Richard W. Wall, University of Idaho, rwall@uidaho.edu

1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

ECE443 Assignment 3: IR Sensor using SMBus Serial Network Communications

Purpose

The purpose of this assignment is to investigate concepts involving a sensor network using SMBus serial communications. The assignment will implement the SMBus protocol using the Microchip PIC32 I2C peripheral to communicate with the Melexis MLX90614 family of infrared temperature transducers. Various methods of error detection are discussed as this assignment also implements eight-bit packet error checking (PEC).

Assignment Deliverables

Using the information provided below, students are to write a device test application that meet the following specifications.

1. The MLX90614 IR sensor is to be connected to I2C port 1 by connecting the I2C device to the chipKit Pro MX7 J7.
2. The device test application only ready the IR sensor and displays selected parameters.
3. During initialization, the LCD display must display “ECE 443” for 1 second.
4. Using Listing 1 as a guide, during initialization, the terminal display must report the following:
    a. UART status
    b. I2C bit rate
    c. Device connection status
    d. The information shown on the first three lines of Listing 3.
        i. Device ID:5A - Configutation:9FB4 - PWM Control:0201
        ii. Emissivity: 65000d - T0max:39315d - T0min:25315d - Ta range:63260d
        iii. Part Number 9406|6fb8|82a4|b0d2
5. During run time, the LCD is to display the ambient temperature (A) and the remote IR temperature as follows:
    a. A:82.13 R:81.81

6. During run time, the PC terminal is to display the sensor data as shown below that is updated once each 2 seconds.
    a. Air: 82.13 T1: 81.81 ID 5A CFG 9FB4
7. The software is to be partitioned as shown in Figure 6. The “program_ir” and “Write_SMBus” functions may be omitted.
8. The CRC byte must be check for each packet received from the IR sensor. If the CRC check bytes do not match, the discrepancy must be reported to the PC terminal.

Minimum Knowledge and Programming Skills

1. Knowledge of C or C++ programming
2. Working knowledge of MPLAB IDE
3. IO pin control
4. Use of logic analyzer or oscilloscope
5. Familiarity with the I^2C and the SMBus serial protocols

Equipment List

1. Digilent chipKit Pro MX7 processor board with USB cable
2. Microchip MPLAB X IDE
3. Microchip MPLAB XC32 Compiler
4. Digilent PmodCLP Parallel Character LCD
5. Logic analyzer (Digilent Analog Discovery)

Software Resources

1. XC32 C/C++ Compiler Users Guide
2. PIC32 Peripheral Libraries for MBLAB C32 or XC32 Compiler
3. PIC32 Family Hardware Reference Manual Section 24 Inter-Integrated Circuit
4. ChipKit Pro MX7 Board Reference Manual
5. LCD library
6. UART library
7. MPLAB ® X Integrated Development Environment (IDE)

References

1. C Programming Reference
2. SMBus protocol specification
3. An I2C Network Protocol for Environmental Monitoring
4. PEC protocol specification
5. AN1148 Cyclic Redundancy Check (CRC)
6. Embedded System Academy – I2C FAQ

The Melexis MLX90614 IR Temperature Sensor

The MLX90614 is an Infra Red thermometer for non contact temperature measurements. Both the IR sensitive thermopile detector chip and the signal conditioning ASIC are integrated in the same TO-39 Package as shown in Figure 1.

Figure 1. The MLX90614 IR Thermometer

SMBus Protocol

The specification for the Systems Management Bus (SMBus) protocol was first defined by Intel in 1995 with version 2.0 being released in August 2000. A similar protocol called the power management bus (PMBus) was introduced in 2007 that adds two additional signals: control and alert. Due to the wide range of operating parameters, a device that uses the SMBus or PMBus can function with any processors that implement an I2C bus either by bit-banging or using the processor's I2C peripheral hardware. Many processors have hardware support for the I2C network including the Microchip PIC32 processor. The Texas Instruments technical reference guide that describes the implementation of PMBus over I2C is available at http://www.ti.com/lit/an/sprabj6/sprabj6.pdf.

Table I and Table II provide a summary of the differences in the electrical specifications in the implementation of the three communications.

Table I. I2C – SMBus Clock bit rate comparison

	I2C	SMBus	PMBus
Minimum bit rate	None	10 kHz	10 kHz
Maximum bit rate	100 kHz (Standard mode) 400 kHz (Fast mode) 2 MHz (High Speed mode)	100 kHz	400 kHz
Timeout	None	35 ms	
Data packet size	None	32 bytes	256 bytes

Since the hardware used to implement the physical layer of the SMBus is essentially the same as for I2C, the data rates listed in Table I are depended on the type of wire, the distance between the I2C master and the slave devices and the capacitive loading of each slave device connected to the network. Electronic components, such as the Phillips 82B715 I2C bus extender can be used if long distance high speed I2C networks are required. Microchip recommends using a 1.6k ohm pull-up resistor with a maximum bus length of 6 ft. if no bus extender hardware is used. If extender hardware is used, the bus can be extended to 60 ft but the value of the pull-up resistor must be reduced to 160 ohm.

