**I2C Lab**

**Videos with the following names are found in Dropbox:**

**Task 1 -** Lab10-T01.mov

**Task 2 -** Lab10-T02.mov

**Task 3 -** Lab10-T03.mov

**Task 1: Adding comments to original code.**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Project : Orbit Lab 7 ATE (ACCEL)

Version : 1.0

Date : 2/20/2013

Author : Brian Zufelt / Craig Kief

Company : COSMIAC/UNM

Comments:

This source provides an introduction to the sensing capabilities

for embedded systems. The student will read accelerometer data

and toggle the proper LED to provide tilt measurement

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Chip type : ARM TM4C123GH6PM

Program type : Firmware

Core Clock frequency : 80.000000 MHz

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define ACCEL\_W 0x3A // Addresses for the accelerometer

#define ACCEL\_R 0x3B

#define ACCEL\_ADDR 0x1D

// Define needed for pin\_map.h

#define PART\_TM4C123GH6PM

#include <stdbool.h>

#include <stdint.h>

#include "inc/tm4c123gh6pm.h"

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "inc/hw\_i2c.h"

#include "driverlib/gpio.h"

#include "driverlib/pin\_map.h"

#include "driverlib/sysctl.h"

#include "driverlib/i2c.h"

void Accel\_int(); // Function prototype to initialize the Accelerometer

signed int Accel\_read(); // Function prototype to read the Accelerometer

void main(void) {

signed short int LED\_value = 1;

SysCtlClockSet(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_OSC | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ); //setup clock

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_I2C0); // Enable I2C hardware

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOB); // Enable Pin hardware

GPIOPinConfigure(GPIO\_PB3\_I2C0SDA); // Configure GPIO pin for I2C Data line

GPIOPinConfigure(GPIO\_PB2\_I2C0SCL); // Configure GPIO Pin for I2C clock line

GPIOPinTypeI2C(GPIO\_PORTB\_BASE, GPIO\_PIN\_2 | GPIO\_PIN\_3); // Set Pin Type

// Enable Peripheral ports for output

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOC); //PORTC

GPIOPinTypeGPIOOutput(GPIO\_PORTC\_BASE, GPIO\_PIN\_6|GPIO\_PIN\_7); // LED 1 LED 2

GPIOPinTypeGPIOOutput(GPIO\_PORTB\_BASE, GPIO\_PIN\_5); // LED 4

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD); //PORT D

GPIOPinTypeGPIOOutput(GPIO\_PORTD\_BASE, GPIO\_PIN\_6); // LED 3

//setup the I2C

GPIOPadConfigSet(GPIO\_PORTB\_BASE, GPIO\_PIN\_2, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD); // SDA MUST BE STD

GPIOPadConfigSet(GPIO\_PORTB\_BASE, GPIO\_PIN\_3, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_OD); // SCL MUST BE OPEN DRAIN

I2CMasterInitExpClk(I2C0\_BASE, SysCtlClockGet(), false); // The False sets the controller to 100kHz communication

Accel\_int(); // Function to initialize the Accelerometer

while(1){

// Fill in this section to read data from the Accelerometer and move the LEDs according to the X axis

LED\_value = LED\_value + Accel\_read();

if(LED\_value <= 1){

// Cycle through the LEDs on the Orbit board

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6|GPIO\_PIN\_7, 0x40); // LED 1 on LED 2 Off

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0x00); // LED 3 off, Note different PORT

GPIOPinWrite(GPIO\_PORTB\_BASE, GPIO\_PIN\_5, 0x00); // LED 4 off

LED\_value = 1; // reset value to maintain range

}

else if(LED\_value == 2){

// Cycle through the LEDs on the Orbit board

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6|GPIO\_PIN\_7, 0x80); // LED 1 off LED 2 on

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0x00); // LED 3 off, Note different PORT

GPIOPinWrite(GPIO\_PORTB\_BASE, GPIO\_PIN\_5, 0x00); // LED 4 on

}

else if(LED\_value == 3){

// Cycle through the LEDs on the Orbit board

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6|GPIO\_PIN\_7, 0x00); // LED 1 off LED 2 off

