**I²C Bus**

- The I²C bus is a two-wire bus (plus ground) where the two wires are called:
  - SCL – Clock line
  - SDA – Data line
  - Gnd – Ground line
- This is a synchronous bus. SCL is the synchronizing signal.
- SCL and SDA are open drain – they can be driven low but they must have a passive pull up resistor.
- Multiple modules make use of the same lines. Only one set of pull up resistors is needed per system.

![I²C Bus Diagram]

- Typically a system has one master module and multiple slaves. The master module is the one that drives the clock line.
- If the clock line is high and the data line goes high to low, a start sequence is initiated.
- If the clock line is high and the data line goes low to high, a stop sequence is initiated.

![Start and Stop Sequences]

- Data is transferred in sequences of 8 bits. The bits are placed on the SDA line starting with the MSB (Most Significant Bit). The SCL line is then pulsed high, then low.
- For every 8 bits transferred, the device receiving the data sends back an acknowledge bit, so there are actually 9 SCL clock pulses to transfer each 8 bit byte of data.
- If the receiving device sends back a low ACK bit, then it has received the data and is ready to accept another byte. If it sends back a high then it is indicating it cannot accept any further data and the master should terminate the transfer by sending a stop sequence.

<table>
<thead>
<tr>
<th>SDA</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCL</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

- The standard bit rate for the clock is 100 KHz. Philips defines faster rates up to a Mbit but most I²C systems run at 100 KHz.
- We will use only a 7-bit address. Philips also defines a 10-bit address. The 7-bit address gives us up to 128 devices.
The 8th bit tells whether we are reading or writing to the address. One means reading from the slave and zero means writing to the slave. Note that the 7-bit address is in the upper 7 bits.

To write data to a slave device:
1. Send a start sequence
2. Send the I2C address of the slave with the R/W bit low (even address)
3. Send the internal register number you want to write to. So if a slave module has 16 internal registers you could send out the number 2 to write to register 2.
4. Send the data byte
5. [Optionally, send any further data bytes]
6. Send the stop sequence.

To read data from a slave device:
1. Send a start sequence
2. Send the I2C address of the slave with the R/W bit low (even address)
3. Send the internal address of the register you want to read.
4. Send a start sequence again (repeated start)
2. Send the I2C address of the slave with the R/W bit high (odd address)
6. Read data byte.
7. Send the stop sequence.

Note that if the slave cannot get data back in time for the clock signal which the master controls, it can hold the clock line low until the data is ready. The master will recognize this and wait for the clock line to go high to get the data.
I²C with the AT89C51CC03 and MPL3115A2 Sensor

The AT89C51CC03 does not have a built in I²C bus so communication to an I²C device must be done by "bit-banging". This means we must set up to output pins, say P1.0 and P1.1, to be the SDA and SCL bus and write a program to output ones and zeros in the right order to send and receive data. To do the bit-banging we will need to write functions which do the following:

1) StartConnection – Initializes a connection to the sensor.
   
   **Return value:** void
   
   **Parameters:** unsigned char slaveAddr
   
   - Set SCL to 1
   - Delay
   - Set SDA to 0
   - Delay
   - Use SendData to send slaveAddr

2) StopConnection – Terminates a connection to the sensor.
   
   **Return value:** void
   
   **Parameters:** void
   
   - Set SDA to 0
   - Set SCL to 1
   - Delay
   - Set SDA to 1
   - Delay

3) SendData – Sends one byte of data to the sensor.
   
   **Return value:** void
   
   **Parameters:** unsigned char dataI2C
   
   - Loop 8 times
     
     {Set SCL to 0
      Delay
      Set SDA to MSB of dataI2C
      Shift dataI2C one place left
      Delay
      Set SCL to 1
      Delay
      Set SCL to 0
      }
   
   - Set SDA to 1
   - Delay
   - Set SCL to 0
   - Delay

4) ReceiveData – Receives one byte of data from the sensor.
   
   **Return value:** unsigned char data
   
   **Parameters:** bit lastByte
   
   - Loop 8 times (i = 0 to7)
     
     {Set SDA to 1
      Set SCL to 1
      Delay
      Set LSB of data to SDA
      }
If (i < 7)
    Shift data left one time
    Delay
    Set SCL to 0
    Delay

} IF lastByte is a 1
    Set SDA to 1
Otherwise
    Set SDA to 0
Set SCL to 1
Delay
Set SCL to 0
Delay
Set SDA to 1
Return data

5) Delay – a time delay used to set the communications rate for the bus

Return value: void
Parameters: void
Does a software delay loop for about 900 μsec.