Table II. I2C - SMBus electrical specification comparison

	I2C	SMBus	PMBus
V_{HIGH}	Fixed Voltage: 3.0 to V_{DD_max} + 0.5V V_{DD} Relative: 0.7 to V_{DD_max} + 0.5V	2.1V to V_{DD}	2.1V to V_{DD}
V_{LOW}	Fixed Voltage: -0.5V to 1.5V V_{DD} Relative: -0.5 to 0.3V_{DD}	to 0.8V	to 0.8V
Maximum Sink Current	3 mA	350 µA	350 µA
Bus Pullup Resistor	Vdd = 3V – R_{PU} >1k Vdd = 5V – R_{PU} > 1.6k	Vdd = 3V – R_{PU} > 8.5k Vdd = 5V – R_{PU} > 14k	Same as SMBus

This assignment features a device that communicates using the SMBus and hence the discussion will be limited to addressing the differences between I2C and SMBus. Application Note 476 from Maxim Integrated circuits provides a discussion and the implications of the I2C and SMBus protocols. Texas Instruments Application Report SLOA132 also discusses I2C – SMBus compatibility issues.

One protocol difference lies in the use of the NACK bus signaling: In I2C, a slave receiver is allowed to not acknowledge the slave address, if for example it's unable to receive because it's

performing some real time task. SMBus requires devices to acknowledge their own address always, as a mechanism to detect a removable device's presence on the bus (battery, docking station, etc.)

I2C specifies that a slave device, although it may acknowledge its own address, may decide, at some time later in the transfer, it cannot receive any more data bytes. I2C specifies that the device may indicate this by generating the not acknowledge on the first byte to follow. Other than to indicate a slave's device-busy condition, SMBus also uses the NACK mechanism to indicate the reception of an invalid command or data. Since such a condition may occur on the last byte of the transfer, it is required that SMBus devices have the ability to generate the not acknowledge after the transfer of each byte and before the completion of the transaction. This is important because SMBus does not provide any other resend signaling. This difference in the use of the NACK signaling has implications on the specific implementation of the SMBus port, especially in devices that handle critical system data such as the SMBus host and the SBS components.

The 10 kHz minimum bit rate stems from the face that SMBus slaves are expected to reset their interface whenever Clock is low for longer than the timeout specified in the SMBus specification of 35mS. The SMBus is limited to a clock speed of 100kHz, whereas I²C permits speeds up to 400kHz.

Instrumentation

An instrument is a device or transducer that converts one form of energy of a physical element (voltage, heat, chemical reaction, radiation, light, motion, pressure, etc.) to a mechanical or electrical signal so that the information can be presented for further processing. Some instruments such as thermometers and meters present the information in human readable form so we are able interpret the information and perform the appropriate action. Computers use electrical signals that are further converted into a format that the processor can perform further computations to make appropriate decisions and generate appropriate outputs. Errors can be introduced at any point along the path between the physical quantity being measured and the place the information is used. Physical, thermal, and electrical interference can affect the output of the transducer thus producing unwanted signals most commonly called noise. Once the energy of the physical quantity has been converted to electrical energy, interference is most commonly introduced by electrical coupling to the circuits that convert the transducer information to the media used for transmission and operations of the electrical circuits themselves.

If the transducer information is communicated as an analog signal, interference will result in amplitude, frequency, and phase distortion that can be minimized by proper analog and digital filtering. Converting the transducer signal to a digital format greatly reduces the influence of communications noise. The noise distortion of a digital communications signal results in one or

more binary bit being changed from a logic one to a zero or vise versa. Over the years different techniques have been developed to detect and even correct digital communications errors.

Error Detection for Communication Information

The fact that the SMBus includes a specification for error detection of the data communicated over the I2C serial bus presents the opportunity to address methods of determining if the data contains any errors in the process of transmission. In this section, we will review the common error checking methods. The first data electromagnetic transmission applications in modern time were telegraphy (1809), Morse Code (1836), and teletypewriters (1906), all of which are digital information signals. Data transmission is utilized in computers in computer buses and for communication with peripheral equipment and serial ports such as RS-232 (1969). Currently there are over 25 different serial data methods or protocols and the list keeps growing.

The general idea for achieving error detection and correction is to add some redundancy (i.e., some extra data) to a message, which receivers can use to check consistency of the delivered message, and to recover data determined to be corrupted. Forms of error detection have been in use as far back as 135 AD to ensure absolutely accuracy in the coping of ancient Jewish writings. Analog communications has been sent electronically since the advent of the telephone.

Error detection schemes have been classified into six general methods: repetition codes, parity bits, checksums, cyclic redundancy checks (CRC), cryptic hash functions, and hybrid schemes that use a combination of the first five methods. Error correction codes are organized into three approaches: automatic repeat request, error correcting codes, and hybrid schemes.

We introduce the concept of Hamming distance which is a measure of the number of bits that differ in two binary arrays of equal length. It represents the number of bits that must be changed to make two arrays have the identical sequence of ones and zeros. Many if not most error detection techniques require additional bits and or bytes to be included in the transmission that are generated from the content of the transmitted data. The following error detection techniques are presented as a matter of review of commonly used algorithms.

Repetition Codes

A *repetition code* is a coding scheme that repeats the bits across a channel to achieve error-free communication. Given a stream of data to be transmitted, the data is divided into blocks of bits. Each block is transmitted some predetermined number of times. Automatic Repeat reQuest (ARQ) is an error control method for data transmission that makes use of error-detection codes, acknowledgment and/or negative acknowledgment messages, and timeouts to achieve reliable data transmission. An *acknowledgment* is a message sent by the receiver to indicate that it has correctly received a data frame.