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0x40); // LED 3 on, Note different PORT

GPIOPinWrite(GPIO\_PORTB\_BASE, GPIO\_PIN\_5, 0x00); // LED 4 0ff

}

else if(LED\_value >= 4){

// Cycle through the LEDs on the Orbit board

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6|GPIO\_PIN\_7, 0x00); // LED 1 off LED 2 Off

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0x00); // LED 3 off, Note different PORT

GPIOPinWrite(GPIO\_PORTB\_BASE, GPIO\_PIN\_5, 0x20); // LED 4 on

LED\_value = 4; // reset value to maintain range

}

}

}

void Accel\_int(){ // Function to initialize the Accelerometer

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, false); // false means transmit

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_START); // Send Start condition

I2CMasterDataPut(I2C0\_BASE, 0x2D); // Writing to the Accel control reg

SysCtlDelay(20000); // Delay for first transmission

I2CMasterDataPut(I2C0\_BASE, 0x08); // Send Value to control Register

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_FINISH); // Send Stop condition

while(I2CMasterBusBusy(I2C0\_BASE)){}; // Wait for I2C controller to finish operations

}

signed int Accel\_read() {// Function to read the Accelerometer

signed int data;

signed short value = 0; // value of x

unsigned char MSB;

unsigned char LSB;

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, false); // false means transmit

I2CMasterDataPut(I2C0\_BASE, 0x32);

SysCtlDelay(20000);

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_SEND); //Request LSB of X Axis

SysCtlDelay(2000000); // Delay for first transmission

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, true); // false means transmit

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE); //Request LSB of X Axis

SysCtlDelay(20000);

LSB = I2CMasterDataGet(I2C0\_BASE);

SysCtlDelay(20000);

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, false); // false means transmit

I2CMasterDataPut(I2C0\_BASE, 0x33);

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_SEND); //Request LSB of X Axis

SysCtlDelay(2000000); // Delay for first transmission

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, true); // false means transmit

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE); //Request LSB of X Axis

SysCtlDelay(20000);

MSB = I2CMasterDataGet(I2C0\_BASE);

value = (MSB << 8 | LSB);

if(value < -250 ){ // testing axis for value

data = -1;

}

else if (value > 250){

data = 1;

}

else{

data = 0;

}

SysCtlDelay(20000);

return data; // return value

}

**Task 2: Modify code to display X, Y, and Z values from accelerometer in UART.**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Project : Orbit Lab 7 ATE (ACCEL)

Version : 1.0

Date : 2/20/2013

Author : Brian Zufelt / Craig Kief

Company : COSMIAC/UNM

Comments:

This source provides an introduction to the sensing capabilities

for embedded systems. The student will read accelerometer data

and toggle the proper LED to provide tilt measurement

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Chip type : ARM TM4C123GH6PM

Program type : Firmware

Core Clock frequency : 80.000000 MHz

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define ACCEL\_W 0x3A // Addresses for the accelerometer

#define ACCEL\_R 0x3B

#define ACCEL\_ADDR 0x1D

// Define needed for pin\_map.h

#ifndef PART\_TM4C123GH6PM

#define PART\_TM4C123GH6PM

#endif

#include <stdbool.h>

#include <stdint.h>

#include "inc/tm4c123gh6pm.h"

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "inc/hw\_i2c.h"

#include "driverlib/gpio.h"

#include "driverlib/uart.h"

#include "driverlib/pin\_map.h"

#include "driverlib/sysctl.h"

#include "driverlib/i2c.h"

void Accel\_int(); // Function prototype to initialize the Accelerometer

signed short Accel\_readX(); // Function prototype to read the Accelerometer X

signed short Accel\_readY(); // Function prototype to read the Accelerometer Y

signed short Accel\_readZ(); // Function prototype to read the Accelerometer Z

void main(void) {

signed short int x, y, z;

signed short int LED\_value = 1;

signed short int hundreds, tens, ones;

SysCtlClockSet(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_OSC | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ); //setup clock

