Department of Computer Engineering
Digital Hardware Systems
CpE 3201 - Embedded Systems

Practical Activity #7
Inter-Integrated Circuit Communication (I^2C)

Exercise Objectives:
1. Configure I^2C mode of the Master Serial Synchronous Port Module (MSSP) of the PIC16F877A
2. Use I^2C master/slave transmit and receive
3. Applying I^2C in interfacing I/O device to the MCU

Student Outcomes:
At the end of the exercise, students must be able to:
1. learn how to configure the I^2C mode of the MSSP
2. use I^2C to transmit and receive data to/from slave devices
3. develop an embedded systems application utilizing I^2C

Tools Required:
The following will be provided for the students:
1. MPLAB IDE v8.92 + MPLAB XC8 v1.33
2. Proteus v8.11

Delivery Process:
The instructor will conduct a briefing for this practical activity. Notes from the lecture class will provided for reference purposes. All inquiries pertaining to the latter must be referred to this manual.

Note:
Images and other contents in this manual are copyright protected. Use of these without consent from the respective authors is unauthorized.

Part I. MSSP Module of PIC16F877A

The Master Synchronous Serial Port (MSSP) module is a serial interface, useful for communicating with other peripheral or microcontroller devices such as serial EEPROMs, shift registers, display drivers, A/D converters, etc. The MSSP module can operate in one of two modes:

- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I^2C)

I^2C Mode

The MSSP module in I^2C mode fully implements all master and slave functions (including general call support) and provides interrupts on Start and Stop bits in hardware to determine a free bus (multi-master function). It also implements the standard mode specifications, as well as 7-bit and 10-bit addressing. Two pins are used for data transfer:

- Serial clock(SCL)–**RC3**/SCK/SCL
- Serial data(SDA)–**RC4**/SDI/SDA

These pins must be configured as inputs or outputs through the TRISC<4:3> bits. Registers associated with the I^2C mode are:

- MSSP Control Register (SSPCON)*
- MSSP Control Register 2 (SSPCON2)
- MSSP StatusRegister (SSPSTAT)
- Serial Receive/Transmit Buffer Register (SSPBUF)
- MSSP Shift Register (SSPSR)**

- MSSP Address Register (SSPADD)

* in the PIC16F87X data sheet, it is referred as SSPCON1 (Register 9-4, page 82)
** not directly accessible

SSPCON, SSPCON2 and SSPSTAT are the control and status registers in I^{2}C mode operation. The SSPCON and SSPCON2 registers are readable and writable. The lower six bits of the SSPSTAT are read-only. The upper two bits of the SSPSTAT are read/write.

SSPSR is the shift register used for shifting data in or out. SSPBUF is the buffer register to which data bytes are written to or read from.

SSPADD register holds the slave device address when the SSP is configured in I^{2}C Slave mode. When the SSP is configured in Master mode, the lower seven bits of SSPADD act as the baud rate generator reload value.

Part II. Master Mode

Master mode is enabled by setting and clearing the appropriate SSPM bits in SSPCON and by setting the SSPEN bit. In Master mode, the SCL and SDA lines are manipulated by the MSSP hardware.

Master mode of operation is supported by interrupt generation on the detection of the Start and Stop conditions. The Stop (P) and Start (S) bits are cleared from a Reset or when the MSSP module is disabled. Control of the I2C bus may be taken when the P bit is set or the bus is Idle, with both the S and P bits clear.

Initialization (Master Mode)

- Set RC3 (SCL) and RC4 (SDA) pins to input.
- Enable synchronous serial port via the SSPEN bit in SSPCON1.
- Configure MCU to operate in Master Mode (SSPM3:SSPM0) in SSPCON1.
- Set start and stop condition to idle (SEN, PEN) in SSPCON2.
- Set receive and acknowledge enable to idle (RCEN, ACKEN) in SSPCON2.
- Configure clock speed by setting SSPADD lower 7-bit to the baud rate generator reload value.

$$SSPADD = \frac{F_{OSC}}{4 * clock} - 1$$

For example, a clock speed of 100 KHz with an F_{OSC} = 4 MHz:

$$SSPADD = \frac{4MHz}{4 * 100KHz} - 1 = 9 \; or \; 0x09$$

```
void init_I2C_Master(void)
{
      TRISC3 = 1;       // set RC3 (SCL) to input
      TRISC4 = 1;       // set RC4 (SDA) to input
      SSPCON = 0x28;    // SSP enabled, I2C master mode
      SSPCON2 = 0x00;   // start condition idle, stop condition idle
                        // receive idle
      SSPSTAT = 0x00;   // slew rate enabled
      SSPADD = 0x09;    // clock frequency at 100 KHz (FOSC = 4MHz)
}
```