Usually, when the transmitter does not receive the acknowledgment before the timeout occurs (i.e., within a reasonable amount of time after sending the data frame), it retransmits the frame

until it is either correctly received or the error persists beyond a predetermined number of retransmissions. TCPIP, CAN, and I2C are examples of communications protocols that employ this type of error detection and correction.

Repetition codes are very inefficient, and can be susceptible to problems if the error occurs in exactly the same place of the data packet. The advantage of repetition codes is that they are extremely simple.

N out of M Bit Voting

This error detection / correction technique uses bit redundancy. For three bit voting schemes, each binary value is represented in triplicate. Thus a binary one is encoded as "111" and a binary zero as "000". The Hamming distance for the N out on M bit voting M-N. For the 3 bit voter, the Hamming distance is two allowing for single bit error correction. The resultant binary value is a result of 3-bit voting as expressed by Eq. 3. D represents the binary value and B the value of the bits.

$$D_i = (B1 \cap B2) \cup (B1 \cap B3) \cup (B2 \cap B3) \quad \text{Eq. 3}$$

Received Bits			Decoded Binary Value
b2	b1	b0	B
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Byte-Wise Parity Error Detection

Asynchronous serial communications, the ubiquitous technology of yesteryear, allows for the addition of a parity bit to be added to the data bit. A *parity bit* is a bit that is added to a group of source bits to ensure that the number of set bits (i.e., bits with value 1) in the outcome is even or odd. It is a very simple scheme that can be used to detect single or any other odd number (i.e., three, five, etc.) of errors in the output. An even number of flipped bits will make the parity bit appear correct even though the data is erroneous. For even parity, the parity bit is defined by Eq. 1 while the odd parity bit is defined by Eq. 2. The Hamming distance for parity error detection in 1 hence it is not capable of error correction.

$$P_{EVEN} = D0 \oplus D1 \oplus D2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7 \quad \text{Eq. 1}$$

$$P_{ODD} = \overline{D0 \oplus D1 \oplus D2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7} \quad \text{Eq. 2}$$

The data format for an eight-bit asynchronous data byte is shown in Figure 2. The bit position marked “S” represents the start bit, the bit position marked “P” the stop bit, and the bit positions marked “PAR” is for the parity bit. Parity is a form of horizontal redundancy check.

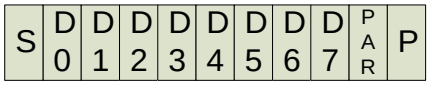

Figure 2. Eight-bit asynchronous data format with parity

Bit Stuffing

One use of bit stuffing is for run length limited coding: to limit the number of consecutive bits of the same value in the data to be transmitted. A bit of the opposite value is inserted after the maximum allowed number of consecutive bits. Since this is a general rule the receiver doesn't need extra information about the location of the stuffing bits in order to do the de stuffing.

This is done to create additional signal transitions to ensure reliable reception or to escape special reserved code words such as frame sync sequences when the data happens to contain them. Zero-bit insertion is a particular type of bit stuffing used in some data transmission protocols to aid clock recovery from the data stream. It was popularized by IBM's SDLC (later renamed HDLC). The name relates to the insertion of only 0 bits. No 1 bits are inserted to limit sequences of 0 bits.

SDLC and Low- and full-speed USB data are sent NRZI encoded: a 0 bit causes a signal transition, whereas a 1 bit causes no change. After a long sequence of 1 bits there could be no transitions in the transmitted data, and it would be possible for the transmitter and receiver clocks to lose synchronization. By inserting a 0 after five (SDLC) or six (USB) sequential 1s the transmitter guarantees a maximum time between transitions. The receiver can synchronise its clock against the transitions to ensure proper data recovery.

In SDLC the transmitted bit sequence "01111110" containing six adjacent 1 bits is the Flag byte. Bit stuffing ensures that this pattern can never occur in normal data, so it can be used as a marker for the beginning and end of frame without any possibility of being confused with normal data.

Controller Area Network (CAN), High Level Data Link Control (HDLC) Network protocol and Universal Serial Bus (USB) all use a form of bit stuffing. Bit stuffing does not ensure that the payload is intact (*i.e.* not corrupted by transmission errors); it is merely a way of attempting to ensure that the transmission starts and ends at the correct places. Error detection and correction techniques are used to check the frame for corruption after its delivery and, if necessary, the frame will be resent.

Checksum

This error detections technique forms a check byte by modulo 8 or module 16 adding all of the bytes in a byte stream. This error detection scheme is a form of longitudinal redundancy check. This approach can detect single bit errors hence has a Hamming distance of one. The advantage of this error checking scheme has over byte-wise parity is number of addition bits that are needed for error checking. Consider a stream of 256 bytes of data. Using Byte-wise

parity error checking, the data stream will contain an additional 256 bits or 32 bytes. Using module 8 checksum, there is only one additional by byte used. The sum may be negated by means of a ones'-complement operation prior to transmission to detect errors resulting in all-zero messages. The check byte is computed using the equation expressed in Eq. 4.

$$Check\_Byte = \left(\sum_i Byte_i \right) moulo 8 \qquad \text{Eq. 4}$$

Another form of checksum is to replace the addition operator used in Eq. 4 with the exclusive OR operator resulting in Eq. 5. The XOR operator requires no more time to compute than does the addition operator. It is capable of multiple bit error provided that the errors occur in different bit positions.

