch_ip2==1)
        switch_word=switch_word|0x02;
      slave.write(switch_word); //load up word to send
      //test for I2C, and act accordingly
      int i = slave.receive();
      if (i == 3){              //slave is addressed, Master will write
        recd_val= slave.read();
      //now set leds according to received word
      red_led=0;
      green_led=0;
      recd_val=recd_val&0x03;
      if (recd_val==1)
        red_led=1;
      if (recd_val==2)
        green_led=1;
      if (recd_val==3){
        red_led=1;
        green_led=1;
      }
   }              //end of while
}                 //end of main
```

6. Experiment

1. In Program **Example1**, try invoking each of the SPI clock modes in turn. Observe both clock and data waveforms on the oscilloscope, and check how they compare to Figure 5.

2. Set the SPI format of Program **Example1** to 12 and then 16 bits, sending the words 0x8A1 and 0x8AA1, respectively. Check each on an oscilloscope.

3. Connect two Xpresso boards as in Figure 6. Test with program **Example2** and **Example3**

4. Connect two Xpresso boards as in Figure 10. Test with program **Example4** and **Example5**