Experiment 8: Interfacing ICs

Objectives:  
To learn about popular interface standards for microcontroller peripherals.

Reading:  
M68HC12B Family Advance Information.  
Microchip 24AA01/24LC01B 1K I2C Serial EEPROM Datasheet.

Introduction:  
Several standards exist for interfacing peripheral ICs to microprocessors and microcontrollers. In this experiment, you will interface the M68EVB912B32 evaluation board to ICs using the SPI protocol and the I\(^2\)C protocol. SPI developed by Motorola is directly supported in hardware on the M68HC12B processors and the I\(^2\)C standard (developed by Intel) will have to be done in software. Either protocol can be carried out in hardware or software and there are some microcontrollers that include hardware support for both protocols and there are some microcontrollers that include neither in hardware. In any case, the advantage of doing the protocols in hardware is improved performance. If there is dedicated hardware the microprocessor can hand the data off to be sent to the peripheral chip, then the processor can do other work and then check on the hardware via polling or having it interrupt when it is done. The protocols we will study in this lab are quite simple to code in software, to do so we will write routines that use standard I/O pins to generate the signals necessary for the various control and data signals called for in the standards. The details of the signaling required for a given chip are always included in the datasheet for the IC. For this lab, I have included the important parts of the appropriate datasheets for the chips you will use.

To give us some interesting data to manipulate in this lab, we will again use the ATD subsystem to read in analog values on channel 7 of the ATD (reuse the code from the Digital Multi-meter lab). In this lab we will read the values from the ATD as fast as possible and then send them to an I\(^2\)C digital potentiometer and an SPI EEPROM.

Experiment:  
Part 1: Write a program that will repeatedly read an analog value from channel 7 of the ATD and will then program the Microchip MCP41XXX SPI digital potentiometer with that value. If we connect the end terminals to +5 and ground, then the wiper arm of the potentiometer will be a value between 0 and 5 volts. In this way, the potentiometer will act as a D to A. This circuit has some flaws because if we loaded the center pin and drew current from the potentiometer, the voltage would change, but we will only sample the voltage with an oscilloscope that has a very high impedance. If you wanted to use the
device as a true D to A you could put an op-amp with unity gain to drive real loads, however, the better solution is to use a dedicated D to A chip like the National DAC0800 family of parts. In addition, some microcontrollers now include D to A modules on-chip. Since the potentiometer is SPI, you will use the SPI subsystem of the HC12. You need to initialize this subsystem to reasonable values and connect the appropriate signals from the HC12 to the MCP41XXX part. Here is C Pseudo code for your program:

```c
main()
{
    unsigned char sample=0;

    init_atd();
    init_spi();

    while(1){
        PortP = PortP&1fh;

        PortP_bit5 = 1;
        sample = read_atd_ch7();
        PortP_bit6 = 1;
        set_potentiometer(sample);
        PortP_bit7 = 1;

    } /* while */
} /* main */
```

Be sure to check that the SPI hardware has sent the previous byte to the potentiometer before you try to send the next byte. Evaluate the performance of your code. How long does it take to read the ATD versus the time it takes to write the SPI. Put the SPI signals on the logic analyzer along with the PortP signals you use for timing. The addition of SPI hardware allows you to go back to the ATD to initiate the next sample while the SPI hardware is still transmitting.

**Part 2:** Now we will modify the above program such that between reading the value from the ATD and writing it to the potentiometer, we will also store it in memory. Storing it in the HC12 RAM would incur a few additional instructions, however, storing it in the Microchip 24LC01B I²C EEPROM will take quite a few more instructions because we can not just hand the transmission of the byte of data off to a hardware unit, instead, we will have to signal the data out a bit at a time to write the EEPROM. Modify the above program to work as follows:

```c
main()
{
    unsigned char i=0,sample=0;

    while(1){

PortP = PortP&0fh;

PortP_bit5 = 1;
sample = read_atd();
PortP_bit6 = 1;
store_in_i2cmemory(i++,sample);
PortP_bit7 = 1;
} /* while */
} /* main */

Use PortP pin 0 for the SCL and PortP pin 1 for SDA.

Here is C pseudo code for the functions to help implement the I²C protocol:

```c
void i2c_start (void)
{
    SDATA = HIGH;      // Set data line high
    delay();
    SCLK = HIGH;      // Set clock line high
    delay();
    SDATA = LOW;      // Set data line low (START SIGNAL)
    delay();
    SCLK = LOW;          // Set clock line low
    delay();
}

void i2c_stop (void)
{
    SCLK = LOW;          // Set clock line low
    Delay();
    SDATA = LOW;      // Set data line low
    delay();
    SCLK = HIGH;      // Set clock line high
    delay();
    SDATA = HIGH;      // Set data line high (STOP SIGNAL)
    delay();
}

void i2c_write (unsigned char data)
{
    unsigned char index;

    for(index = 0; index < 8; index++)  // Send 8 bits to the I2C Bus
    {
        // Output the data bit to the I2C Bus
        SDATA = ((data & 0x80) ? 1 : 0);
        data  <<= 1;         // Shift the byte by one bit
        delay();
        SCLK = HIGH;      // Clock the data into the I2C Bus
        delay();
        SCLK = LOW;
        delay();
    }
```
index = SDATA; //??
SCLK = HIGH; // Clock the ACK from the I2C Bus
delay();
SCLK = LOW;
delay();
}

unsigned char i2c_read (void)
{
    unsigned char index, input_data;

    input_data = 0x00;
    for(index = 0; index < 8; index++) // Send 8 bits to the I2C Bus
    {
        input_data <<= 1;         // Shift the byte by one bit
        SCLK = HIGH;          // Clock the data into the I2C Bus
        delay();
        input_data |= SDATA;    // Input the data from the I2C Bus
        delay();
        SCLK = LOW;
        delay();
    }
    return input_data;
}

Instructor Signature (Part 1):______________________________

Instructor Signature (Part 2):______________________________