$$Check\_Byte_{i+1} = Byte_{i+1} \oplus Check\_Byte_i,\ i = 0..N-1, Check\_Byte_0 = 0 \qquad \text{Eq. 5}$$

Cyclic Redundancy Check

CRCs are based on the theory of cyclic error-correcting codes. The use of systematic cyclic codes, which encode messages by adding a fixed-length check value, for the purpose of error detection in communication networks, was first proposed by W. Wesley Peterson during 1961. Cyclic codes are not only simple to implement but have the benefit of being particularly well suited for the detection of burst errors, contiguous sequences of erroneous data symbols in messages. This is important because burst errors are common transmission errors in many communication channels, including magnetic and optical storage devices. Typically an *n*-bit CRC applied to a data block of arbitrary length will detect any single error burst not longer than *n* bits and will detect a fraction $1 - 2^{-n}$ of all longer error bursts.

Specification of a CRC code requires definition of a so-called generator polynomial. This polynomial becomes the divisor in a polynomial long division, which takes the message as the dividend, and in which the quotient is discarded and the remainder becomes the result. The important caveat that the polynomial coefficients are calculated according to the arithmetic of a finite field, so the addition operation can always be performed bitwise-parallel (there is no carry between digits). The length of the remainder is always less than the length of the generator polynomial, which therefore determines how long the result can be.

A CRC is called an *n*-bit CRC when its check value is *n* bits. For a given *n*, multiple CRCs are possible, each with a different polynomial. Such a polynomial has highest degree *n*, which means it has *n* + 1 terms. In other words, the polynomial has a length of *n* + 1; its encoding requires *n* + 1 bits. Note that most integer encodings either drop the MSB or LSB bit, since they are always 1. The CRC and associated polynomial typically have a name of the form CRC-*n*-XXX.

The simplest error-detection system, the parity bit, is in fact a trivial 1-bit CRC: it uses the generator polynomial *x* + 1 (two terms), and has the name CRC-1.

Cryptographic hash functions

The output of a *cryptographic hash function*, also known as a *message digest*, can provide strong assurances about data integrity, whether changes of the data are accidental (e.g., due to transmission errors) or maliciously introduced. Any modification to the data will likely be detected through a mismatching hash value. Furthermore, given some hash value, it is infeasible to find some input data (other than the one given) that will yield the same hash value. If an attacker can change not only the message but also the hash value, then a *keyed hash* or message authentication code (MAC) can be used for additional security. Without knowing the key, it is infeasible for the attacker to calculate the correct keyed hash value for a modified message.

Automatic repeat request (ARQ)

Automatic Repeat reQuest (ARQ) is an error control method for data transmission that makes use of error-detection codes, acknowledgment and/or negative acknowledgment messages, and timeouts to achieve reliable data transmission. An *acknowledgment* is a message sent by the receiver to indicate that it has correctly received a data frame.

Usually, when the transmitter does not receive the acknowledgment before the timeout occurs (i.e., within a reasonable amount of time after sending the data frame), it retransmits the frame until it is either correctly received or the error persists beyond a predetermined number of retransmissions.

Three types of ARQ protocols are Stop-and-wait ARQ, Go-Back-N ARQ, and Selective Repeat ARQ. ARQ is appropriate if the communication channel has varying or unknown capacity, such as is the case on the Internet. However, ARQ requires the availability of a back channel, results in possibly increased latency due to retransmissions, and requires the maintenance of buffers and timers for retransmissions, which in the case of network congestion can put a strain on the server and overall network capacity.

Hybrid Error Correcting Schemes

One approach to a hybrid schemes is that the messages are always transmitted with FEC parity data (and error-detection redundancy). A receiver decodes a message using the parity information, and requests retransmission using ARQ only if the parity data was not sufficient for successful decoding (identified through a failed integrity check).

A second approach is where the messages are transmitted without parity data (only with error-detection information). If a receiver detects an error, it requests FEC information from the transmitter using ARQ, and uses it to reconstruct the original message.

The latter approach is particularly attractive on an erasure channel when using a rate less erasure code.

Error Detection for the SMBus Programming

Version 1.1 of the SMBus provides for the inclusion of Packet Error Correction (PEC). PEC is a 8-bit CRC that uses the primitive polynomial $X^8 + X^2 + X^1 + 1$ to generate the checking code. PEC is optional but is required if participating in the address resolution process (ARP) enumeration process which is a method of resolving two devices with the same device ID. The APR processes requires the device be able to have its ID dynamically reassigned.

Melexis MLX90614 IR Temperature Sensor Assignment

The original reference design project is divided into three programs that use many of the same support libraries. The first program is used to configure a MLX90614 IR Temperature Sensor. The second program monitors a single sensor device and the third monitors three IR sensors on one SMbus network. Figure 3 is a picture of the two circuits used for this reference design. Schematic diagrams are provided in the discussions below. The chpKIT Prom MX7 processor board includes the pull-up resistors that are needed for the I2C network.

