

TI ARM Lab 9 Final Project



National
Science
Foundation

Funded in part, by a grant from the
National Science Foundation
DUE 1068182

Acknowledgements

Developed by Craig Kief, Brian Zufelt, and Jacy Bitsoie at the Configurable Space Microsystems Innovations & Applications Center (COSMIAC). Co-Developers are Bassam Matar from Chandler-Gilbert and Karl Henry from Drake State. *Funded by the National Science Foundation (NSF).*

Lab Summary

This lab introduces the concepts of the UART and how it is implemented on the ARM processor.

Lab Goal

The goal of this lab is to continue to build upon the skills learned from previous labs. This lab helps the student to continue to gain new skills and insight on the C code syntax and how it is used in the TI implementation of the ARM processor. Each of these labs will add upon the previous labs and it is the intention of the authors that students will build with each lab a better understanding of the ARM processor and basic C code, syntax and the new pieces of hardware that make up this system. Even though these tutorials assume the student has not entered with a knowledge of C code, it is the desire that by the time the student completes the entire series of tutorials that they will have a sufficient knowledge of C code so as to be able to accomplish useful projects on the ARM processor.

Learning Objectives

The student should begin to

Grading Criteria

N/A

Time Required

Approximately one hour

Lab Preparation

It is highly recommended that the student read through this entire procedure once before actually using it as a tutorial. It is also recommended that the tutorial software was run first to preload compiler options and source files as well as to load many of the main.c files.

Equipment and Materials

It is assumed that the student has already completed prior labs and the software is installed properly.

Software needed	Quantity
Install the tutorial framework from the autoinstaller located at http://cosmiac.org/thrust-areas/academic-programs-and-design-services/education-and-workforce-development/community-portal/ . The designer will also want Putty or similar RS-232 terminal program for viewing the UART output.	1
Hardware needed	Quantity
The hardware required is the TI Tiva LaunchPad Kit and the Digilent Orbit board	1



Additional References

Current TI ARM manuals found on TI web site: www.ti.com/lit/ug/spmu289a/spmu289a.pdf .
COSMIAC tutorials found at: <http://cosmiac.org/thrust-areas/academic-programs-and-design-services/education-and-workforce-development/microcontrollers/ate-developed-material/>



Lab Procedure

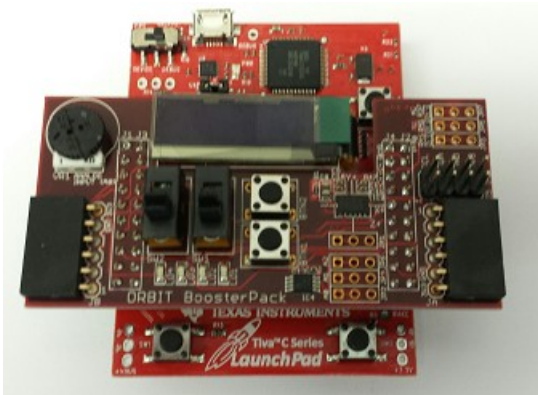


Figure 1. ARM and ORBIT Combination

This picture of the correct way to mate the Tiva LaunchPad and the Digilent Orbit boards together. Please do so at this point and connect them as shown in Figure 1.



Figure 2. Code Composer Icon

Launch Code Composer and where prompted, chose the workspaces location to store your project (as shown in Figure 3).