I^{2}C Wait

- Before performing a data transmission, make sure no operation in I^{2}C (all operations are complete).
- To do this, check the bit in SSPSTAT if transmission is in progress or that ACKEN, RCEN, PEN, RSEN and SEN in SSPCON2 are set to 0 (to be absolutely sure that no I^{2}C operation is going on).

- Once either conditions is satisfied, an operation (send/receive) can now be started.

```
void I2C_Wait(void)
{
      /* wait until all I2C operation are finished*/
      while((SSPCON2 & 0x1F) || (SSPSTAT & 0x04));
}
```

<u>Start & Stop Condition</u>

- Each time an operation is performed (read or write), a **start condition** must be issued via the SEN bit in SSPCON2.
- After an operation, a **stop condition** must be issued via the PEN bit in SSPCON2.
- Before setting SEN or PEN bits, make sure that no operation is currently on going.

```
void I2C_Start(void)
{
      /* wait until all I2C operation are finished*/
      I2C_Wait();

      /* enable start condition */
      SEN = 1;    // SSPCON2
}

void I2C_Stop(void)
{
      /* wait until all I2C operation are finished*/
      I2C_Wait();

      /* enable stop condition */
      PEN = 1;    // SSPCON2
}
```

<u>Repeated Start</u>

- The master can issue another address + R/W byte without issuing a stop condition by enabling repeated start via the RSEN bit in SSPCON2.
- Before setting RSEN bit, make sure that no operation is currently on going.

```
void I2C_RepeatedStart(void)
{
      /* wait until all I2C operation are finished*/
      I2C_Wait();

      /* enable repeated start */
      RSEN = 1;    // SSPCON2
}
```

<u>I^{2}C Transmit (Master Mode)</u>

- The "7-bit slave address + R/W" byte or data frame to be transmitted to the I^{2}C bus shall be written to the SSPBUF register.
- Before writing to SSPBUF, make sure that no operation is ongoing.
- In between the start and stop condition, the slave address + R/W and 8-bit data frame will be sent (master mode transmit).

```
void I2C_Send(unsigned char data)
{
      /* wait until all I2C operation are finished*/
      I2C_Wait();

      /* write data to buffer and transmit */
      SSPBUF = data;
}
```

Writing to SSPBUF starts the transmission which is done automatically by the MCU.

<u>I²C Transmit (Master Mode)</u>

- The data received from the slave device via I²C bus is stored in the SSPBUF register.
- During I²C communication, the slave will send an acknowledge bit after sending the 8-bit data. The master acknowledge bit should be set via the ACKDT bit in SSPCON2 register.
- When the master received the acknowledgment bit from the slave, it has to enable the “acknowledge sequence” via the ACKEN bit in SSPCON2.

```
unsigned char I2C_Receive(unsigned char ack)
{
      unsigned char temp;

      I2C_Wait();    // wait until all I2C operation are finished
      RCEN = 1;      // enable receive (SSPCON2 reg)

      I2C_Wait();    // wait until all I2C operation are finished
      temp = SSPBUF; // read SSP buffer

      I2C_Wait();    // wait until all I2C operation are finished
      ACKDT = (ack)?0:1;  // set acknowledge (ACK) or not acknowledge (NACK)
      ACKEN = 1;     // enable acknowledge sequence

      return temp;
}
```

R/W bit in SPSSTAT is automatically set to '1' when reading SSPBUF. ACKDT is the "ACK" or "NACK" bit that will be sent when the acknowledge sequence is started. When the master issues an "ACK", the slave will send the next byte. However, if the master issues an "NACK", then the slave will not send the next byte and releases the clock.

<u>I²C Send/Receive Example (Master Mode)</u>

- Process to send I²C packet* to slave:
  - Initialize I²C as master mode.
  - Initiate **start** condition.
  - Send the 7-bit address followed by the 8-bit data frame(s)**.
  - Initiate **stop** condition.
- Process to read data from slave:
  - Initialize I²C as master mode. (not needed if I²C is already initialized)
  - Initiate **start** condition.
  - Send the 7-bit address of the slave.
  - Read data frame(s).
  - Initiate **stop** condition.