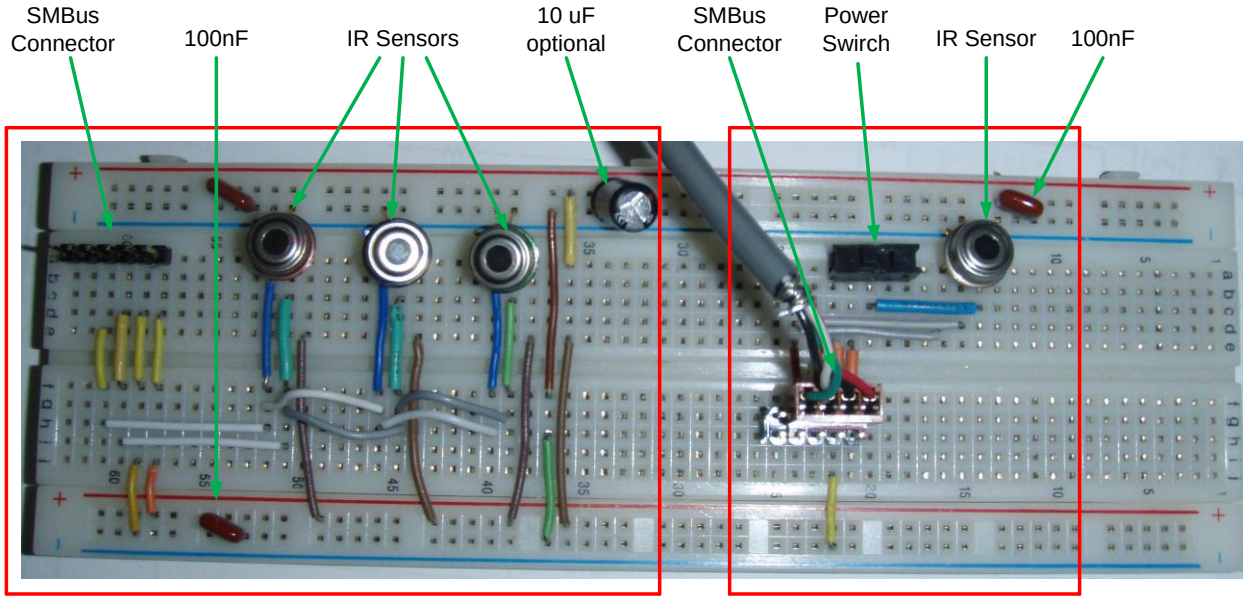

Figure 3. Picture of breadboard sensor network and sensor programming circuits

Except for the *program_ir* function, both programs use a data flow model as shown in Figure 6. The *IRSensor_lib* files provide the low level I2C functions to support reading the RAM and EEPROM as well as writing to the EEPROM to change the sensor configuration by the *program_ir* function. When reading the IR sensor EEPROM or RAM, two SMBus packets are needed as shown in Figure 4. The first packet consists writing a command that identifies the EEPROM or RAM address to be read. EEPROM address are offset from RAM address by a

value 0x20. The second packet reads the 16 bits of data as two bytes followed by the PEC byte. In this implementation of the IRSensor_lib, the PEC is computed but not used to check for data errors. Figure 5 shows the format for writing to the EEPROM of the IR sensor. The correct PEC must be appended to the two bytes of data or the write operation will be ignored by the IR sensor.