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_I2C0); // Enable I2C hardware

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOB); // Enable Pin hardware

GPIOPinConfigure(GPIO\_PB3\_I2C0SDA); // Configure GPIO pin for I2C Data line

GPIOPinConfigure(GPIO\_PB2\_I2C0SCL); // Configure GPIO Pin for I2C clock line

GPIOPinTypeI2C(GPIO\_PORTB\_BASE, GPIO\_PIN\_2 | GPIO\_PIN\_3); // Set Pin Type

// Enable Peripheral ports for output

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOC); //PORTC

GPIOPinTypeGPIOOutput(GPIO\_PORTC\_BASE, GPIO\_PIN\_6|GPIO\_PIN\_7); // LED 1 LED 2

GPIOPinTypeGPIOOutput(GPIO\_PORTB\_BASE, GPIO\_PIN\_5); // LED 4

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD); //PORT D

GPIOPinTypeGPIOOutput(GPIO\_PORTD\_BASE, GPIO\_PIN\_6); // LED 3

//Setup UART

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0); // Enable UART hardware

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA); // Enable Pin hardware

GPIOPinConfigure(GPIO\_PA0\_U0RX); // Configure GPIO pin for UART RX line

GPIOPinConfigure(GPIO\_PA1\_U0TX); // Configure GPIO Pin for UART TX line

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); // Set Pins for UART

UARTConfigSetExpClk(UART0\_BASE, SysCtlClockGet(), 115200, // Configure UART to 8N1 at 115200bps

(UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE | UART\_CONFIG\_PAR\_NONE));

//---------

//setup the I2C

GPIOPadConfigSet(GPIO\_PORTB\_BASE, GPIO\_PIN\_2, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD); // SDA MUST BE STD

GPIOPadConfigSet(GPIO\_PORTB\_BASE, GPIO\_PIN\_3, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_OD); // SCL MUST BE OPEN DRAIN

I2CMasterInitExpClk(I2C0\_BASE, SysCtlClockGet(), false); // The False sets the controller to 100kHz communication

Accel\_int(); // Function to initialize the Accelerometer

while(1){

//x axis

UARTCharPut(UART0\_BASE, 'X');

UARTCharPut(UART0\_BASE, ':');

UARTCharPut(UART0\_BASE, ' ');

x = Accel\_readX();

if(x >= 0)

neg = false;

else{

neg = true;

x = -x;

}

hundreds = x / 100 + 48;

tens = x % 100 / 10 + 48;

ones = x % 100 % 10 + 48;

if(!neg){

UARTCharPut(UART0\_BASE, ' ');

if(x >= 100)

UARTCharPut(UART0\_BASE, hundreds);

else

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, tens);

UARTCharPut(UART0\_BASE, ones);

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, ' ');

}

else{

if(x >= 100){

UARTCharPut(UART0\_BASE, '-');

UARTCharPut(UART0\_BASE, hundreds);

}

else{

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, '-');

}

UARTCharPut(UART0\_BASE, tens);

UARTCharPut(UART0\_BASE, ones);

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, ' ');

}

// y axis

UARTCharPut(UART0\_BASE, 'Y');

UARTCharPut(UART0\_BASE, ':');

UARTCharPut(UART0\_BASE, ' ');

y = Accel\_readY();

if(y >= 0)

neg = false;

else{

neg = true;

y = -y;

}

hundreds = y / 100 + 48;

tens = y % 100 / 10 + 48;

ones = y % 100 % 10 + 48;

if(!neg){

UARTCharPut(UART0\_BASE, ' ');

if(y >= 100)

UARTCharPut(UART0\_BASE, hundreds);

else

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, tens);

UARTCharPut(UART0\_BASE, ones);

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, ' ');

}

else{

if(y >= 100){

UARTCharPut(UART0\_BASE, '-');

UARTCharPut(UART0\_BASE, hundreds);

}

else{

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, '-');

}

UARTCharPut(UART0\_BASE, tens);

UARTCharPut(UART0\_BASE, ones);

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, ' ');

