C for GPIO

E155

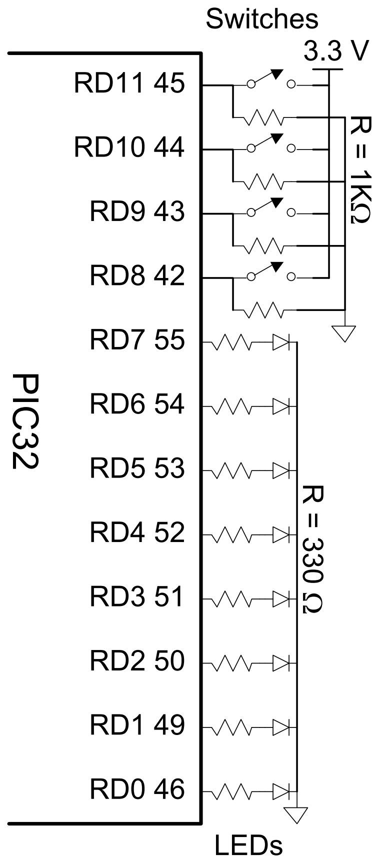
Outline

C-Programming Examples

- ❑ GPIO
- ❑ Timer
- ❑ SPI
- ❑ UART

GPIO

- Write a C program to read the four switches and turn on the corresponding bottom four LEDs using the hardware in Figure.



Read SW Write LED

```
/*Configure TRISD so that pins RD[7:0] are outputs and RD[11:8] are
inputs.
Read the switches by examining pins RD[11:8]
Write this value back to RD[3:0] to turn on the appropriate LEDs.

 c code */
#include <p32xxxx.h>
void main(void) {
  int switches;

  TRISD = 0xFF00;                   // set RD[7:0] to output, RD[11:8] to
input
  while (1) {
    switches = (PORTD >> 8) & 0xF; // Read and mask RD[7:4]
    PORTD = switches;              // display on the LEDs
  }
}
```

Count to 1 sec

- Run periphery clock at ¼ speed for 10MHz
- Set Pclk=clk/4

  Prescale=256 $\rightarrow$ each count=256*0.1us=25.6us
- Count=1sec/25.6us=39062

Count to 1 sec

```
//         Include
#include <P32xxxx.h>
#include <plib.h>
// Prototypes
void main(void);
void initTimers(void);
void main(void) {
          unsigned short duration;
          duration = 39062;   //1sec/25.6us=39062
          TRISF = 0;          // Use PORTF for output
          initTimers();       // Set up Timer1
          TMR1 = 0;           // Reset timers
          PORTFbits.RF0 = 0;  // Output low
          while (TMR1 < duration) {}    // wait until duration of 1sec is up
                 PORTFbits.RF0 = 1;     // Output high
}
```

Count to 1 sec

```
void initTimers(void) {

          //        Assumes peripheral clock at 10MHz

          //        Use Timer1 for note duration
          //        T1CON
          //        bit 15:   ON=1: enable timer
          //        bit 14: FRZ=0: keep running in exception mode
          //        bit 13: SIDL = 0: keep running in idle mode
          //        bit 12: TWDIS=1: ignore writes until current write completes
          //        bit 11: TWIP=0: don't care in synchronous mode
          //        bit 10-8: unused
          //        bit 7:    TGATE=0: disable gated accumulation
          //        bit 6:    unused
          //        bit 5-4: TCKPS=11: 1:256 prescaler, 0.1us*256=25.6us
          //        bit 3:    unused
          //        bit 2:    don't care in internal clock mode
          //        bit 1:    TCS=0: use internal peripheral clock
          //        bit 0:    unused
          T1CON = 0b1001000000110000;
```