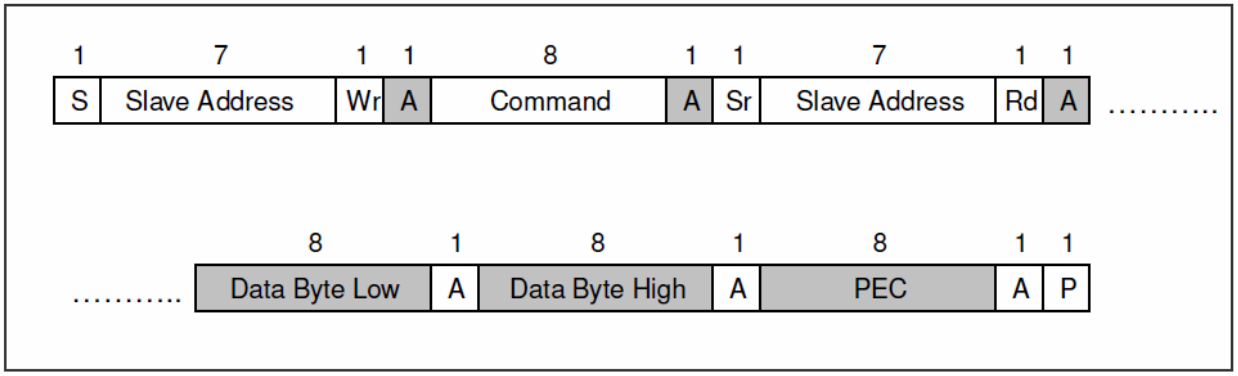

Figure 4. SMBus read word packet format

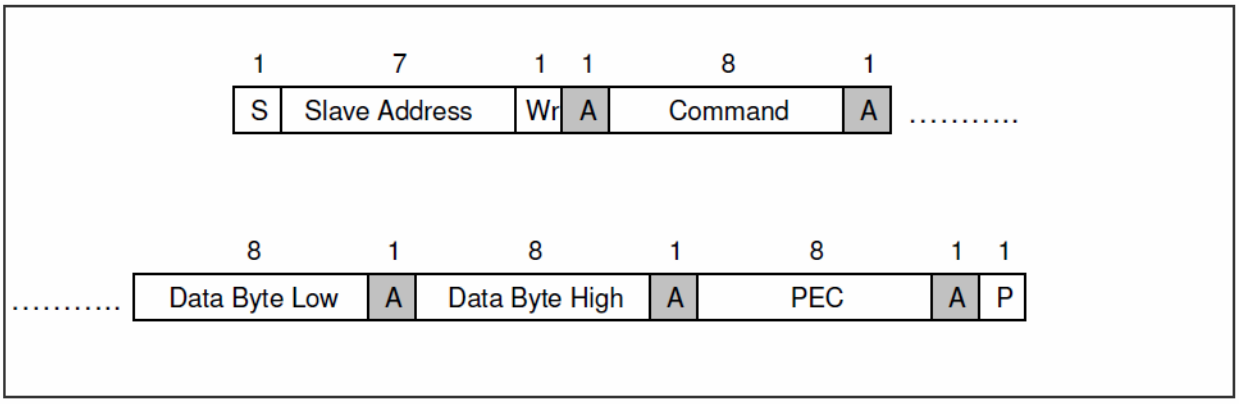

Figure 5. SMBus write packet format

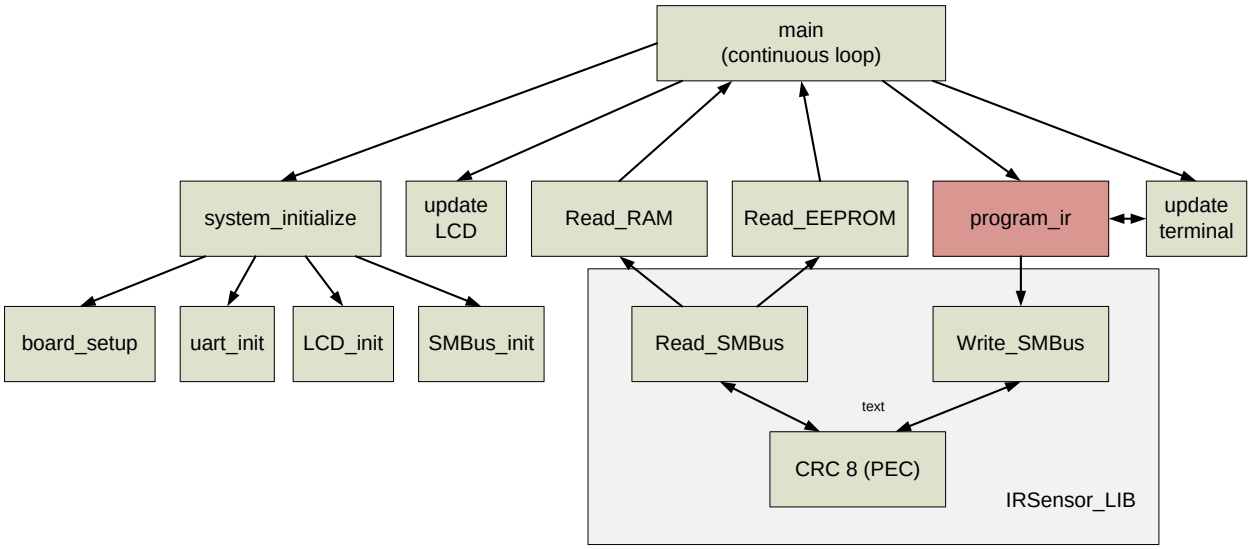

Figure 6. Data flow diagram for program to configure the IR sensor

As shown in Figure 6, the *IRSensor_lib* file also contains the function to compute the PEC value. The PEC value can be compute are two methods: a byte-wise table lookup method or a bit-wise algorithmic method. When implemented on a PIC32 running at 80 MHz, the lookup table method requires 348 bytes of code memory and executes in 2.8 μs. The algorithmic method needs 216 bytes of code memory but executes in 11.5 μs. Based on the amount of memory available in the PIC32 processor, one can conclude that the lookup table method is preferable since if executes four times faster. Similar results can be extrapolated to different processors running at different speeds.

If the software does not receive an acknowledge bit from the IR sensor device during initialization, the error message shown in Listing 1 will be displayed on the computer terminal after a system reset. If this error message is observed, the there is either a wiring error or the device ID of the IR sensor is incorrectly coded in the application program. (The ID of 90 shown in Listing 1 is the default ID value of 0x5A.)

Listing 1

```
chipKIT PRO MX7 Serial Port 1 ready

 I2C1 clock frequency = 100000 Hz
Error: Sent byte was not acknowledged
I2C check data failed for ID 90.
```

Configuring the MLX90614 IR Sensor

The circuit shown in Figure 7 is used for configuring the MLX90614 IR Temperature Sensor. Since all IR devices will acknowledge the device ID of 0x00 that is used when programming, the SMBus used for programming must clearly have only one IR sensor device. The sensor data sheet clearly states that the SMBus must not have multiple IR sensors that are programmed for the same device ID and hence does not support SMBus ARP capability.

