Laboratory Assignment #7
Communication via I2C

Autumn 2013 TCES 430 – Microprocessor System Design

by

Alvin Baldemeca and Edward Bassan

Date: 11/26/2013

Table of Contents

I. Objective

The purpose of this lab is to write a C program for the Microchip's PICDEM2 Plus Demonstration board that would write and read a page of data into the 24LC256 EEPROM (Electrically Erasable Programmable Read-Only Memory) that is on the demo board. We are given instructions not to use the XC8 I2C library functions. It is also stated in the instructions that we complete the code template provide on TCES 430 class site (canvas.uw.edu) and to use the completed code in our program. Lastly we also need to design a way to verify that the page of data is written to the EEPROM properly.

II. Procedure

I first created a new project for the MPLABX IDE, by choosing a standalone project. See Figure 1. Following the steps to create a new project I selected is the PIC18F4520 for the device. For our hardware tools I selected the ICD3, finally we selected the XC8 compiler to compile our program.

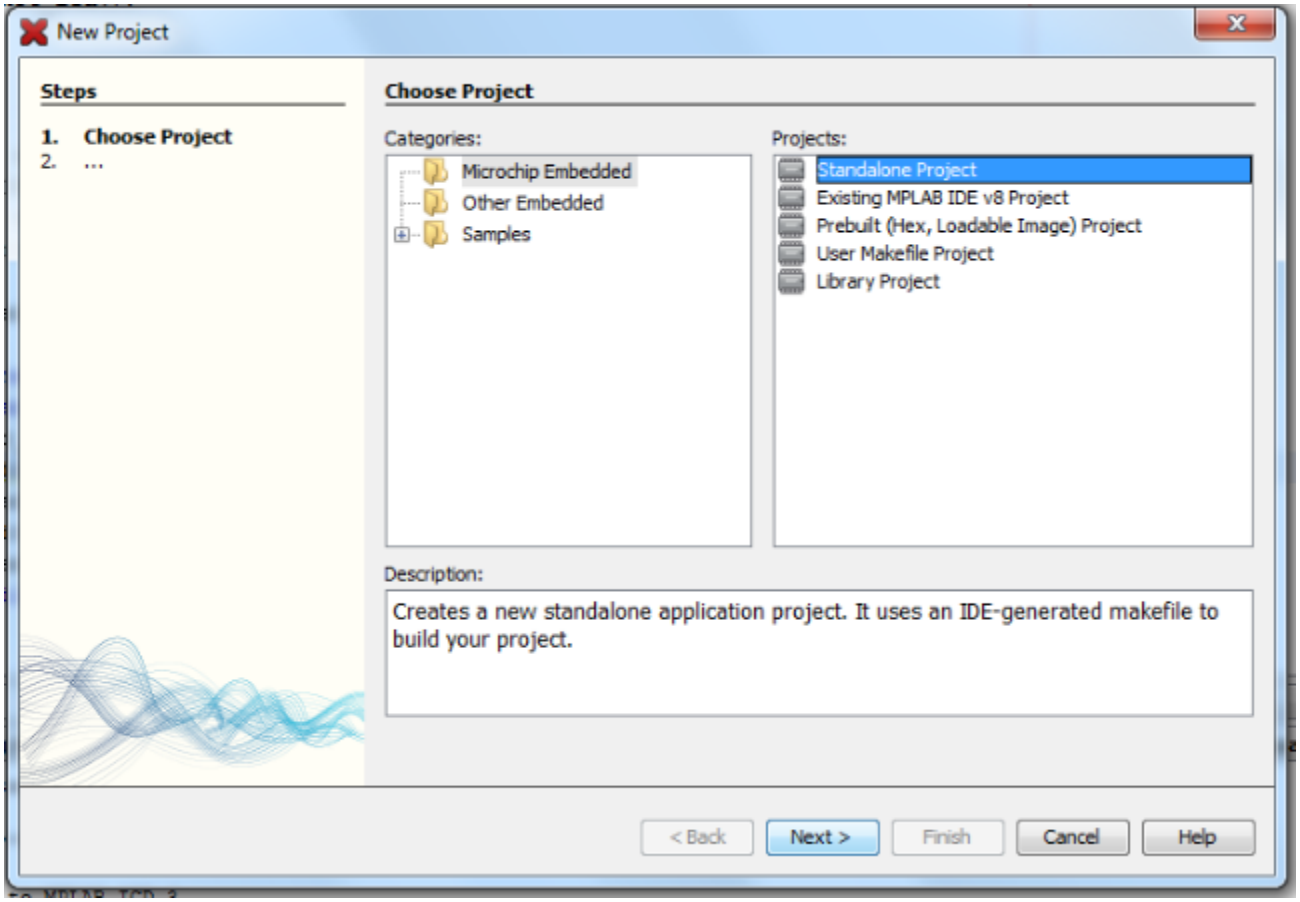

FIGURE 1. MPLABX New Project pop up screen

Once the new project was created I created a new source file and copied and pasted the code template provided. See Appendix A for the code template given. The template given is written for the Microchip's C18 compiler, however the compiler chosen to be used in this project is the XC8 so the template had to be ported over to the XC8 compiler. To verify that a page of data is written to the eeprom properly, the created program retrieves the data written to the eeprom and displays this on the LCD (liquid crystal display) of the demo board.

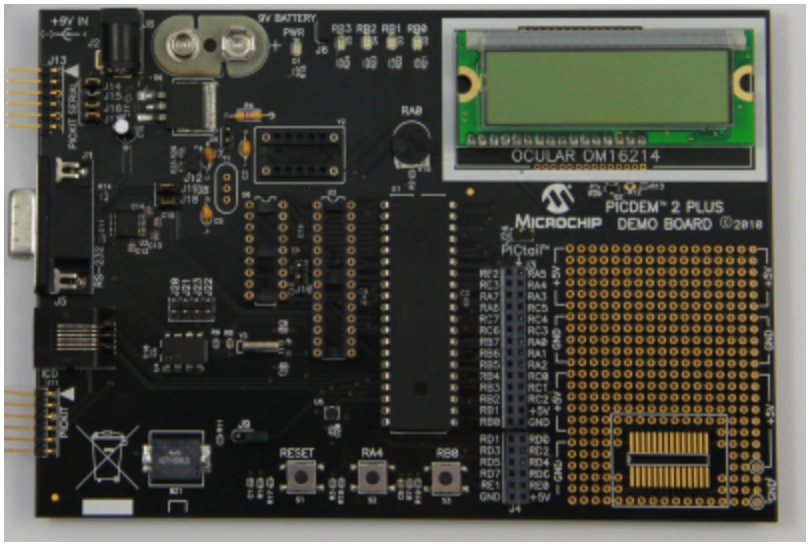

FIGURE 2. PICDEM 2 PLUS demo board

III. Data and Analysis

III.A Introduction I²C

This section will talk about the process that were taken in-order to create the program. It will go into some detail of how the program works and talk about parts of the code. In the following sub-sections I'll talk about: how to configure the microcontroller, how to send data, how to receive data and finally problems and difficulties with this lab.