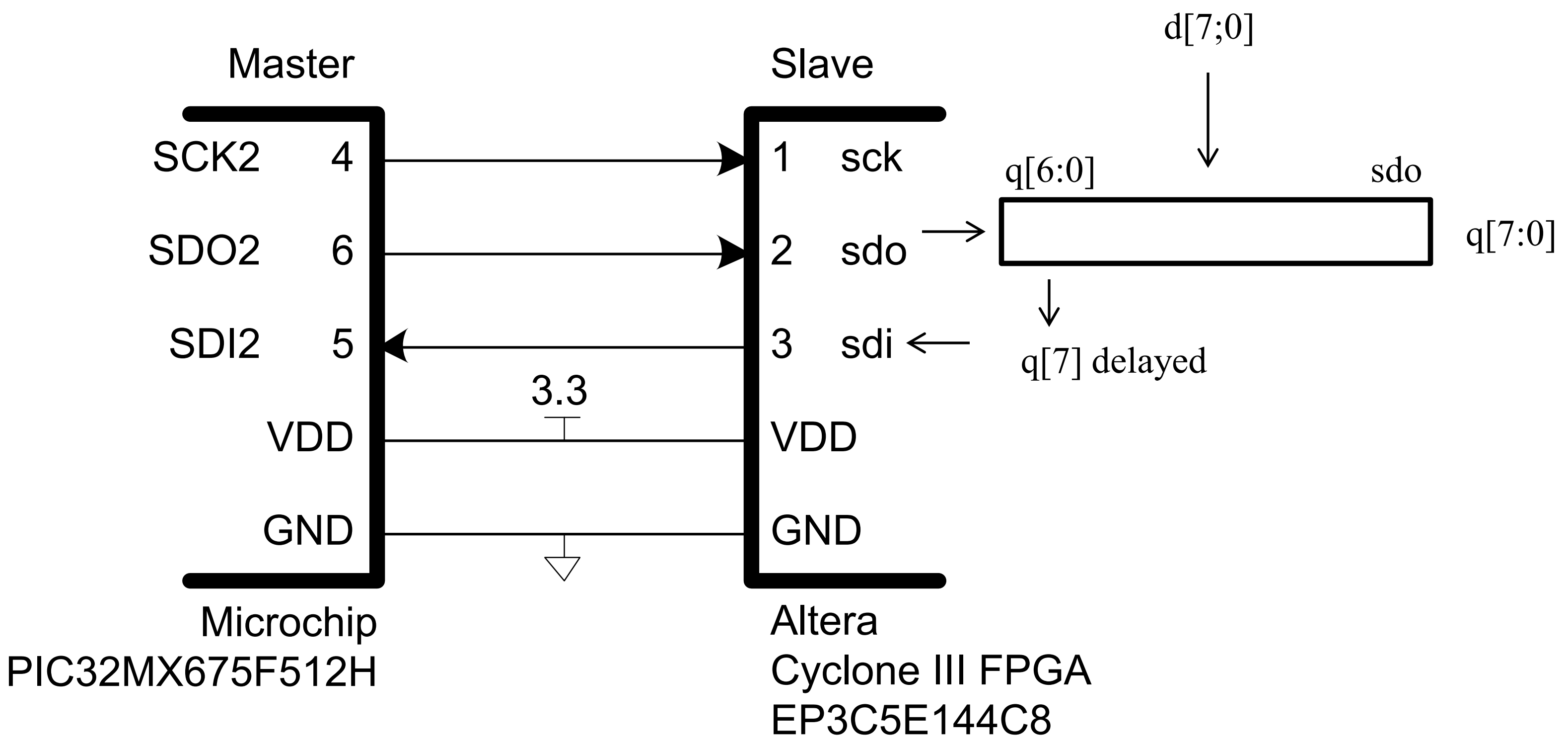
Lab5 Audio

- A music score is represented as a sequence of numbers specifying the period and duration of each note. Both are 16-bit values.
- The period is given in units of t_1 = 1.6 ms. The duration is given in units of t_0 = 51.2 ms.
- Set the peripheral clock to Fosc/8 = 5 MHz. You can configure Timer 1 to use a prescalar of 256 so that each count is 51.2 ms. Timiner 1 used for duration.
- Similarly, configure Timer 2 to use a prescalar of 4 so that each count is 0.8 ms, or half a period unit (convenient to set the number of units of time for a high output and for a low output). Timer 2 used for period.
- Both timers should use the 5 MHz peripheral clock as their source.
- C code: lab05c_la.c

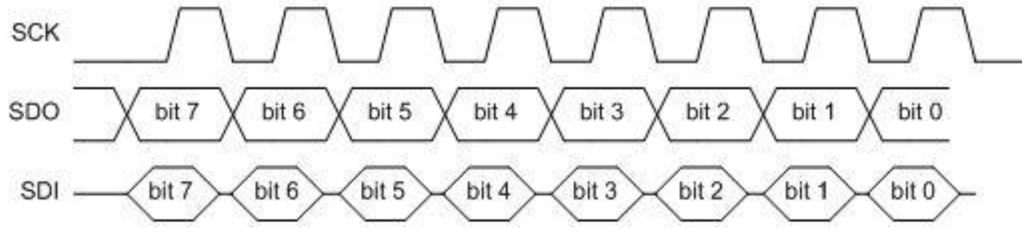
SPI

- Design a system to communicate between a PIC master and an FPGA slave over SPI.
- Write the C code for the PIC to send the character ‘A’ and receive a character back.
- Write HDL code for an SPI slave on the FPGA.
- Sketch a schematic of the interface.