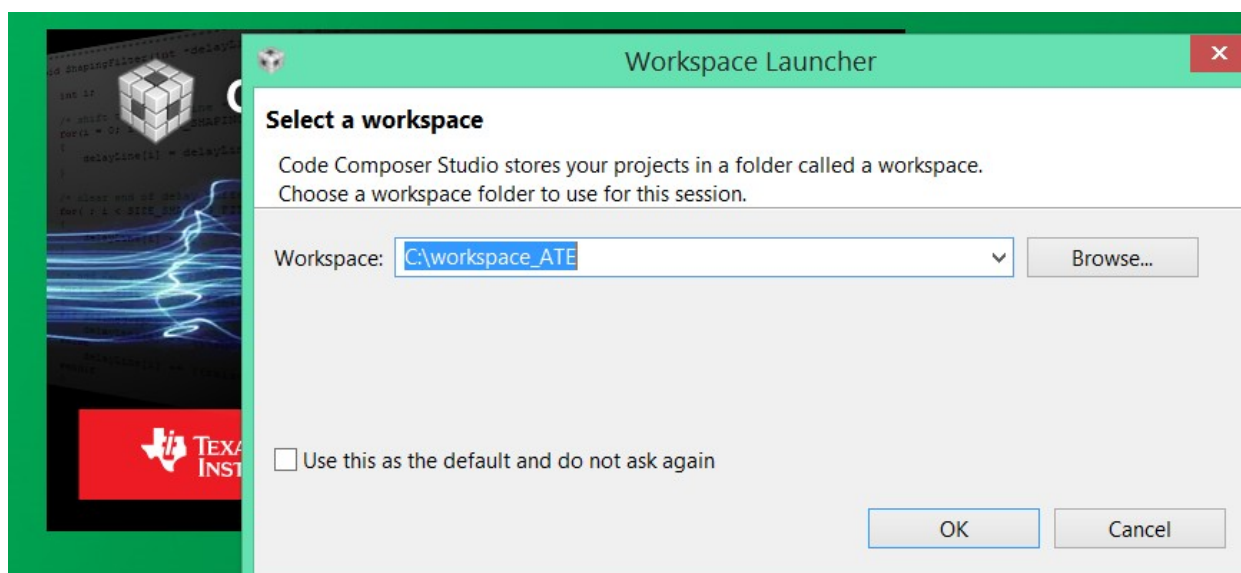


Figure 3. Workspace Selection

Since the installer for the workshop has been run prior to this, the user will be presented with the following view (Figure 4) where all lab projects exist.

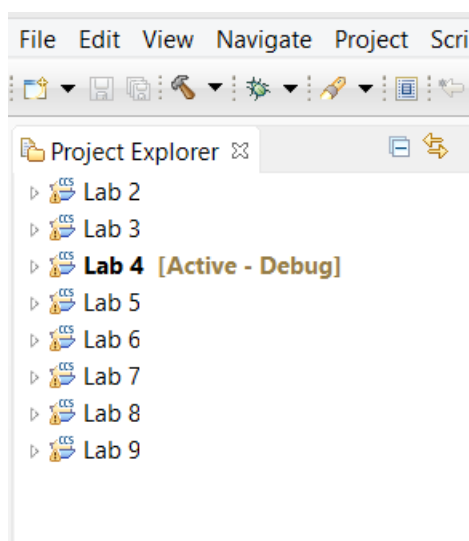


Figure 4. CCS Starting Point

The laboratory material is created to have the students type in a lot of the code. Only by typing the code and then debugging the errors will a user ever really understand how to do projects. For the sake of this activity, the source code is provided at the end of the tutorial. In Lab 9, open main.c. Then either type in all the code from attachment 2 or copy and paste in the code from Attachment 2 into main.c. Once again, it is easy to just copy and paste in all the code but what is learned is very minimal. The true power is typing in the code and dealing with the errors that you will create.