The first thing we need to do is to understand the principles Inter-Integrated Circuit (I^2C) protocol and how it works as suggested in assignment instructions.

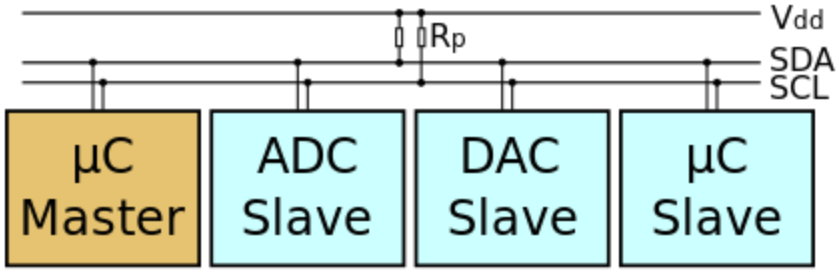

FIGURE 3. A sample schematic with one master (a microcontroller), three slave nodes (an ADC, a DAC, and a microcontroller), and pull-up resistors R_p (wiki link http://en.wikipedia.org/wiki/I%C2%B2C)

As seen on Figure 3, I²C uses two wires which are the Serial Data Line (SDA) and Serial Clock (SCL). These two lines are bidirectional and has a pull-up resistor (R_p) because the components tied to the lines are open-drain. Multiple master and slaves can communicate using these two lines. In our case we will just focus on the PIC18F4520 microcontroller as the master and the 24LC256 eeprom as the slave.

III.B How do we configure the PIC18F4520?

How do we configure the micro-controller for this lab? To answer this question we need to read the datasheet for both the master and slave devices. To configure the serial port for SDA (PORTC3) and SCL (PORTC4) we need to enable the SSPEN bit of the SSPCON1 by writing: `SSPCON1bits.SSPEN = 1;` in the code. **Section 17.4 I²C MODE** of the datasheet contains important information that will be discussed and should be reviewed for further understanding.

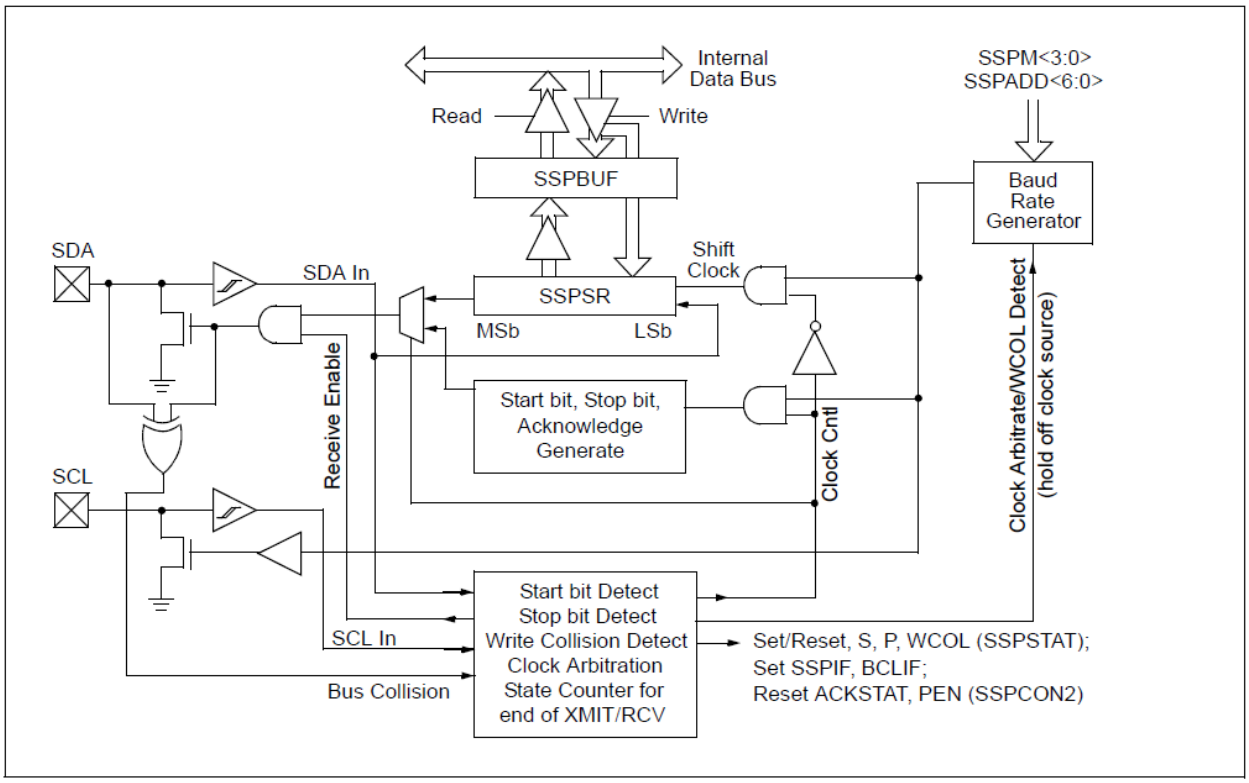

FIGURE 4. MSSP BLCOCK DIAGRAM (I²C MASTER MODE)

Section 17.4.6 contains the FIGURE 4, which is seen above. The first thing we need to do to initialize the PIC is to set the tristate buffer (seen here)

that is tied to SDA and SCL for input, this makes the lines bi-directional. In the code this is done by setting `TRISC3` and `TRISC4` to 1. See the code below:

```
void i2c_init(void){
    SSPCON1bits.SSPEN = 1;
    SSPCON1bits.SSPM = 0X8;  //initialize I2C mode
    TRISC3 = 1; //CLOCK
    TRISC4 = 1; //DATA
    SSPADD = 0X04; // 8MHZ /(4*(SSPAD + 1) = 400KHZ, SSPAD = 0X04
    return;
}
```