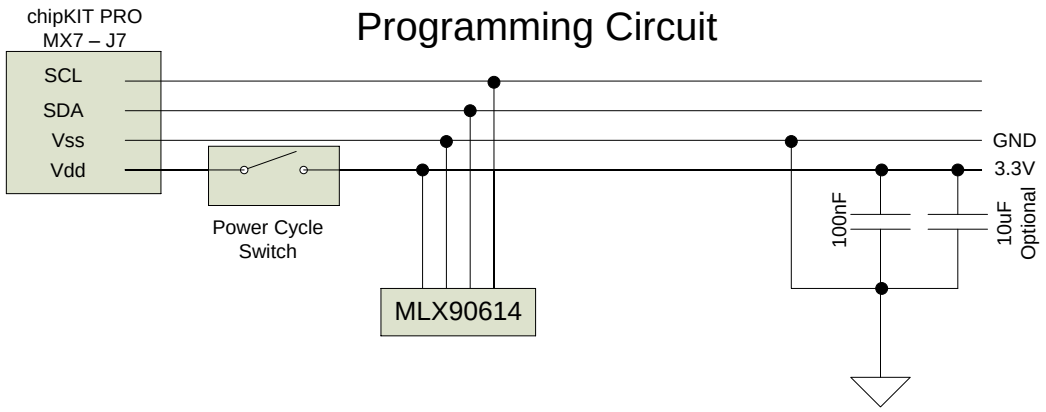

Figure 7. Schematic diagram for programming the IR sensor

When this program begins, it prompts the user to use the default device ID of 0x5A as shown in Listing 2 below. If BTN 1 is pressed within 10 seconds of reset, the IR sensor will be programmed for the default value.

Listing 2.

```
chipKIT PRO MX7 Serial Port 1 ready
 I2C1 clock frequency = 100000 Hz
Press BTN1 to set IR device ID to 0x5A (default)
```

If the processor can read the IR sensor EEPROM, the configuration data is displayed on the computer terminal followed by the real-time air and IR temperature read from the senor's RAM as shown in Listing 3. The temperature is updated every 250ms.

Listing 3.

```
Device ID:5A - Configutation:9FB4 - PWM Control:0201
Emissivity: 65500d - T0max:39315d - T0min:25315d - Ta range:63260d

Air: 81.41 T1:  81.45    ID 5A CFG 9FB4
```

Entering the ESC key on the terminal will cause the program to begin operating in the sensor configuration programming mode and will display the prompt as shown in Listing 4. The configuration data can be changed using the format shown in the prompting text. Listing 5 is an example of changing the emissivity value to 0.992. Entering the text “RUN=” on the terminal is required to exit the programming mode. Note the prompt to cycle the IR sensor using the switch provided on the breadboard. Listing 6 shows the terminal message after exiting from the the programming mode. Refer to the IR sensor data sheet for additional information concerning the configuration settings.

Listing 4.

```
IR Sensor Programming mode
Only one IR sensor should be attached to the I2C network.
Power must be removed from sensor device for programming to take effect.
Entry format:  CMD=####

RUN=  Return to sensor mode
EMS=  EMMISIVITY in range of 0 to 65535 - present setting =65500
CFG=  Configuration register in range of 0 to 65535 - present setting =40884
PWM=  PWM control in range of 0 to 65535 - present setting =513
TAR=  Ta control in range of 0 to 65535 - present setting =39315
T0X=  T0min control in range of 0 to 65535 - present setting =1
T0N=  T0max control in range of 0 to 65535 - present setting =25315
DEV=  DEVICE ID number in range of 1 and 127


Enter text:
```

Listing 5.

```
Enter text: EMS=65000
EMISSIVITY set for 65000d
Enter text: RUN=

Returning to RUN mode
Power cycle the IR sensor and hit SPACE key to continue -
```

Listing 6.

```
Device ID:5A - Configutation:9FB4 - PWM Control:0201
Emissivity: 65000d - T0max:39315d - T0min:25315d - Ta range:63260d
Part Number 9406|6fb8|82a4|b0d2
Air: 82.13 T1:  81.81    ID 5A CFG 9FB4
```

The MLX90614 IR Sensor Network

The single IR sensor system can use either one of the circuits shown in Figure 7 or Figure 8. The one precaution is to be sure to use the sensors with the correct device IDs. The terminal display for the single sensor network is the same as for the multiple sensor network except for the number of devices reported.

Multiple MLX90614 IR Sensor Network

The schematic diagram for the multi-sensor network is shown in Figure 8. Since the MLX90614 IR sensor does not support APR device discovery, the application program must be coded with the device ID values set using the device configuration application program. The device ID values used in this example are 0x64 through 0x66. The optional 10uF capacitor is included power supply filtering needed on networks with long wires between to power source and the devices.