SPI



Master
Slave
SCK2 4
SDO2 6
SDI2 5
VDD
GND
1 sck
2 sdo
3 sdi
VDD
GND
3.3
Microchip
PIC32MX675F512H
Altera
Cyclone III FPGA
EP3C5E144C8
d[7;0]
q[6:0]
sdo
q[7:0]
q[7] delayed



SCK
SDO
SDI
bit 7
bit 6
bit 5
bit 4
bit 3
bit 2
bit 1
bit 0

SPI Master

```
#include <p32xxxx.h>
void initspi(void) {
  char junk;

  SPI2CONbits.ON = 0;    // disable SPI to reset any previous state
  junk = SPI2BUF;        // read SPI buffer to clear the receive buffer
  SPI2BRG = 7;           //set BAUD rate to 1.25MHz, with Pclk at 20MHz
  SPI2CONbits.MSTEN = 1; // enable master mode
  SPI2CONbits.CKE = 1;   // set clock-to-data timing (data centered on rising SCK edge)
  SPI2CONbits.ON = 1;    // turn SPI on
}
 char spi_send_receive(char send) {
  SPI2BUF = send;             // send data to slave
  while (!SPI2STATbits.SPIBUSY); // wait until received buffer fills, indicating data received
  return SPI2BUF;             // return received data and clear the read buffer full
}

void main(void) {
  char received;

  initspi();                  // initialize the SPI port
  received = spi_send_receive('A'); // send letter A and receive byte back from slave
}
```