To set the mode for I²C we have to set the `SSPM` bits of the `SSPCON1` register. See the snippet below (Figure 5) taken from the data sheet. In the code above we set this by doing the following: `SSPCON1bits.SSPM = 0X8;`

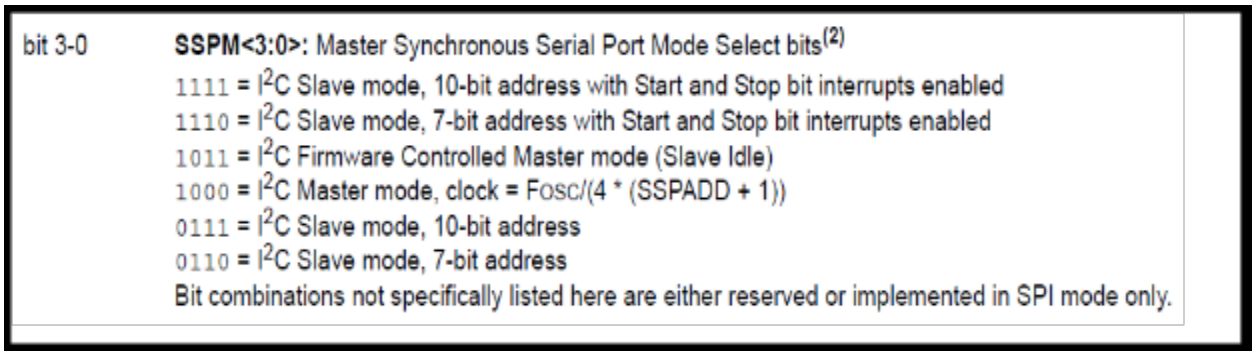

FIGURE 5. I²C Configuration (Setting SSPM bits)

The last thing we need to configure is the `SSPAD` which as seen above for I²C MASTER mode in Figure 5 affects the clock (CLK) or the Baud Rate Generator seen on the block diagram on Figure 4. Other bits can be configured but this is the most basic. Other configuration like setting up I²C interrupt, slew rate etc. ,will not be discussed here. The registers on the PIC18F that are important to understand and read on the datasheet are: `SSPCON1, SSPCON2, SSPSTAT, SSBUF` and `SSPADD`.

III.C How do we send data to the slave?

In summary we have to send a start bit, then send the address with a write bit and wait for an acknowledge from the slave before we send data. We will continue the discussion in a little more detail, specific to the PIC18F and the eeprom. The I²C protocol states that a `START` bit has to be given before any communication can happen. A `START` condition happens when CLK is high and SDA transitions from high to low (see Figure 6). Of course its necessary to check the the BUS isn't busy to avoid collision therefore in theory and in practice we should make sure that the CLK and SDA is high for 2 or more clock cycle.

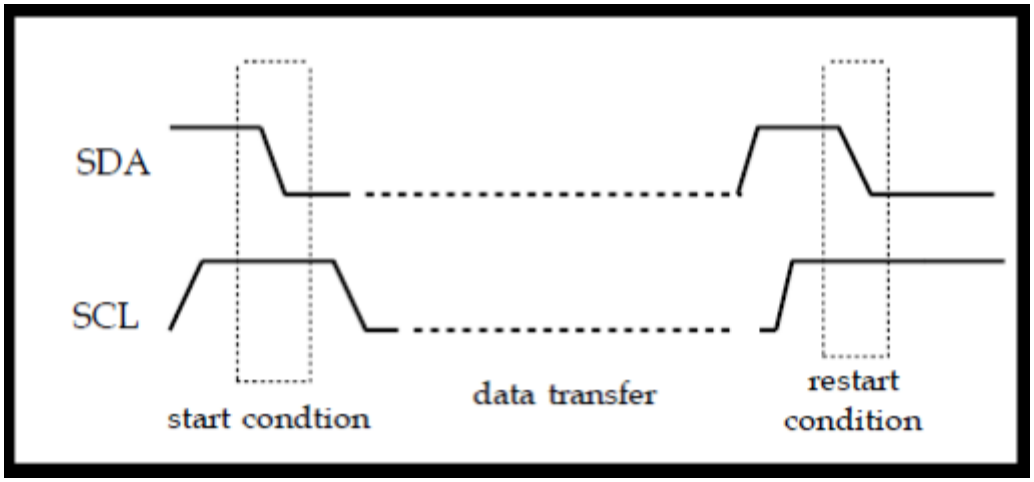

FIGURE 6. I²C Start and Restart Condition

Once the start signal has been sent we then send a byte of data which contains the address of the slave device we want to talk to bits <7:1> and weather we want to read(bit<0> = 1) or write (bit<0> = 0). Note: The 24LC365 eeprom is a 7-bit address device with an address of `0b1010000` configured on the PICDEM2 board. Figure 7 shows how to send the address and read and write mode for a 7-bit and 10-bit address device.

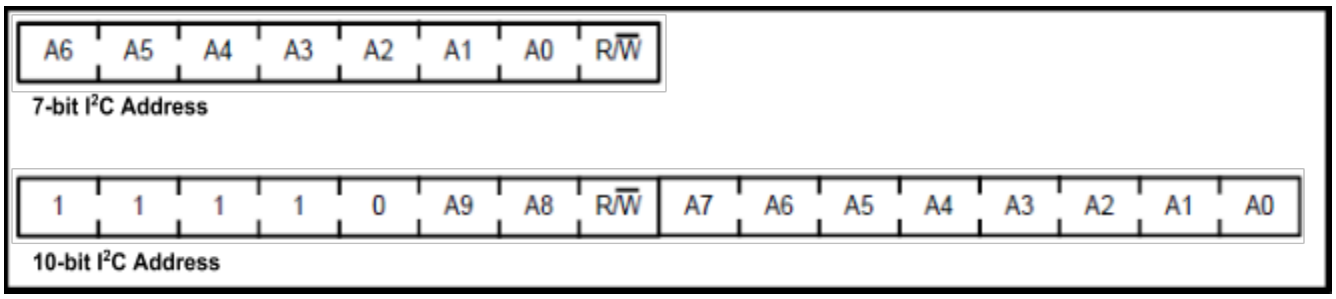

FIGURE 7. 7-bit and 10-bit Read/Write addressing

After each byte of data sent we must wait for an acknowledge (`ACK`) from the slave. If we don't receive an acknowledgement from the slave we either gave the wrong address or the slave didn't get all the bits. If its the wrong address then we must debug and troubleshoot our project to make sure we give the right address. If the slave didn't get all the bits the master must give a restart signal (see Figure 6) , send the data again and wait for an acknowledgement from the slave. Below is a code snippet of how this is accomplished. The the full implementation of the function for I²C can be seen in Appendix A i2c.c code.

```
unsigned char i2c_send(unsigned char data)
{
  i2c_idle();
  SSPBUF = data; //write single byte to SSPBUF(transmitted to slave)
  if ( SSPCON1bits.WCOL )      // test if write collision occurred
```

```
    return ( 1 );               // if WCOL bit is set return 1
  else
  {
    while( SSPSTATbits.BF ); // wait until write cycle is complete
    i2c_idle();                 // ensure module is idle
    if ( SSPCON2bits.ACKSTAT ) // test for ACK condition received
      return ( 2 );             // return NACK
    else return ( 0 );              //return ACK
    }
}
```