}

//z axis

UARTCharPut(UART0\_BASE, 'Z');

UARTCharPut(UART0\_BASE, ':');

UARTCharPut(UART0\_BASE, ' ');

z = Accel\_readZ();

if(z >= 0)

neg = false;

else{

neg = true;

z = -z;

}

hundreds = z / 100 + 48;

tens = z % 100 / 10 + 48;

ones = z % 100 % 10 + 48;

if(!neg){

UARTCharPut(UART0\_BASE, ' ');

if(z >= 100)

UARTCharPut(UART0\_BASE, hundreds);

else

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, tens);

UARTCharPut(UART0\_BASE, ones);

UARTCharPut(UART0\_BASE, '\n');

UARTCharPut(UART0\_BASE, '\r');

}

else{

if(z >= 100){

UARTCharPut(UART0\_BASE, '-');

UARTCharPut(UART0\_BASE, hundreds);

}

else{

UARTCharPut(UART0\_BASE, ' ');

UARTCharPut(UART0\_BASE, '-');

}

UARTCharPut(UART0\_BASE, tens);

UARTCharPut(UART0\_BASE, ones);

UARTCharPut(UART0\_BASE, '\n');

UARTCharPut(UART0\_BASE, '\r');

}

//----

/\*LED\_value = LED\_value + Accel\_readX();

if(LED\_value <= 1){

// Cycle through the LEDs on the Orbit board

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6|GPIO\_PIN\_7, 0x40); // LED 1 on LED 2 Off

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0x00); // LED 3 off, Note different PORT

GPIOPinWrite(GPIO\_PORTB\_BASE, GPIO\_PIN\_5, 0x00); // LED 4 off

LED\_value = 1; // reset value to maintain range

}

else if(LED\_value == 2){

// Cycle through the LEDs on the Orbit board

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6|GPIO\_PIN\_7, 0x80); // LED 1 off LED 2 on

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0x00); // LED 3 off, Note different PORT

GPIOPinWrite(GPIO\_PORTB\_BASE, GPIO\_PIN\_5, 0x00); // LED 4 on

}

else if(LED\_value == 3){

// Cycle through the LEDs on the Orbit board

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6|GPIO\_PIN\_7, 0x00); // LED 1 off LED 2 off

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0x40); // LED 3 on, Note different PORT

GPIOPinWrite(GPIO\_PORTB\_BASE, GPIO\_PIN\_5, 0x00); // LED 4 0ff

}

else if(LED\_value >= 4){

// Cycle through the LEDs on the Orbit board

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6|GPIO\_PIN\_7, 0x00); // LED 1 off LED 2 Off

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0x00); // LED 3 off, Note different PORT

GPIOPinWrite(GPIO\_PORTB\_BASE, GPIO\_PIN\_5, 0x20); // LED 4 on

LED\_value = 4; // reset value to maintain range

}

\*/

}

}

void Accel\_int(){ // Function to initialize the Accelerometer

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, false); // false means transmit

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_START); // Send Start condition

I2CMasterDataPut(I2C0\_BASE, 0x2D); // Writing to the Accel control reg

SysCtlDelay(20000); // Delay for first transmission

I2CMasterDataPut(I2C0\_BASE, 0x08); // Send Value to control Register

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_FINISH); // Send Stop condition

while(I2CMasterBusBusy(I2C0\_BASE)){}; // Wait for I2C controller to finish operations

}

signed short Accel\_readX() {// Function to read the Accelerometer

signed int data;

signed short value = 0; // value of x

unsigned char MSB;

unsigned char LSB;

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, false); // false means transmit

I2CMasterDataPut(I2C0\_BASE, 0x32);

SysCtlDelay(20000);

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_SEND); //Request LSB of X Axis

SysCtlDelay(2000000); // Delay for first transmission

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, true); // false means transmit

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE); //Request LSB of X Axis

SysCtlDelay(20000);

LSB = I2CMasterDataGet(I2C0\_BASE);

SysCtlDelay(20000);

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, false); // false means transmit

I2CMasterDataPut(I2C0\_BASE, 0x33);