* packet includes the start condition, address, data frames & stop condition (R/W bit is set automatically)

In this example, the master will send a data frame from PORTD to a slave device with address 0x10 (the I²C receive example will be included in Part III). I/O connections: PORTD <- DIP switches (8-bit).

```
void main(void)
{
      TRISD = 0xFF;       // set all bits in PORTD to input
      init_I2C_Master();  // initialize I2C as master

      for(;;)
      {
            I2C_Start();      // initiate start condition
            I2C_Send(0x10);   // send the slave address + write
            I2C_Send(PORTD);  // send 8-bit data frame
            I2C_Stop();       // initiate stop condition
            __delay_ms(200); // delay before next operation
      }
}
```

After the start condition, the master will send the address + R/W bit. The 7-bit slave address is the MSB while the R/W bit takes the LSB. Therefore the address + R/W byte is: 1000000**0**. R/W bit is '0' since this a write operation. Then the data frame can be send then a stop condition is issued no more data will be sent.

Open Proteus and construct the required circuit with filename "LE7-1.pdsprj" (do not include a firmware project). Connect an 8-bit DIP switch to PORTD and an I2C DEBUGGER (see Figure 1) to the MCU via the SCL (RC3) and SDA (RC4). The I^{2}C debugger will monitor the packet/message sent or received via the I^{2}C bus. It can display the start condition, slave address, data frames, ACK/NACK and stop condition. Place *pull up resistors* of SDA and SCL with a value of 10K ohms.

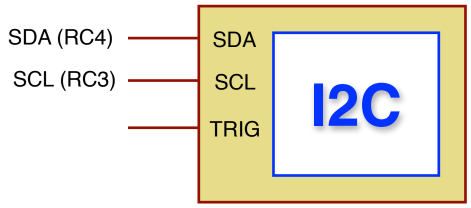
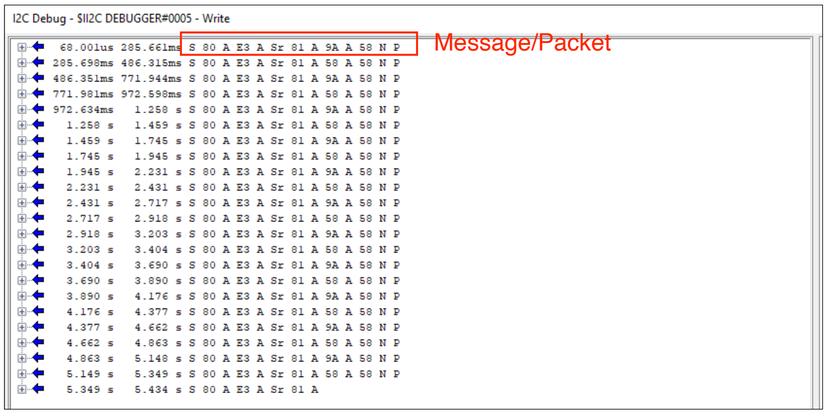


Fig. 1. I2C DEBUGGER in Proteus

When the simulation is started, the I^{2}C debugger window will open and it will display the packet/message sent through the bus in real-time from master to slave and vice versa.



```
 68.001us 285.661ms S 80 A E3 A Sr 81 A 9A A 58 N P
285.698ms 486.315ms S 80 A E3 A Sr 81 A 58 A 58 N P
486.351ms 771.944ms S 80 A E3 A Sr 81 A 9A A 58 N P
771.981ms 972.598ms S 80 A E3 A Sr 81 A 58 A 58 N P
972.634ms   1.258 s S 80 A E3 A Sr 81 A 9A A 58 N P
  1.258 s   1.459 s S 80 A E3 A Sr 81 A 58 A 58 N P
  1.459 s   1.745 s S 80 A E3 A Sr 81 A 9A A 58 N P
  1.745 s   1.945 s S 80 A E3 A Sr 81 A 58 A 58 N P
  1.945 s   2.231 s S 80 A E3 A Sr 81 A 9A A 58 N P
  2.231 s   2.431 s S 80 A E3 A Sr 81 A 58 A 58 N P
  2.431 s   2.717 s S 80 A E3 A Sr 81 A 9A A 58 N P
  2.717 s   2.918 s S 80 A E3 A Sr 81 A 58 A 58 N P
  2.918 s   3.203 s S 80 A E3 A Sr 81 A 9A A 58 N P
  3.203 s   3.404 s S 80 A E3 A Sr 81 A 58 A 58 N P
  3.404 s   3.690 s S 80 A E3 A Sr 81 A 9A A 58 N P
  3.690 s   3.890 s S 80 A E3 A Sr 81 A 58 A 58 N P
  3.890 s   4.176 s S 80 A E3 A Sr 81 A 9A A 58 N P
  4.176 s   4.377 s S 80 A E3 A Sr 81 A 58 A 58 N P
  4.377 s   4.662 s S 80 A E3 A Sr 81 A 9A A 58 N P
  4.662 s   4.863 s S 80 A E3 A Sr 81 A 58 A 58 N P
  4.863 s   5.148 s S 80 A E3 A Sr 81 A 9A A 58 N P
  5.149 s   5.349 s S 80 A E3 A Sr 81 A 58 A 58 N P
  5.349 s   5.434 s S 80 A E3 A Sr 81 A
```