Once the slave device is activated we need to send the high-byte of the address (8bits) and wait for an `ACK` (acknowledge) from the slave then send the low-byte address and wait for an `ACK` again. Subsequent data will be written to the memory address in sequential order for example if the we we start to write at memory address `0x0100`, we just keep writing data and the next place the eeprom saves it to is address `0x0101` from a single byte up to a page (64 bytes). NOTE: It is important to make sure that write not cross page boundaries because data will be wrapped around to the beginning of the current page according the eeprom's datasheet. Lastly, and a very importantly we have to issue a `STOP` signal to the slave. When the eeprom receives this signal it initiates an internal write cycle. If this signal is never given the data sent to the slave just stays in its buffer which could get lost and never gets save.

On the PIC18F we write the byte we want to transmit to the SSBUF, the microcontroller then transmits this over the bus according to the baud rate we configured. When the write cycle completes the `BF` flag of the `SSPSTAT` register is cleared (`BF = 0`).

III.D How do we retrieve data from the slave?

Retrieving data is much the same way as sending data except that we issue a high (1) to indicate that the master wants to read or retrieve data from the eeprom. The first thing we must do is make sure that the bus is idle, send a `START` then we issue a write to the address we want to read from. Before we write data we issue a `STOP` signal, why? This forces the internal pointer on the eeprom to point at the starting address we wish to read from, then we must issue a `START` signal. We then send the 7-bit address for bits<7:1> and a high for bit<0> (see Figure 7) to tell the eeprom that we want to read. The eeprom will then send the microcontroller (master) a byte of data the microcontroller must follow this with an acknowledge so that the slave knows to send the next byte. Once we got (the microcontroller received) all data needed the master must issue a stop signal.

On the PIC18F upon receiving a byte the `SSPSR` register shifts this byte to the `SSBUF` register and sets the `BF` flag on the `SSPSTAT` register. The first read of the `SSBUF` clears the `SSBUF` register and the `BF` flag. Another important bit to set is the `RCEN` flag on the `SSPCON2`

register to enable the microcontroller receive data, the code below shows how we implement an I^2C receive function in C.

```
unsigned char i2c_receive(){
    unsigned char data;
    i2c_idle();
    SSPCON2bits.RCEN = ENABLE;// enable master for 1 byte reception
    while(SSPCON2bits.RCEN);
    while (!SSPSTATbits.BF); // wait until byte received
    data = SSPBUF;
  return ( data );
}
```

One we received the data it is stored in a buffer (`char array[ ]`). I've decided to received a byte then display this on the LCD on the PICDEM2 board. The code below shows the main function which scrolls the message on LCD.

```
void main (void){

    const char mypage[64] = "My message to you : "\
    "Have a Merry Christmas and a Happy New Year\0";
    char message[64];
    i2c_init(); // Initiate for I2C
    write_page(mypage, MSB_ADD, LSB_ADD);
    read_page(message, MSB_ADD, LSB_ADD);
    init_lcd(); //Initiate LCD
    load_string(message); //Load the page to LCD
    while (1)
    {
        send_data(0x18,1,1,1); //scroll display
        __delay_ms(SCROLL_SPEED); //Slow down the scroll
    }
}
```

III.E Problems and Difficulties

The biggest difficulty in doing this lab understanding the hardware microcontroller and the eeprom. The concepts and principles behind I^2C is easy to understand and infact would be much easier if this was done manually, like sending bits and clocking the CLK line. I had to understand the I^2C features of the PIC18F4520 to ensure that the hardware is doing what I want it to do.

Another issue I had was the timing, when I was debugging I was not sure if the baud rate I selected was correct and considered putting an logic analyzer (oscilloscope with this feature) so that I can see what the SDA and CLK lines were doing. When writing a page I tried to create a `for loop` but discovered that the `for loop` was taking too long causing the program to miss the acknowledge signal from the slave. This was a very challenging lab because one can completely understand the principles of I^2C and not have their program work because of the timing issue. I had the code in one big C file which I've attached to Appendix A Code HW7_LAB7.c, I soon realized that this was too big and messy so I've split the code into different source files and headers. The complete project was compressed into a .zip file and turned in along with this report.

IV. Conclusion

I was able to program the PIC18F4520 microcontroller to read and write a page of data to the external eeprom on the PICDEM2 demo board. I was able to verify that the page of data written to the eeprom is correct by retrieving the data saved and displaying this on the LCD of the PICDEM2. I learned more about the PIC18F microcontroller as well as the 24LC256 EEPROM and the principles and concepts of the I^2C protocol. It is important to understand the timing as well as the features and specifications of the devices in order to fix and debug any issues.