The main program which uses these five functions to determine the altitude and temperature needs the following constants:

SLAVEREAD = 0xC1;  //Address to read from sensor
SLAVEWRIT = 0xC0;  //Address to write to sensor
CNTLREG1 = 0x26;  //Control register address
PTDATACFG = 0x13;  //Event flag register address
STATUS = 0;        //Status register address
OUTPMSB = 0x01;    //Most significant byte of pressure data
OUTPCSB = 0x02;    //Middle byte of pressure data
OUTPLSB = 0x03;    //Least significant byte of pressure data
OUTTMSB = 0x04;    //Most significant byte of temperature data
OUTTLSB = 0x05;    //Least significant byte of temperature data

The MPL3115A2 sensor has three modes of operation: Standby, Active altimeter, and Active barometer. The mode is determined by bits in the CNTLREG1. This register has the following bit definitions:

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALT</td>
<td>RAW</td>
<td>OS2</td>
<td>OS1</td>
<td>OS0</td>
<td>0</td>
<td>RST</td>
<td>OST</td>
</tr>
</tbody>
</table>

Reset

where

ALT – 1 for altimeter mode and 0 for barometer mode
RAW – 1 for raw data mode, 0 for processed data mode
OS2, OS1, OS0 – a three-bit number for the oversample ratio. $2^{OS}$ is the amount of oversampling. For example, if these three bits are 101 = 5 there is $2^5 = \text{factor 32}$ oversampling.

RST = 1 means software reset enabled and 0 means software reset disabled.

OST – 1 means initiate measurement immediately

SBYB – 1 is the active mode and 0 means standby mode.

To determine when altitude and temperature data is ready for reading we look at the status register. This register has the following bit definitions:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>PTOW, POW, TOW (pressure and temperature overwrite bits)</td>
</tr>
<tr>
<td>6</td>
<td>PTDR (Pressure/Temperature Data Ready)</td>
</tr>
<tr>
<td>5</td>
<td>PDR, TDR (pressure and temperature data ready bits)</td>
</tr>
</tbody>
</table>

Where:
PTOW, POW, and TOW are pressure and temperature overwrite bits. These bits are set to 1 when the pressure or temperature has been overwritten before it was read.

PTDR – Pressure/Temperature Data Ready. This bit is set to 1 when the pressure and temperature data is ready for reading.

PDR and TDR are the pressure data ready and the temperature data ready bits. They are set to 1 when the pressure or temperature is ready for reading.

The configuration register has three flags which indicate when a temperature or pressure has been read. The bits in the configuration register are defined as:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>DREM (Data Ready Event Mode)</td>
</tr>
<tr>
<td>6</td>
<td>PDEFE (Pressure Data Event Flag Enable)</td>
</tr>
<tr>
<td>5</td>
<td>TEFE (Temperature Data Event Flag Enable)</td>
</tr>
</tbody>
</table>

Where:
DREM – Data Ready Event Mode – if this is 1 it enables the other two flags to be set.

PDEFE – Pressure Data Event Flag Enable – if this is 1 it allows a flag to be set when new pressure data has been set.

TEFE – Temperature Data Event Flag Enable – if this is 1 it allows a flag to be set when new temperature data has been set.

**Write Process**
The process for writing to the chip involves the following steps:
1. Start a connection for writing by using the StartConnection function with the SLAVWRIT address.
2. Use the SendData function to send the address of the register to be written.
3. Use the SendData function to send the data to be written to the address.
4. Use the StopConnection function to end the message.

**Read Process**
The process for reading from the chip involves the following steps:
1. Start a connection for writing by using the StartConnection function with the SLAVWRIT address.
2. Use the SendData function to send the address of the register to be read.
3. Use the StartConnection function to start a connection for reading with the SLAVREAD address.
4. Use the ReadData function to read the data.
5. Use the StopConnection function to end the message.

To read the altitude and temperature data from the sensor you can write an infinite loop which does the following:
1. Send 1010 1000 to Control Register 1 to get altitude data.
2. Send 0000 0111 to the Configuration register.
3. Send 1010 1001 to Control Register 1 to get altitude data.
4. Wait for the conversion to complete
   ```
   Set Sta = Read the status register
   While bit 3 of Sta is 0
     {Set Sta = Read the status register
   }
   5. Read the MSB of altitude data
   6. Read the CSB of altitude data
   7. Read the LSB of altitude data
   8. Read the MSB of temperature data
   9. Read the LSB of temperature data
   ```

The altitude data is received as 20-bit number in three bytes. The first two bytes, MSB and CSB, form a 16-bit number representing the whole number part of the altitude. The third byte has 4-bits in positions 7-4 which represent the fractional part of the altitude. The altitude is in meters.

If the configuration register is set up to produce a pressure reading rather than an altitude reading, the pressure data is in Pascals and is received as a 20-bit number in three bytes. The first 18-bits are in MSB, CSB, and bits 7 and 6 of LSB. This 18-bit number is a whole number in twos complement form. Bits 5 and 4 of LSB are a fractional component of the pressure. This fractional component is not in twos complement form.

The pressure in Pascals can be converted to altitude using the following nonlinear equation:

\[ h = 44330.77\left(1 - \left(\frac{p}{p_0}\right)^{0.1902632}\right) + \text{Offset} \]

Where:
- \( h \) = altitude in meters.
- \( p \) = pressure in Pascals from the 18-bit input number.
- \( p_0 \) = pressure in Pascals at sea level = 101326 Pascals.
- Offset = Error correction due to normal pressure variation.

The temperature data is in degrees Celsius and is a 12-bit number. The MSB is the twos complement whole number representing the temperature. Bits 7 – 4 of LSB is the fractional portion of the temperature is it too is not in twos complement form.

Once you have your pressure in meters (or feet) and the temperature in degrees C or degrees F you need to convert each to the appropriate ASCII code and send it to the LCD display.