Fig. 2. I2C DEBUGGER Window

The format of the message/packet displayed on the debugger window in Figure 2 is:

```
S XX A XX A Sr XX A XX N P
```

where:			
	`S` - start condition	`XX` - address/data	`Sr` - repeated start condition
	`A` - acknowledge	`N` - not acknowledge	`P` - stop condition

In MPLAB, create a new project using the project wizard with the name "CpE 3201-LE7". Create a new source file with the filename "LE7-1.c" and add it to the project. Write the source code with the pre-processor directives. Write the example program to the source file. Build the program and debug if necessary. After successfully building the source code, simulate the program in Proteus ("LE7-1.pdsprj") by loading the generated .hex file to the microcontroller. Monitor the debugger window and check the data sent or received by the master.

Part III. Slave Mode

In Slave mode, the SCL and SDA pins must be configured as inputs (TRISC<4:3> set). The MSSP module will override the input state with the output data when required (slave-transmitter).

The I2C Slave mode hardware will always generate an interrupt on an address match. Through the mode select bits, the user can also choose to interrupt on Start and Stop bits.

When an address is matched, or the data transfer after an address match is received, the hardware automatically will generate the Acknowledge (ACK) pulse and load the SSPBUF register with the received value currently in the SSPSR register.

Initialization (Slave Mode)

- Set RC3 (SCL) and RC4 (SDA) pins to input.
- Disable slew rate control via SMP bit (SSPSTAT).
- Enable synchronous serial port via SSPEN (SSPCON).
- Set SCK bit to "release clock" (SSPCON1).
- Configure MCU to operate in Slave Mode, 7-bit address via SSPM3:SSPM0 bits (SSPCON).
- Enable clock stretching for slave transmit & receive via SEN bit (SSPCON2).
- Set a valid* 7-bit slave address via the SSPADD register.
- Enable synchronous serial port interrupt via SSPIE bit (PIE1).
- Clear interrupt flag via SSPIF bit (PIR1).
- Enable peripheral interrupts via PEIE and GIE bits (INTCON).

```
void init_I2C_Slave(unsigned char slave_add)
{
      TRISC3 = 1;        // set RC3 (SCL) to input
      TRISC4 = 1;        // set RC4 (SDA) to input
      SSPCON = 0x36;     // SSP enabled, SCK release clock
                         // I2C slave mode 7-bot address
      SSPCON2 = 0x01;    // start condition idle, stop condition idle
                         // receive idle
      SSPSTAT = 0x80;    // slew rate control disabled
      SSPADD = slave_add;  // 7-bit slave address
      SSPIE = 1;         // enable SSP interrupt
      SSPIF = 0;         // clear interrupt flag
      PEIE = 1;          // enable peripheral interrupt
      GIE = 1;           // enable unmasked interrupt
}
```

Receive Interrupts (Slave Mode)