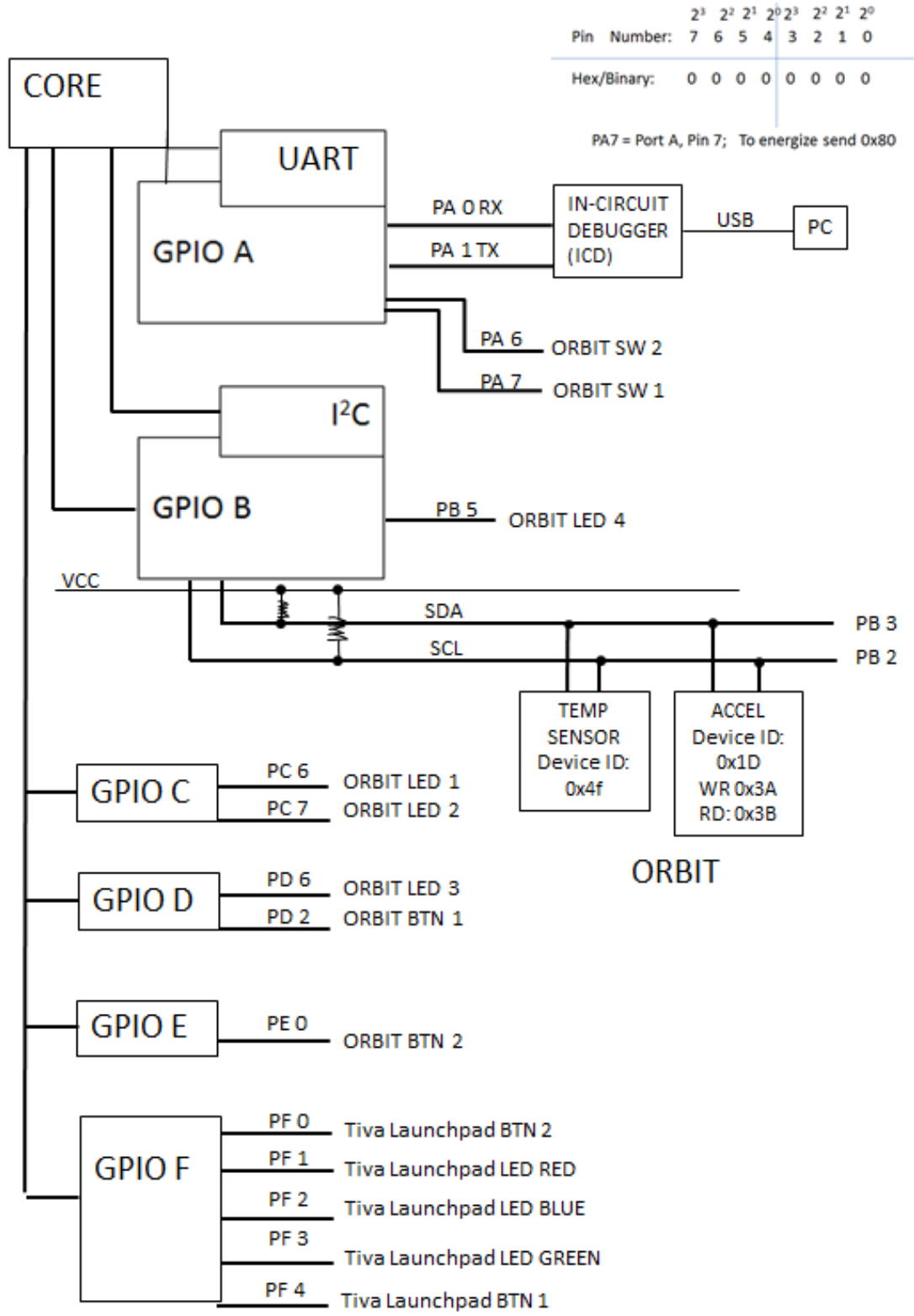


Figure 5. Graphic of Final Project



The block diagram in Figure 5 is a pictorial view of the final project. The accelerometer is shown on this block diagram but is not implemented in this project. It is added to help the student understand how countless items can be hung from the I²C bus and the user only need to know the address of the device to communicate with them.

```
#define TEMP_ADDR 0x4F           // Address for Temp Sensor

// 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 "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"
```

This first portion should seem very familiar. We are declaring some global variables for the address on the temperature sensor and also for the specific Tiva chip we are using. The program also has an array created for some character outputting through the UART.

```
unsigned char start_screen[31] = "\n\n\r ATE Lab 9 Final Project \n\n\r";
unsigned char log[18] = "\n\n\r Temp reading: ";
unsigned char start_temp = 0x00;           // starting temp
```

Next there are three function prototypes. Once again, these are identified at the top of the project but actually created at the end of the file. It is much cleaner that way.

```
void Print_header();           // Prints Header
unsigned short int Read_temp(unsigned char *data); // Read Temperature sensor
void Set_Temp_Start();        // Set starting value

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
    unsigned short int LED_value = 0;               // LED Power switch
```