Appendix A Code for 4520I2C_temp.c

```
//This program provide a template to write and read an EEPROM
//using the PIC MSSP in I2C synchronous serial mode.

#include <p18f4520.h>

#pragma config WDT = OFF


volatile unsigned char flag;


#pragma code high_vector=0x08
void interrupt_at_high_vector(void){
        _asm
                goto i2c_isr
        _endasm
}
#pragma code

void i2c_idle(void);
void i2c_start(void);
void i2c_rstart(void);
void i2c_stop(void);
void i2c_ack(unsigned char ackbit);
unsigned char i2c_send(unsigned char data);
unsigned char i2c_receive(unsigned char ackbit);
void i2c_init(void);
void i2c_isr(void);
void delayms(unsigned char cnt);


void main (void){

        i2c_init();

        while (1){
                //
                //
```

```
            //add your codes here for the required tasks
                //
                //
        }
}

void i2c_idle(void){
        unsigned char stat;
        do{
                stat = SSPCON2 & 0x1f; //i2c function control status
        } while(stat | SSPSTATbits.R); //wait for stat = 0
        return; //and no transmit
}

void i2c_start(void){
        i2c_idle();     //wait for idle bus
                        //initiate start
                        //and wait for start to finish
        return;
}

void i2c_rstart(void){
        i2c_idle();     //wait for idle bus
                        //initiate restart
                        //and wait for restart to finish
        return;
}

void i2c_stop(void){
        i2c_idle();     //wait for idle bus
                        //initiate stop
                        //and wait for stop to finish
        return;
}

void i2c_ack(unsigned char ackbit){
                        //set acknowledge bit
                        //initiate acknowledgement
                        //wait for ack to finish
        return;
}

unsigned char i2c_send(unsigned char data){
```

```
	unsigned char i;
			//wait until data sent
			//check ack status
			//
	return(i);	//return ack status
}

unsigned char i2c_receive(unsigned char ackbit){
	unsigned char data;
			//initiate read
			//wait for read to finish
			//wait for buffer full
			//get data
			//send ackkowledgment
	return(data);
}

void i2c_init(void){
			//
			//
			//
			//
	return;
}

#pragma interrupt i2c_isr
void i2c_isr(void){
	if(INTCONbits.INT0IF){ //check interrupt flag
		INTCONbits.INT0IF = 0; //clear interrupt flag
		flag = 1; //set semaphore
	}
	return;
}
#pragma code

#pragma udata access accessdata

void delayms(unsigned char cnt){
	static near unsigned char i,j;
	for (i=0; i<cnt; i++){
		j=245; //approx. 1 ms delay at 4 MHz
		_asm
		loop: nop
```

```
            decfsz j,1,1
            bra loop
            _endasm
        }
        return;
}
```

Appendix A Code for i2c.c

```
/*
 * Author  : Alvin Baldemeca
 * Credits : These functions were modified from the code provided by Dr. Sheng
 *           from University of Washington Tacoma, TCES 430.
 *
 * Version : 11.23.2013
 *
 * Copy Right (C):
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * any later version. ©
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 */

/*******************************************************************************
 * FileName:                i2c.c
 * Dependencies:       See include below , PICDEM2 PLUS demo board
 * Processor:               PIC18F4520
 * Compiler:                XC8
 *
 * Synopsis :
 *  #include "i2c.c"
 *   i2c_idle();
 *   i2c_start();
 *   i2c_rstart();
 *   i2c_stop();
 *   i2c_ack(unsigned char ackbit);
 *   i2c_send(unsigned char data);
 *   i2c_receive(unsigned char ackbit);
 *   i2c_init()
 *
 * Description :
 *   i2c_idle()   - checks for idle conditions
 *   i2c_start()  - sends a start condition
```

```
 *   i2c_rstart() - sends a restart condition
 *   i2c_stop()   - sends a stop condition
 *   i2c_ack(unsigned char ackbit) - sends acknowldege or no acknowledge
 *   i2c_send(unsigned char data) - send a byte of data
 *   i2c_receive(unsigned char ackbit) - recieves a byte of data
 *   i2c_init() - initializes microcontroller for I2C
*******************************************************************************/



#include "global.h"

/**
 * Used for Inter-Integrated Circuit (I2C) communication to check for when the
 * lines are idel
 */
void i2c_idle(void){
   while((SSPCON2 & 0x1F)| SSPSTATbits.R_NOT_W); //Check all flags for idle
   return;
}

/**
 * Used for Inter-Integrated Circuit (I2C) to have the micro-controller send a
 * start condition SDA (LOW to HIGH), SCL (HIGH).  This is a feature of the
 * PIC18F microcontroler i.e. SSCON2bits.SEN.
 */
void i2c_start(void){
        i2c_idle();             //wait for idle bus
        SSPCON2bits.SEN    = 1;     //initiate start
        while(SSPCON2bits.SEN); //and wait for start to finish
        return;
}

/**
 * Used for Inter-Integrated Circuit (I2C) to have the micro-controller send a
 * re-start condition SDA (LOW to HIGH), SCL (HIGH) with out issuing a stop.
 * This is a feature of the PIC18F microcontroler i.e. SSCON2bits.RSEN
 */
void i2c_rstart(void){
        i2c_idle();                     //wait for idle bus
        SSPCON2bits.RSEN=1;
        while(SSPCON2bits.RSEN); //and wait for restart to finish
        return;
```

```
}

/**
 * Used for Inter-Integrated Circuit (I2C) to have the micro-controller send a
 * stop condition SDA (HIGH to LOW), SCL (HIGH).  This is a feature of the
 * PIC18F microcontroler i.e. SSCON2bits.PEN
 */
void i2c_stop(void){
        i2c_idle();                 //wait for idle bus
        SSPCON2bits.PEN= 1;           //initiate stop
        while(SSPCON2bits.PEN);     //and wait for stop to finish
        return;
}

/**
 * Used for Inter-Integrated Circuit (I2C) to have the micro-controller send a
 * ACKNOWLEDGE or NO-ACKNOWLEDGE. stop condition SDA (HIGH to LOW), SCL (HIGH).
 * This is a feature of the PIC18F microcontroler i.e. SSCON2bits.ACKDT
 *
 * @param ackbit - 1 for acknowledge, 0 for no-acknowlege
 */
void i2c_ack(unsigned char ackbit){
   if ( ackbit )
   {
      SSPCON2bits.ACKDT=0; //set acknowledge bit
   }
   else
   {
      SSPCON2bits.ACKDT=1; //set no acknowledge bit
   }
   i2c_idle();
   ACKEN=1; //initiate acknowledgement
   i2c_idle(); //wait for ack to finish
   return;
}

/**
 * Used for Inter-Integrated Circuit (I2C) to have the micro-controller send
 * data to a slave device (client for the politically correct).
 *
 * @param data - the data to send
 */
unsigned char i2c_send(unsigned char data)
```

```
{
   i2c_idle();
   SSPBUF = data;          // write single byte to SSPBUF
   if ( SSPCON1bits.WCOL )      // test if write collision occurred
     return ( 1 );             // if WCOL bit is set return 1
   else
   {
     while( SSPSTATbits.BF );   // wait until write cycle is complete
     i2c_idle();               // ensure module is idle
     if ( SSPCON2bits.ACKSTAT ) // test for ACK condition received
       return ( 2 );                          // return NACK
     else return ( 0 );             //return ACK
   }
}

/**
 * Used for Inter-Integrated Circuit (I2C) to have the micro-controller recieve
 * data from a slave device (client for the politically correct).
 *
 * @param ackbit - 1 for acknowlege and 0 for no-acknowledge
 * @return - the recieved byte
 */
unsigned char i2c_receive(unsigned char ackbit){
   unsigned char data;
   i2c_idle();
   SSPCON2bits.RCEN = ENABLE;// enable master for 1 byte reception
   while(SSPCON2bits.RCEN);
   while (!SSPSTATbits.BF); // wait until byte received
   data = SSPBUF;
   i2c_ack(ackbit);
  return ( data );
}

/**
 * @breif Initializes the PIC18F micro-controller for Master I2C mode.
 *
 * TODO - should make this a macro since its only used once, makes the main func
 * bigger but should initialize faster (obsessed with speed!!!)
 */
void i2c_init(void){

   /*
    * Enable Master Synchronous Serial Port and configures SDA and SCL pins as
```

```
     * serial port pins for I2C protocol.
     *
     * SSPCON1 = I2C_M_E_CONF;  Equivalent to code below (faster)
     */
    SSPCON1bits.SSPEN = ENABLE;
    SSPCON1bits.SSPM = I2C_M;  //initialize I2C mode

    /*
     * This is not needed if used on the PICDEM2 Plus board but is good practice
     * as for manufacturer's recomendations
     */
    SSPSTATbits.CKE = ENABLE; //Enable SMBus specific inputs
    SSPSTATbits.SMP = ENABLE; //Disable maximum rate of change of output voltage per unit of
time

    /*
     * Set SDA and CLK as inputs on the micro-controller
     */
    TRISC3 = INPUT; //CLOCK
    TRISC4 = INPUT; //DATA

    SSPADD = BAUD_100KHZ; // 1MHZ /(4*( 0X09 + 1) = 100KHZ, SSPAD = 0X09
    return;
}
```