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_SEND); //Request LSB of X Axis

SysCtlDelay(2000000); // Delay for first transmission

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, true); // false means transmit

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE); //Request LSB of X Axis

SysCtlDelay(20000);

MSB = I2CMasterDataGet(I2C0\_BASE);

value = (MSB << 8 | LSB);

if(value < -250 ){ // testing axis for value

value = -250;

}

else if (value > 250){

value = 250;

}

SysCtlDelay(20000);

return value; // return value

}

signed short Accel\_readY() {// Function to read the Accelerometer

signed int data;

signed short value = 0; // value of y

unsigned char MSB;

unsigned char LSB;

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, false); // false means transmit

I2CMasterDataPut(I2C0\_BASE, 0x34); //used for Y axis reading

SysCtlDelay(20000);

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_SEND); //Request LSB of X Axis

SysCtlDelay(2000000); // Delay for first transmission

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, true); // false means transmit

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE); //Request LSB of X Axis

SysCtlDelay(20000);

LSB = I2CMasterDataGet(I2C0\_BASE);

SysCtlDelay(20000);

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, false); // false means transmit

I2CMasterDataPut(I2C0\_BASE, 0x35); //used for Y axis reading

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_SEND); //Request LSB of X Axis

SysCtlDelay(2000000); // Delay for first transmission

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, true); // false means transmit

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE); //Request LSB of X Axis

SysCtlDelay(20000);

MSB = I2CMasterDataGet(I2C0\_BASE);

value = (MSB << 8 | LSB);

if(value < -250 ){ // testing axis for value

value = -250;

}

else if (value > 250){

value = 250;

}

SysCtlDelay(20000);

return value; // return value

}

signed short Accel\_readZ() {// Function to read the Accelerometer

signed int data;

signed short value = 0; // value of z

unsigned char MSB;

unsigned char LSB;

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, false); // false means transmit

I2CMasterDataPut(I2C0\_BASE, 0x36); //used for z axis reading

SysCtlDelay(20000);

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_SEND); //Request LSB of X Axis

SysCtlDelay(2000000); // Delay for first transmission

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, true); // false means transmit

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE); //Request LSB of X Axis

SysCtlDelay(20000);

LSB = I2CMasterDataGet(I2C0\_BASE);

SysCtlDelay(20000);

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, false); // false means transmit

I2CMasterDataPut(I2C0\_BASE, 0x37); //used for z axis reading

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_SEND); //Request LSB of X Axis

SysCtlDelay(2000000); // Delay for first transmission

I2CMasterSlaveAddrSet(I2C0\_BASE, ACCEL\_ADDR, true); // false means transmit

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE); //Request LSB of X Axis

SysCtlDelay(20000);

MSB = I2CMasterDataGet(I2C0\_BASE);

value = (MSB << 8 | LSB);

if(value < -250 ){ // testing axis for value

value = -250;

}

else if (value > 250){

value = 250;

}

SysCtlDelay(20000);

return data; // return value

}

**Task 3: Modify code to display temperature values and use LEDs to show status. Blue is for cold (below 30 deg C) and red is for hot (above 30 deg C).**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Project : Orbit Lab 8 ATE (Temp With UART)

Version : 1.0

Date : 2/20/2013

Author : Brian Zufelt / Craig Kief

Company : COSMIAC/UNM

Comments:

This lab will extend the concepts from LAB 7. This Lab

will pull data from the temperature sensor found on the

Orbit board and output the data through the UART to be

read from a terminal program.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Chip type : ARM TM4C123GH6PM

Program type : Firmware

Core Clock frequency : 80.000000 MHz

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#define** TEMP\_ADDR 0x4F // Address for Temp Sensor

// Define needed for pin\_map.h

**#ifndef** PART\_TM4C123GH6PM

**#define** PART\_TM4C123GH6PM

**#endif**

**#include** <stdbool.h>

**#include** <stdint.h>

**#include** "inc/tm4c123gh6pm.h"

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/pin\_map.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/uart.h"