- Hold the clock low via the CKP bit (SSPCON).
- Check for overflow via SSPOV bit or data collision via WCOL bit (SSPCON). If overflow or data collision occurs, no master read/write operation can happen.
- Check if master operation is "read" via the R/$\overline{\text{W}}$ bit and the last received data frame matches the address via the D/$\overline{\text{A}}$ bit (SSPSTAT).
- Check if master operation is "write" the R/$\overline{\text{W}}$ bit and the last received data frame matches the address via the D/$\overline{\text{A}}$ bit (SSPSTAT).
- Clear the SSP interrupt flag (SSPIF).

```
void interrupt ISR(void)
{
      unsigned char temp;

      CKP = 0;           // hold clock low (SSPCON reg)

      if(WCOL || SSPOV) // check if overflow or data collision (SSPCON reg)
      {
            temp = SSPBUF;  // read SSPBUF to clear buffer
            WCOL = 0;       // clear data collision flag
            SSPOV = 0;      // clear overflow flag
            CKP = 1;        // release clock (SSPCON reg)
      }

      /* check operation if "write" or "read"*/
```

```
	if(!SSPSTATbits.D_nA && !SSPSTATbits.R_nW)      // write to slave
	{
		temp = SSPBUF; // read SSPBUF to clear buffer
		while(!BF);    // wait until receive is complete (SSPSTAT reg)
		/* read data from SSPBUF */
		/* data = SSPBUF; */
		CKP = 1;         // release clock (SSPCON reg)
	}
	else if(!SSPSTATbits.D_nA && SSPSTATbits.R_nW)  // read from slave
	{
		temp = SSPBUF;  // read SSPBUF to clear buffer
		BF = 0;         // clear buffer status bit (SSPSTAT reg)
		/* send data by writing to SSPBUF */
		/* SSPBUF = data; */
		CKP = 1;      // release clock (SSPCON reg)
		while(BF);    // wait until transmit is complete (SSPSTAT reg)
	}

		SSPIF = 0;              // clear interrupt flag
}
```

The SSP module compares the received data frame to the slave address after a start or repeated restart condition. When a match occurs, $D/\overline{A}$ bit will be set to '0'. If the LSB of the address + R/W byte is '0', the master device operation to the slave is "write" or "send" otherwise it is "read" or "receive".

<u>I²C Send/Receive Example (Master & Slave Mode)</u>

For this example, two (2) MCUs will be used configured as master and slave respectively. The slave device has an address of 0x10. When the master "writes" data to the slave, data from PORTD the master will be sent. The received data will be written to PORTB of the slave. When the master "reads" data from the slave, data from PORTD of the slave will be sent to the master. The received data from the slave will be written to PORTB of the master. No foreground routine in the slave device since all operations are interrupt based. I/O connection:

MCU1 (master): PORTB -> LED bar graph, PORTD <- DIP switches (8-bit).
MCU2 (slave):   PORTB -> LED bar graph, PORTD <- DIP switches (8-bit).

```
/* Slave Device*/
void main(void)
{
	TRISB = 0x00;   // set all bits in PORTB to output
	PORTB = 0x00;   // all LEDs in PORTB are off
	TRISD = 0xFF;   // set all bits in PORTD to input

	init_I2C_Slave(0x10);  // initialize I2C as slave with address 0x01

	for(;;)
	{
	}
}

void interrupt ISR(void)
{
	unsigned char temp;

	CKP = 0;          // hold clock low (SSPCON reg)

	if(WCOL || SSPOV) // check if overflow or data collision (SSPCON reg)
	{
		temp = SSPBUF;  // read SSPBUF to clear buffer
		WCOL = 0;       // clear data collision flag
		SSPOV = 0;      // clear overflow flag
		CKP = 1;        // release clock (SSPCON reg)
	}
```

```
      /* check operation if “write” or "read"*/
      if(!SSPSTATbits.D_nA && !SSPSTATbits.R_nW)      // write to slave
      {
            temp = SSPBUF; // read SSPBUF to clear buffer
            while(!BF);    // wait until receive is complete (SSPSTAT reg)
            PORTB = SSPBUF;   // write data from master to PORTB
            CKP = 1;          // release clock (SSPCON reg)
      }
      else if(!SSPSTATbits.D_nA && SSPSTATbits.R_nW)  // read from slave
      {
            temp = SSPBUF;    // read SSPBUF to clear buffer
            BF = 0;           // clear buffer status bit (SSPSTAT reg)
            SSPBUF = PORTD;   // send data from PORTD to master
            CKP = 1;          // release clock (SSPCON reg)
            while(BF);    // wait until transmit is complete (SSPSTAT reg)
      }

      SSPIF = 0;              // clear interrupt flag
}
```