Appendix A Code for HW7_LAB7.c

```
//This program provide a template to write and read an EEPROM
//using the PIC MSSP in I2C synchronous serial mode.



////////////////////////////////Header info ////////////////////////////////////
//Should put this in a .h file but placed here for simplicity ////////////////////
//////////////////////////////////////////////////////////////////////////////
#include <p18f4520.h>
#include <plib/i2c.h>
#include <xc.h>

#pragma config WDT = OFF  // Set watch dot timer off
#pragma config LVP = OFF  // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)
#pragma config PBADEN = OFF // Turn off PortB A/D
#pragma config OSC = INTIO67  //Internal frequency is 1MHz by default

#define _XTAL_FREQ 1000000  //used by __delay_ms
#define LCD_PWR PORTDbits.RD7
#define LCD_E PORTDbits.RD6
#define LCD_RS PORTDbits.RD4
#define LCD_RW PORTDbits.RD5
#define LCD_TXRX TRISD
#define LCD_IO_DATA PORTD
#define INPUT 1
#define OUTPUT 0
#define INPUT_ALL 0xFF
#define OUTPUT_ALL 0x00
#define ENABLE 1
#define DISABLE 0
#define READ 1
#define WRITE 0
#define I2C_MASTER   0b00001000 //I2C Master mode, clock = FOSC/(4*(SSPAD + 1)
#define I2C_M 8 //I2C Master mode, clock = FOSC/(4*(SSPAD + 1
#define I2C_M_E_CONF 0b00101000 //I2C Master mode, clock = FOSC/(4*(SSPAD + 1) and
SSPEN = 1
#define I2C_M 8
#define BAUD_100KHZ 0X09
#define WRITE_EEPROM 0b10100000
#define READ_EEPROM  0b10100001
#define SCROLL_SPEED 187
```

```
void i2c_idle(void);
void i2c_start(void);
void i2c_rstart(void);
void i2c_stop(void);
void i2c_ack(unsigned char ackbit);
unsigned char i2c_send(unsigned char data);
unsigned char i2c_receive(unsigned char ackbit);
void i2c_init(void);
void send_data(unsigned char data,unsigned char cflag,
            unsigned char chk_busy,unsigned char dflag);
void epulse(void);
void load_string(const char *pt);
int write_page(char *the_page, unsigned char msb_add, unsigned char lsb_add);
int write_byte(unsigned char data, unsigned char msb_add, unsigned char lsb_add);
int read_page(char *the_page, unsigned char msb_add, unsigned char lsb_add);
////////////////////////////////////////////////////////////////////////////
////////////////////////// End of Header ///////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

////Globals/////////////// Just because/////////////////
char myletter1 = 'z';
//const char mypage[64] = "ZSEM SHOULD be a page worth of information
aaaaaaaaaaaaaaaa\0";
//const char mypage[64] = "ABCDEFGHIJ KLMNOPQRST UVWXYZ!@#$ abcdefghij klmnopqrst
uvwxy63\0";
const char mypage[64] = "My message to you: Have a Merry Christmast and a Happy New
Year\0";
//////////////////////////

/**
 * The main file write to 24LC256 EEPROM on the PICDEM2 board and retrieves the
 * ASCII characters written and displays it on the LCD.
 */
void main (void){

    char myletter2 = 'z';
    char myletter3 = 'z';
    char message[64];
    i2c_init();                    // Send START condition
    write_page(mypage, 0x00, 0x00);
    read_page(message, 0x00, 0x00);
```

```
    LCD_TXRX = OUTPUT_ALL; //port direction write
    LCD_IO_DATA = 0x00; //port outputs 0
    LCD_PWR = ENABLE; //power to lcd
    send_data(0x20,1,0,0); //4-bit data
    send_data(0x20,1,0,1); //1 line display
    send_data(0x20,1,0,1); //1 line display
    send_data(0x06,1,0,1); //enable display set to incrementing mode
    send_data(0x0c,1,0,1); //turn on display
    send_data(0x02,1,0,1); //clear display


    send_data(0x01,1,0,1); //Move cursor to home position
    load_string(message);
    while (1)
    {

        send_data(0x18,1,1,1); //scroll display
        __delay_ms(SCROLL_SPEED); //Slow down the scroll
    }
}

////////////////////////////////////////////////////////////////////////////
// Could put everything below in to a seperate . c source file but placed it ///
// here for simplicity. ////////////////////////////////////////////////////////



int write_byte(unsigned char data, unsigned char msb_add, unsigned char lsb_add)
{
    i2c_start();
    while(i2c_send(WRITE_EEPROM))
    {
       i2c_rstart();
    }
    while(i2c_send(msb_add));
    while(i2c_send(lsb_add));
    while(i2c_send(data));
    i2c_stop();
}



/**
```

```
 * Used for Inter-Integrated Circuit (I2C) communication to check for when the
 * lines are idel
 */
void i2c_idle(void){
   while((SSPCON2 & 0x1F)| SSPSTATbits.R_NOT_W); //Check all flags for idle
   return;
}

/**
 * Used for Inter-Integrated Circuit (I2C) to have the micro-controller send a
 * start condition SDA (LOW to HIGH), SCL (HIGH).  This is a feature of the
 * PIC18F microcontroler i.e. SSCON2bits.SEN.
 */
void i2c_start(void){
        i2c_idle();             //wait for idle bus
        SSPCON2bits.SEN    = 1;     //initiate start
        while(SSPCON2bits.SEN); //and wait for start to finish
        return;
}

/**
 * Used for Inter-Integrated Circuit (I2C) to have the micro-controller send a
 * re-start condition SDA (LOW to HIGH), SCL (HIGH) with out issuing a stop.
 * This is a feature of the PIC18F microcontroler i.e. SSCON2bits.RSEN
 */
void i2c_rstart(void){
        i2c_idle();                     //wait for idle bus
        SSPCON2bits.RSEN=1;