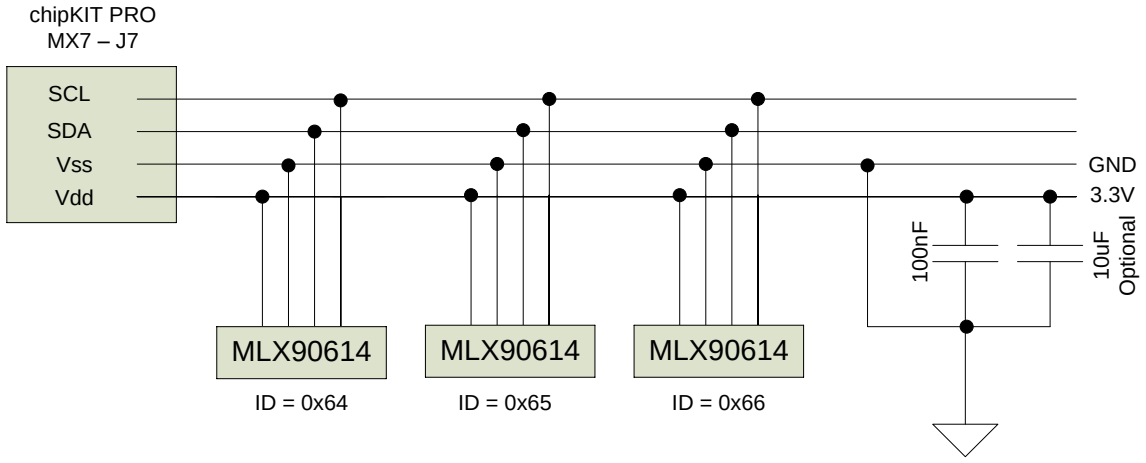

Figure 8. Schematic diagram of the multi-sensor I2C network

Listing 7 shows the output to the computer terminal when scanning the three IR sensors that are shown in Figure 3. This display scrolls up the terminal screen as the new data is polled.

Listing 7.

```
Air: 72.30 T1:  72.66   ID 64 CFG 9FB4
Air: 71.83 T1:  71.91   ID 65 CFG 9FB4
Air: 71.80 T1:  72.77   ID 66 CFG 9FB4

Air: 72.19 T1:  72.63   ID 64 CFG 9FB4
Air: 71.91 T1:  72.23   ID 65 CFG 9FB4
Air: 71.83 T1:  72.73   ID 66 CFG 9FB4
```

Appendix A: HW 3 Parts Configuration

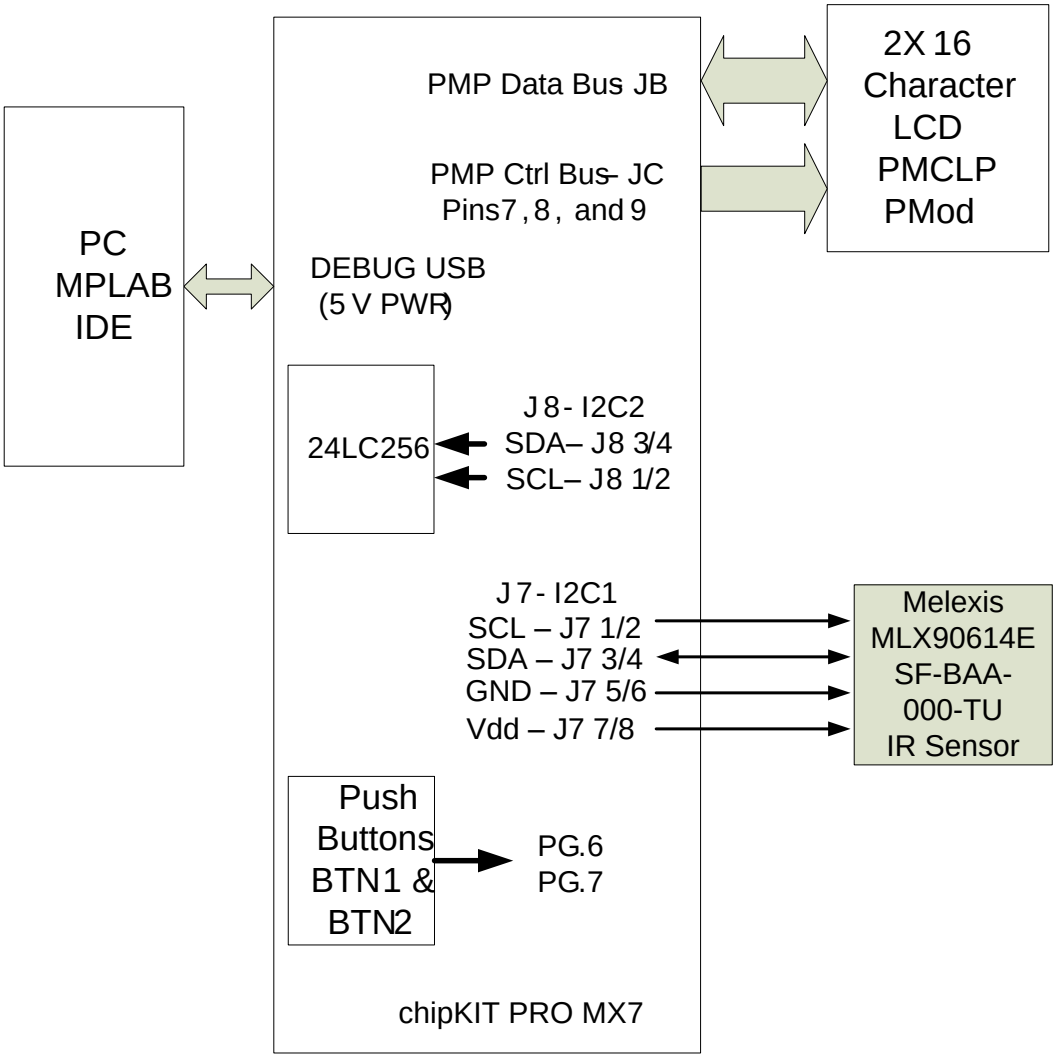

Figure 9. Block diagram of the equipment used in ECE 443 HW 3.