The master device code is the same as the master mode transmit including all the functions. In this example, the master will send a packet/message to the slave then also read or receive data.

```
/* Master Device*/
void main(void)
{
      TRISB = 0x00;       // set all bits in PORTB to output
      PORTB = 0x00;       // all LEDs in PORTB are off
      TRISD = 0xFF;       // set all bits in PORTD to input

      init_I2C_Master();  // initialize I2C as master

      for(;;)
      {
            I2C_Start();     // initiate start condition
            I2C_Send(0x10);  // send the slave address + write
            I2C_Send(PORTD); // send 8-bit data frame
            I2C_Stop();      // initiate stop condition

            __delay_ms(200); // delay before next operation

            I2C_Start();     // initiate repeated start condition
            I2C_Send(0x11);  // send the slave address + read
            PORTB = I2C_Receive(0); // read data and not acknowledge (NACK)
                                    // end of read operation
                                    // write received data to PORTB
            I2C_Stop();      // initiate stop condition

            __delay_ms(200); // delay before next operation
      }
}
```

The code above shows the master device will send a data on the first packet/message and read or receive data on the second. Note that the slave address in the second packet is 0x11 where the slave address is still 0x10 but the LSB is set to '1'. In this case the bit is ‘1’ since the operation is a “read”.

Open Proteus and construct the required circuit with filename “LE7-2.pdsprj” (do not include a firmware project). Connect the SCL (RC3) and SDA (RC4) of MCU1 to SCL (RC3) and SDA (RC4) of MCU2 (see Figure 3). Place pull up resistors of SDA and SCL with a value of 10K ohms.

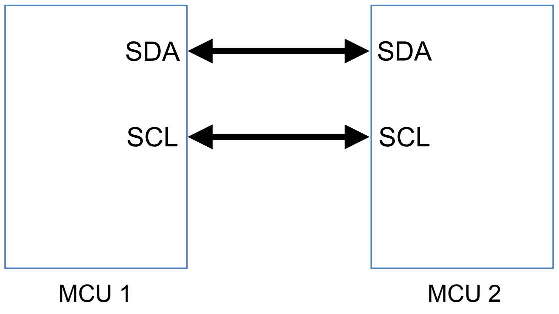


Fig. 3. I^{2}C MCU to MCU connection

(MCU 1, Master)

In MPLAB, create a new project using the project wizard with the name “CpE 3201-LE7-M”. Create a new source file with the filename “LE7-2-M.c” and add it to the project. Write the source code with the pre-processor directives. Write the example program (master) to the source file. Build the program and debug if necessary.

(MCU 2, Slave)

In MPLAB, create a new project using the project wizard with the name “CpE 3201-LE7-S”. Create a new source file with the filename “LE7-2-M.c” and add it to the project. Write the source code with the pre-processor directives. Write the example program (master) to the source file. Build the program and debug if necessary.

Build the programs and debug if necessary. After successfully building, simulate the program in Proteus (“LE7-2.pdsprj”) by loading the generated .hex files to MCU1 (“CpE 3201-LE7-M.hex”) and MCU2 (“CpE 3201-LE7-S.hex”). Set the DIP switches of the master device and observe the ouput on the slave. The data from the master should be reflected on the slave output. Similarly, do the same with the slave device.

Part IV. Hands-on Exercise

Create a new project file “LE7-3.pdsprj” in Proteus (do not include a firmware project). Connect an SHT21 (Humidity and Temperature Sensor IC) to the MCU via the SDA and SCL. Connect an LCD to the MCU (you can select the port). Place *pull up resistors* to SDA and SCL lines with a value of 10K ohms. Add also an I2C DEBUGGER and connect to SDA and SCL lines.

In MPLAB, create a new source file with the filename “LE7-3.c” and add it to the project “CpE 3201-LE7”. Write a program that will read the temperature and relative humidity data and display it on the LCD in real-time. The display should be similar to figure below:

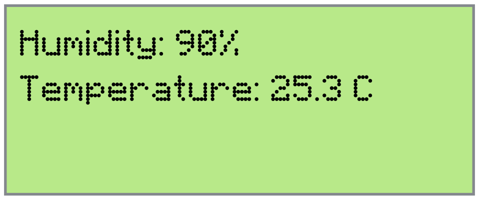


Fig. 4. Humidity and Temperature display on LCD

Study the data sheet of the SHT21 sensor and go to page 7 for the “Sending a Command”. Table 6 (page) shows the basic command to control the sensor. In this exercise, use the “hold master” command for reading the humidity and temperature. Hold master means that the SHT21 will hold down the SCL line (clock stretching) while measuring either humidity or temperature. The sensor has a 14-bit digital data with a slave address of 0x80 sending command and 0x81 for reading data (note the LSB denote the operation either read or write). To read the the relativity humidity, refer to the communication sequence below:

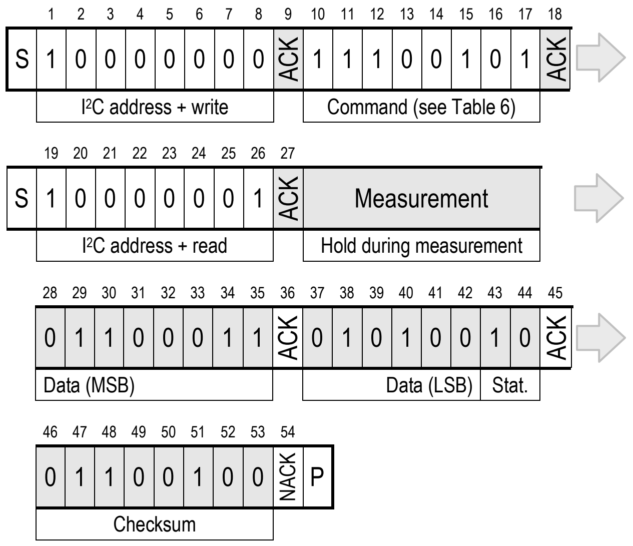


Fig. 5. Hold master communication sequence.

The command for reading humidity (hold master) is 11100101_2 or 0xE5. To read the data without releasing the I²C bus, a repeated restart condition will be issued by the master (MCU) followed by the address + read. Since the slave will hold down the SCL line, there will be no activity until the measurement is done. Once the measurement and conversion is complete, the slave (SHT21) will send the next three bytes which contains the data (2 bytes) and checksum. Refer to Figure 5 for the data MSB and LSB. Note that the 2-bit LSB of the second byte are status bits. You can skip the checksum by system by issuing an “NACK” after reading the second data byte. Once the 14-bit data is extracted, you can convert it into relative humidity (%) using the formula:

$$RH = -6 + 125x\frac{S_{RH}}{2^{RES}}$$

S_{RH} is the 14-bit digital data for relative humidity and RES is the resolution which is 14-bits.

For reading the temperature data, use the same communication sequence in Figure 5 but the command should be 11100011_2 or 0xE3 (hold master). To convert the 14-bit data read, use the formula below:

$$T = -46.85 + 175.72x\frac{S_T}{2^{RES}}$$

Build the program and debug if necessary. After successfully building the source code, simulate the program in Proteus (“LE7-3.pdsprj”) by loading the generated .hex file (“CpE 3201-LE7.hex”) Press a key on the keypad in MCU1 and verify the display in MCU2. Adjust the relative humidity or temperature using the up/down arrow of the virtual SHT21 sensor. You can also toggle between relative humidity and temperature. Verify the display on the LCD and the data should match with the SHT21.

Instructions for submission:

- Submit the following:
  - LE7-3.pdsprj
  - LE7-3.c
- Submit the files in Canvas in the correct order and file format.

Assessment

Criteria	Outstanding (4 pts)	Competent (3 pts)	Marginal (2 pts)	Not Acceptable (1 pt)	None (0 pts)
Feature Configuration	Configuration was properly done.	Configuration has minor errors.	-	Configuration is incorrect.	No project created.
Functionality	The systems function perfectly.	The system functions with minor issues.	The system has several issues which already affect the functionality.	The presented system is non-functioning.	No project created.
Firmware	Firmware created successfully without issues and followed the requirements.	Firmware created successfully with some issues and followed the requirements.	Firmware has issues and missing some of the requirements.	Firmware has several issues and did not follow the requirements.	No project created.

Copyright Information

References

- PIC16F87X Data Sheet, Microchip Technology Inc. 2003.
- SHT21 Data Sheet, Sensirion AG, 2011.
- https://i2c.info/
- https://electrosome.com/i2c-pic-microcontroller-mplab-xc8/
- https://circuitdigest.com/microcontroller-projects/i2c-communication-with-pic-microcontroller-pic16f877a
- https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html#
- CpE 3201 Lecture Notes on I^{2}C in PIC16F877A.

Change Log:

Date	Version	Author	Changes
March 31, 2021	1.0	Van B. Patiluna	- Initial draft.