        while(SSPCON2bits.RSEN); //and wait for restart to finish
        return;
}

/**
 * Used for Inter-Integrated Circuit (I2C) to have the micro-controller send a
 * stop condition SDA (HIGH to LOW), SCL (HIGH).  This is a feature of the
 * PIC18F microcontroler i.e. SSCON2bits.PEN
 */
void i2c_stop(void){
        i2c_idle();                 //wait for idle bus
        SSPCON2bits.PEN= 1;             //initiate stop
        while(SSPCON2bits.PEN);     //and wait for stop to finish
        return;
}
```

```
/**
 * Used for Inter-Integrated Circuit (I2C) to have the micro-controller send a
 * ACKNOWLEDGE or NO-ACKNOWLEDGE. stop condition SDA (HIGH to LOW), SCL (HIGH).
 * This is a feature of the PIC18F microcontroler i.e. SSCON2bits.ACKDT
 *
 * @param ackbit - 1 for acknowledge, 0 for no-acknowlege
 */
void i2c_ack(unsigned char ackbit){
    if ( ackbit )
    {
        SSPCON2bits.ACKDT=0; //set acknowledge bit
    }
    else
    {
        SSPCON2bits.ACKDT=1; //set no acknowledge bit
    }
    i2c_idle();
    ACKEN=1; //initiate acknowledgement
    i2c_idle(); //wait for ack to finish
    return;
}

/**
 * Used for Inter-Integrated Circuit (I2C) to have the micro-controller send
 * data to a slave device (client for the politically correct).
 *
 * @param data - the data to send
 */
unsigned char i2c_send(unsigned char data)
{
    i2c_idle();
    SSPBUF = data;          // write single byte to SSPBUF
    if ( SSPCON1bits.WCOL )     // test if write collision occurred
      return ( 1 );             // if WCOL bit is set return 1
    else
    {
      while( SSPSTATbits.BF );   // wait until write cycle is complete
      i2c_idle();                // ensure module is idle
      if ( SSPCON2bits.ACKSTAT ) // test for ACK condition received
        return ( 2 );                           // return NACK
      else return ( 0 );             //return ACK
    }
```

```
}

/**
 * Used for Inter-Integrated Circuit (I2C) to have the micro-controller recieve
 * data from a slave device (client for the politically correct).
 *
 * @param ackbit - 1 for acknowlege and 0 for no-acknowledge
 * @return - the recieved byte
 */
unsigned char i2c_receive(unsigned char ackbit){
    unsigned char data;
    i2c_idle();
    SSPCON2bits.RCEN = ENABLE;          // enable master for 1 byte reception
    while(SSPCON2bits.RCEN);
    while (!SSPSTATbits.BF);      // wait until byte received
    data = SSPBUF;
    i2c_ack(ackbit);
  return ( data );
}

/**
 * @breif Initializes the PIC18F micro-controller for Master I2C mode.
 *
 * TODO - should make this a macro since its only used once, makes the main func
 * bigger but should initialize faster (obsessed with speed!!!)
 */
void i2c_init(void){

    /*
     * Enable Master Synchronous Serial Port and configures SDA and SCL pins as
     * serial port pins for I2C protocol.
     *
     * SSPCON1 = I2C_M_E_CONF;  Equivalent to code below (faster)
     */
    SSPCON1bits.SSPEN = ENABLE;
    SSPCON1bits.SSPM = I2C_M;  //initialize I2C mode

    /*
     * This is not needed if used on the PICDEM2 Plus board but is good practice
     * as for manufacturer's recomendations
     */
    SSPSTATbits.CKE = ENABLE; //Enable SMBus specific inputs
    SSPSTATbits.SMP = ENABLE; //Disable maximum rate of change of output voltage per unit of
```

```
time

    /*
     * Set SDA and CLK as inputs on the micro-controller
     */
    TRISC3 = INPUT; //CLOCK
    TRISC4 = INPUT; //DATA

    SSPADD = BAUD_100KHZ; // 1MHZ /(4*( 0X09 + 1) = 100KHZ, SSPAD = 0X09
    return;
}

/**
 * This fucntions sends control data or ascii data to the LCD.
 * @param data The data to send to the LCD
 * @param cflag - a flag to indicate if the data sent is a command or not
 * @param chk_busy - 0 not checking if the LCD is busy, else it checks if the
 *                   LCD is busy
 * @param dflag - indicates if the lower nibble is sent if its not a control
 *                  signal
 */
void send_data(unsigned char data,unsigned char cflag,
unsigned char chk_busy, unsigned char dflag){
        char c,d,bflag;
        if(chk_busy)
        {
                LCD_TXRX = 0x0F; //read on lower 4 bits
                LCD_RS = 0; //RS=0, control signal
                LCD_RW = READ; //read
                bflag = 1; //set busy flag

                while(bflag)
                {
                        Nop();
                        LCD_E =1; //E high
                        Nop(); //__delay_us(1);
                        bflag = PORTDbits.RD3; //read busy signal
                        Nop();
                        LCD_E=0; //E low
                        epulse(); //read lower 4 bits
                }
        }
        else
```

```
        __delay_ms(500); // Wiat have a second

    LCD_TXRX = 0x00; //write from port
    if(cflag)
        LCD_RS = 0; //control signal
    else
        LCD_RS = 1; //data


    LCD_RW = WRITE; //write
    c = data>>4; //shift upper nibble to lower
    d = LCD_IO_DATA & 0xF0; //clear lower nibble of port
    LCD_IO_DATA = d | c; //add lower bits for data or command
    epulse();
    if (dflag){ //send lower nibble if not 4 bit control