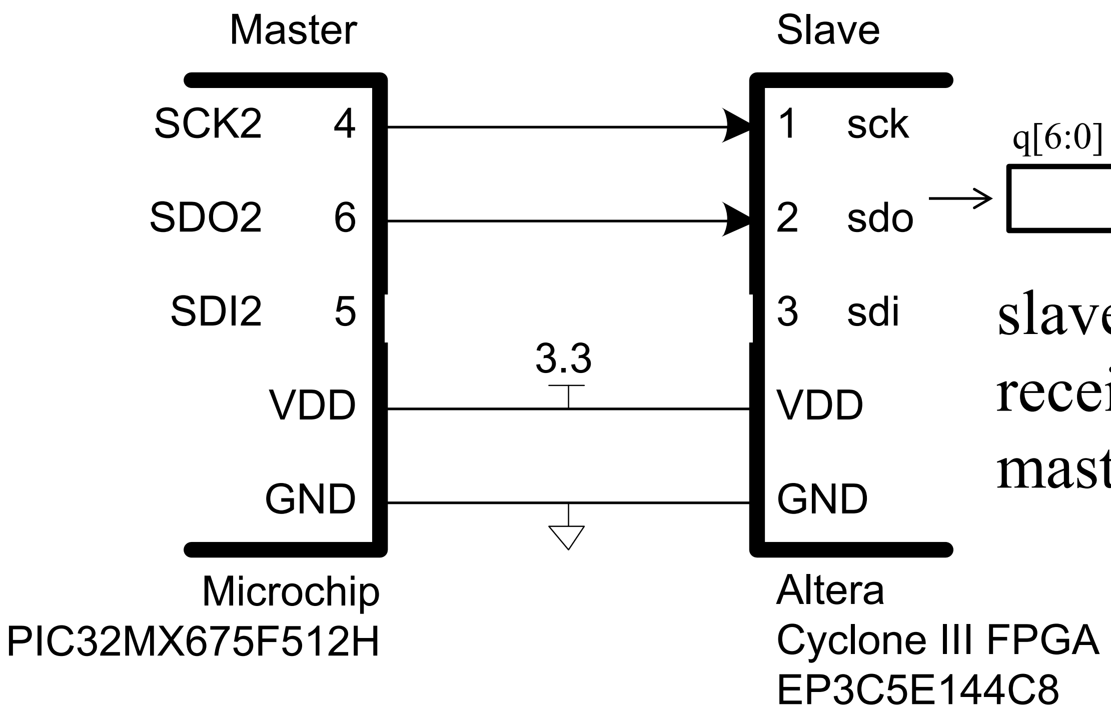
SPI Slave

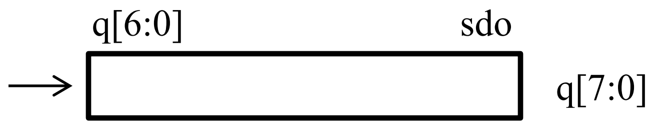
- The FPGA uses a shift register to hold the bits that have been received from the master and the bits that remain to be sent to the master.
- On the first rising sck edge after reset and each 8 cycles thereafter, a new byte from d is loaded into the shift register.
- On each subsequent cycle, a bit is shifted in from sdo and a bit is shifted out to sdi.
- sdi is delayed until the falling edge of sck so that it can be sampled by the master on the next rising edge. After 8 cycles, the byte received can be found in q.

SPI Slave

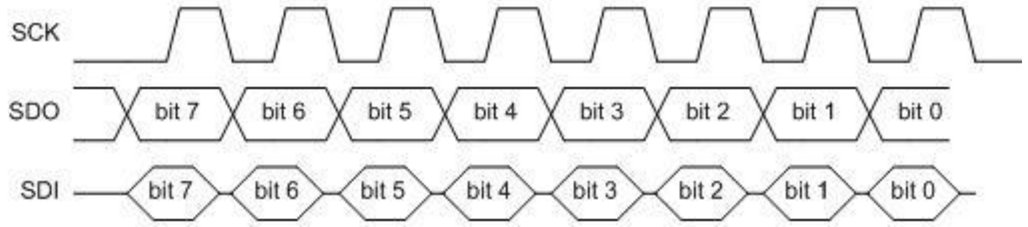
```
module spi_slave(input  logic       sck,   // from master
                 input  logic       sdo,   // from master
                 output logic       sdi,   // to master
                 input  logic       reset, // system reset
                 input  logic [7:0] d,     // data to send
                 output logic [7:0] q);    // data received

  logic [2:0] cnt;
  logic       qdelayed;

  // 3-bit counter tracks when full byte is transmitted and new d should be sent
  always_ff @(negedge sck, posedge reset)
    if (reset) cnt = 0;
    else       cnt = cnt + 3'b1;

  // loadable shift register
  // loads d at the start, shifts sdo into bottom position on subsequent step
  always_ff @(posedge sck)
    q <= (cnt == 0) ? d : {q[6:0], sdo};

  // align sdi to falling edge of sck
  // load d at the start
  always_ff @(negedge sck)
    qdelayed = q[7];
  assign sdi = (cnt == 0) ? d[7] : qdelayed;
endmodule
```

SPI





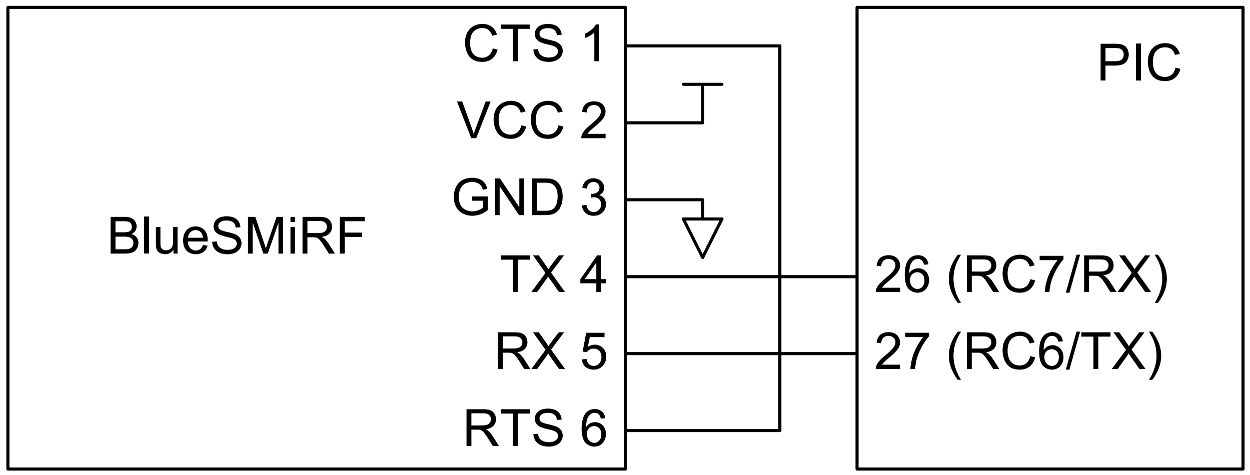
slave only need to received data from the master