**#include** "inc/hw\_i2c.h"

**#include** "driverlib/i2c.h"

**unsigned** **char** start\_screen[29] = "\n\n\r ATE Lab 8 Temp Sensor \n\n\r";

**unsigned** **char** log[18] = "\n\n\r Temp reading: ";

**void** **Print\_header**(); // Prints Header

**void** **Read\_temp**(**unsigned** **char** \*data); // Read Temperature sensor

**void** **main**(**void**) {

**unsigned** **char** temp\_data[10] = "00.0 C \n\n\r"; // Temp format to be edited by read

**unsigned** **short** **int** i = 0; // general counter

// Setup the I2C see lab 7 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SysCtlClockSet(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_OSC | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ); //setup clock

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_I2C0); // Enable I2C hardware

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOB); // Enable Pin hardware

GPIOPinConfigure(GPIO\_PB3\_I2C0SDA); // Configure GPIO pin for I2C Data line

GPIOPinConfigure(GPIO\_PB2\_I2C0SCL); // Configure GPIO Pin for I2C clock line

GPIOPinTypeI2C(GPIO\_PORTB\_BASE, GPIO\_PIN\_2 | GPIO\_PIN\_3); // Set Pin Type

GPIOPadConfigSet(GPIO\_PORTB\_BASE, GPIO\_PIN\_2, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD); // SDA MUST BE STD

GPIOPadConfigSet(GPIO\_PORTB\_BASE, GPIO\_PIN\_3, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_OD); // SCL MUST BE OPEN DRAIN

I2CMasterInitExpClk(I2C0\_BASE, SysCtlClockGet(), false); // The False sets the controller to 100kHz communication

I2CMasterSlaveAddrSet(I2C0\_BASE, TEMP\_ADDR, true); // false means transmit

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//configure Port F for LEDs

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF); //enable port F

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3); //set output pins on port F for the LEDs

// Setup the UART see lab 6 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0); // Enable UART hardware

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA); // Enable Pin hardware

GPIOPinConfigure(GPIO\_PA0\_U0RX); // Configure GPIO pin for UART RX line

GPIOPinConfigure(GPIO\_PA1\_U0TX); // Configure GPIO Pin for UART TX line

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); // Set Pins for UART

UARTConfigSetExpClk(UART0\_BASE, SysCtlClockGet(), 115200, // Configure UART to 8N1 at 115200bps

(UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE | UART\_CONFIG\_PAR\_NONE));

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Print\_header**(); // Print Header

**while**(1){

**Read\_temp**(temp\_data); // Read Data from Temp Sensor

**if**(temp\_data[0] < '3') //turn LED blue with temperatures <30 deg and red for temperatures >30

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, 4); //turn on blue LED

**else**

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, 2); //turn on red LED

SysCtlDelay(6000000); // Delay

**for**(i=0;i<10;i++){ // Loop to print out data string

UARTCharPut(UART0\_BASE, temp\_data[i]);

}

}

}

**void** **Print\_header**(){ // Print Header at start of program

**int** i = 0; // general counter

**for**(i=0;i<29;i++){ // Print Header at start of program

UARTCharPut(UART0\_BASE, start\_screen[i]);

}

}

**void** **Read\_temp**(**unsigned** **char** \*data){ // Read Temperature sensor

**unsigned** **char** temp[2]; // storage for data

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_START); // Start condition

SysCtlDelay(20000); // Delay

temp[0] = I2CMasterDataGet(I2C0\_BASE); // Read first char

SysCtlDelay(20000); // Delay

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_CONT); // Push second Char

SysCtlDelay(20000); // Delay

temp[1] = I2CMasterDataGet(I2C0\_BASE); // Read second char

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_RECEIVE\_FINISH); // Stop Condition

data[0] = (temp[0] / 10) + 0x30; // convert 10 place to ASCII

data[1] = (temp[0] - ((temp[0] / 10)\*10)) + 0x30; // Convert 1's place to ASCII

**if**(temp[1] == 0x80){ // Test for .5 accuracy

data[3] = 0x35;

}

**else**{

data[3] = 0x30;

}

}