        c = data & 0x0F; //clear upper nibble of data
        d = LCD_IO_DATA & 0xF0; //clear lower nibble of port
        LCD_IO_DATA = d | c; //send lower nibble
        epulse();
    }
return;
}

/**
 * This is a helper function for send_data that creates a single pulse on the
 * e (enable) pin of the LCD.
 */
void epulse(void){

    Nop(); //__delay_us(1);
    LCD_E=1; //E high
    Nop();
    LCD_E=0; //E low
    Nop();
    return;
}

/**
 * This function loads an ASCII string to the LCD.
 * @param ptr - the pointer to the string to display on the LCD
 */
void load_string(const char *ptr){
    while(*ptr != 0){ //while char != null
```

```
            send_data(*ptr,0,1,1); //send char
            ptr++; //increment pointer
        }
        return;
}


int write_page(char *the_page, unsigned char msb_add, unsigned char lsb_add)
{
    i2c_start();                    // Send START condition
    while(i2c_send(WRITE_EEPROM))
    {
      i2c_rstart();
    }
    i2c_send(msb_add);
    i2c_send(lsb_add);
    i2c_send(the_page[0]);
    i2c_send(the_page[1]);
    i2c_send(the_page[2]);
    i2c_send(the_page[3]);
    i2c_send(the_page[4]);
    i2c_send(the_page[5]);
    i2c_send(the_page[6]);
    i2c_send(the_page[7]);
    i2c_send(the_page[8]);
    i2c_send(the_page[9]);
    i2c_send(the_page[10]);
    i2c_send(the_page[11]);
    i2c_send(the_page[12]);
    i2c_send(the_page[13]);
    i2c_send(the_page[14]);
    i2c_send(the_page[15]);
    i2c_send(the_page[16]);
    i2c_send(the_page[17]);
    i2c_send(the_page[18]);
    i2c_send(the_page[19]);
    i2c_send(the_page[20]);
    i2c_send(the_page[21]);
    i2c_send(the_page[22]);
    i2c_send(the_page[23]);
    i2c_send(the_page[24]);
    i2c_send(the_page[25]);
    i2c_send(the_page[26]);
```

```
    i2c_send(the_page[27]);
    i2c_send(the_page[28]);
    i2c_send(the_page[29]);
    i2c_send(the_page[30]);
    i2c_send(the_page[31]);
    i2c_send(the_page[32]);
    i2c_send(the_page[33]);
    i2c_send(the_page[34]);
    i2c_send(the_page[35]);
    i2c_send(the_page[36]);
    i2c_send(the_page[37]);
    i2c_send(the_page[38]);
    i2c_send(the_page[39]);
    i2c_send(the_page[40]);
    i2c_send(the_page[41]);
    i2c_send(the_page[42]);
    i2c_send(the_page[43]);
    i2c_send(the_page[44]);
    i2c_send(the_page[45]);
    i2c_send(the_page[46]);
    i2c_send(the_page[47]);
    i2c_send(the_page[48]);
    i2c_send(the_page[49]);
    i2c_send(the_page[50]);
    i2c_send(the_page[51]);
    i2c_send(the_page[52]);
    i2c_send(the_page[53]);
    i2c_send(the_page[54]);
    i2c_send(the_page[55]);
    i2c_send(the_page[56]);
    i2c_send(the_page[57]);
    i2c_send(the_page[58]);
    i2c_send(the_page[59]);
    i2c_send(the_page[60]);
    i2c_send(the_page[61]);
    i2c_send(the_page[62]);
    i2c_send(the_page[63]);
    i2c_stop();
    return 0;
}

int read_page(char *message, unsigned char msb_add, unsigned char lsb_add)
{
```

```
i2c_start();
while(i2c_send(WRITE_EEPROM))
{
   i2c_rstart();
}
i2c_send(msb_add);
i2c_send(lsb_add);
i2c_rstart();
i2c_send(READ_EEPROM);
message[0] = i2c_receive(1);
message[1] = i2c_receive(1);
message[2] = i2c_receive(1);
message[3] = i2c_receive(1);
message[4] = i2c_receive(1);
message[5] = i2c_receive(1);
message[6] = i2c_receive(1);
message[7] = i2c_receive(1);
message[8] = i2c_receive(1);
message[9] = i2c_receive(1);
message[10] = i2c_receive(1);
message[11] = i2c_receive(1);
message[12] = i2c_receive(1);
message[13] = i2c_receive(1);
message[14] = i2c_receive(1);
message[15] = i2c_receive(1);
message[16] = i2c_receive(1);
message[17] = i2c_receive(1);
message[18] = i2c_receive(1);
message[19] = i2c_receive(1);
message[20] = i2c_receive(1);
message[21] = i2c_receive(1);
message[22] = i2c_receive(1);
message[23] = i2c_receive(1);
message[24] = i2c_receive(1);
message[25] = i2c_receive(1);
message[26] = i2c_receive(1);
message[27] = i2c_receive(1);
message[28] = i2c_receive(1);
message[29] = i2c_receive(1);
message[30] = i2c_receive(1);
message[31] = i2c_receive(1);
message[32] = i2c_receive(1);
message[33] = i2c_receive(1);
```

```
    message[34] = i2c_receive(1);
    message[35] = i2c_receive(1);
    message[36] = i2c_receive(1);
    message[37] = i2c_receive(1);
    message[38] = i2c_receive(1);
    message[39] = i2c_receive(1);
    message[40] = i2c_receive(1);
    message[41] = i2c_receive(1);
    message[42] = i2c_receive(1);
    message[43] = i2c_receive(1);
    message[44] = i2c_receive(1);
    message[45] = i2c_receive(1);
    message[46] = i2c_receive(1);
    message[47] = i2c_receive(1);
    message[48] = i2c_receive(1);
    message[49] = i2c_receive(1);
    message[50] = i2c_receive(1);
    message[51] = i2c_receive(1);
    message[52] = i2c_receive(1);
    message[53] = i2c_receive(1);
    message[54] = i2c_receive(1);
    message[55] = i2c_receive(1);
    message[56] = i2c_receive(1);
    message[57] = i2c_receive(1);
    message[58] = i2c_receive(1);
    message[59] = i2c_receive(1);
    message[60] = i2c_receive(1);
    message[61] = i2c_receive(1);
    message[62] = i2c_receive(1);
    message[63] = i2c_receive(0);
    i2c_idle();
    i2c_stop();
    return 0;
}
```