SPI Slave

```
// If the slave only need to received data from the master
// Slave reduces to a simple shift register given by following HDL:
module spi_slave_receive_only(input logic sck,   //from master
                              input logic sdi,  //from master
                              output logic[7:0] q);  data received

always_ff @(posedge sck)
    q <={q[6:0], sdi};    //shift register

endmodule
```

UART



Configure UART

Using UART3 since nothing else uses PORTF

UART

```
/*The main function demonstrates printing to the console and reading from the console using the putstrserial()
and getstrserial() functions.  It also demonstrates using printf(), from stdio.h, which automatically prints through
UART3.*/
#include <P32xxxx.h>
#include <stdio.h>

void initUART(void)
{
        // Configure UART
        // Using UART3 since nothing else uses PORTF

        TRISFbits.TRISF5 = 0; // RF5 is UART3 TX (output)
        TRISFbits.TRISF4 = 1; // RF4 is UART3 RX (input)

        // Want rate of 115.2 Kbaud
        // Assuming PIC peripheral clock Fpb = Fosc / 2 = 20 MHz
        //  based on default instructions in lab 1.
        // U3BRG = (Fpb / 4*baud rate) - 1
        // -> U3BRG = 10 (decimal)
        // Actual baud rate 113636.4 (-1.2% error)
        U3ABRG = 10;
```

UART

```
// UART3 Mode Register
          // bit 31-16: unused
          // bit 15:    ON = 1:     enable UART
          // bit 14:    FRZ = 0: don't care when CPU in normal state
          // bit 13:    SIDL = 0: don't care when CPU in normal state
          // bit 12:    IREN = 0: disable IrDA
          // bit 11:    RTSMD = 0: don't care if not using flow control
          // bit 10:    unused
          // bit 9-8: UEN = 00: enable U1TX and U1RX, disable U1CTSb and U1RTSb
          // bit 7:     WAKE = 0: do not wake on UART if in sleep mode
          // bit 6:     LPBACK = 0: disable loopback mode
          // bit 5:     ABAUD = 0: don't auto detect baud rate
          // bit 4:     RXINV = 0: U1RX idle state is high
          // bit 3:     BRGH = 0: standard speed mode
          // bit 2-1: PDSEL = 00: 8-bit data, no parity
          // bit 0:     STSEL = 0: 1 stop bit
          U3AMODE = 0x8000;
```

UART

```
// UART3 Status and control register
            // bit 31-25: unused
            // bit 24-16: write 0 when not using auto address detect
            // bit 15-14: UTXISEL = 00: interrupt when TX buffer not full
            // bit 13:      UTXINV = 0: U1TX idle state is high
            // bit 12:      URXEN = 1: enable receiver
            // bit 11:      UTXBRK = 0: disable break transmission
            // bit 10:      UTXEN = 1: enable transmitter
            // bit 9:       UTXBF: don't care (read-only)
            // bit 8:       TRMT: don't care (read-only)
            // bit 7-6:     URXISEL = 00: interrupt when receive buffer not empty
            // bit 5:       ADDEN = 0: disable address detect
            // bit 4:       RIDLE: don't care (read-only)
            // bit 3:       PERR: don't care (read-only)
            // bit 2:       FERR: don't care (read-only)
            // bit 1:       OERR = 0: reset receive buffer overflow flag
            // bit 0:       URXDA: don't care (read-only)
            U3ASTA = 0x1400;
}
```

UART (RX)

RX Port

```
char getcharserial(void) {
 while (!U3ASTAbits.URXDA);      // wait until data available
 return U3ARXREG;                // return character received from serial port
}
```

```
void getstrserial(char *str) {
 int i = 0;
 do {                          // read an entire string until detecting
   str[i] = getcharserial();    // carriage return
 } while (str[i++] != '\r');    // look for carraige return
 str[i-1] = 0;                  // null-terminate the string
}
```

UART (TX)

```
 void putcharserial(char c) {
 while (U3ASTAbits.UTXBF);       // wait until transmit buffer empty
 U3ATXREG = c;                   // transmit character over serial port
}

void putstrserial(char *str) {
 int i = 0;
putcharserial('\n');
 putcharserial('\r');
 while (str[i] != 0) {          // iterate over string
   putcharserial(str[i++]);     // send each character
 }
}
void main(void) {
 char str[80];

 inituart();
 while(1) {
   putstrserial("Please type something: ");
   getstrserial(str);
   printf("\n\rYou typed: %s\n\r", str);
 }
}
```