The main program is created. The first thing that is accomplished is the assignment of three local variables used for temperature, counting and LEDs. Following this is the I²C code. There is nothing here different from before in the previous lab. This is the type of code you would use for every part that is in your project that hangs off the I²C bus (where each individual item is addressed through its specific unique address). Enable the I²C, enable the port, and then configure the pins (PB 2 (port B pin 2) and PB 3).



```
// 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
//*****

// 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));
//*****
```

Next is setting up the UART. This is very similar to everything done before. Enable the UART, enable the port then enable the two pins for transmit and receive: PA 0 and PA 1.

```
// Setup the LEDs see lab 2-5
*****
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC); //PORTC
GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7); // LED 1 LED 2

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB); //PORTB
GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5); // LED 4

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); //PORT D
GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_6); // LED 3
```

This is where the students begin typing. You must type in the section to enable and use the LEDs. There are four pins: PC 6, PC 7, PB 5, and PD 6. If you look at the figure in the attachment 3, it is easy to see where the pins attach. At this point though, all that is being done is enabling the ports and enabling the pins.

```
Print_header(); // Print Header
Set_Temp_Start();

while(1){
```



```

LED_value = Read_temp(temp_data); // Read Data from Temp Sensor
SysCtlDelay(6000000);           // Delay
for(i=0;i<10;i++){              // Loop to print out data string
    UARTCharPut(UART0_BASE, temp_data[i]);
}

if(LED_value == 1){

    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 ON

    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);           // LED 4 ON
}

else if (LED_value == 2){

    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0xFF); // LED 1 ON, LED 2 ON

    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00);           // LED 3 OFF

    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);           // LED 4 OFF
}

else if (LED_value == 3){

    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0xFF); // LED 1 ON, LED 2 ON

    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0xFF);           // LED 3 ON

    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);           // LED 4 OFF
}

else if(LED_value == 4){

    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0xFF); // LED 1 ON, LED 2 ON

    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0xFF);           // LED 3 ON

    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20);           // LED 4 ON
}

}

```

The section shown above starts with calling of the two functions that were prototyped and that will be spelled out after the main program. One of these functions is `start_temp`. All this does is run once. It senses the current room temperature and stores it so that the LEDs will have a starting place to begin from. The students begin to type again. In this case, what is required is typing in the portion that turns on and off the LEDs based on the “read-temp” function. This function will return an integer that is the value 1, 2, 3 or 4. Remember that the function is constantly running so this value will change often. However, based on the value, you will be able to see the LEDs change. The student must type in the LED portion. This is done to develop experience in clearing all the errors that will be created.



```

void Print_header(){                                     // Print Header at start of program

    int i = 0; // general counter

    for(i=0;i<31;i++){ // Print Header at start of program
        UARTCharPut(UART0_BASE, start_screen[i]);
    }
}

```

Next is the instantiation of the print_header function that was prototyped earlier. All this little bit of code does is print "\n\n\r ATE Lab 9 Final Project \n\n\r" which means new line a couple of times, carriage return, some pretty text, a couple of new lines and a carriage return. New line just brings you down a line. Carriage return brings you back to the start of the line. A little bit of experimentation on this will help you to better understand the formatting.

```

unsigned short int Read_temp(unsigned char *data){       // Read Temperature sensor

    unsigned char temp[2];                               // storage for data
    signed int temp_diff = 0;                            // difference between start and new data

    I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START); // Start
condition
    SysCtlDelay(20000);                                 // Delay
    temp[0] = I2CMasterDataGet(I2C0_MASTER_BASE);      // Read first char
    SysCtlDelay(20000);                                 // Delay
    I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT); // Push
second Char
    SysCtlDelay(20000);                                 // Delay
    temp[1] = I2CMasterDataGet(I2C0_MASTER_BASE);      // Read second char
    I2CMasterControl(I2C0_MASTER_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;
    }

    temp_diff = temp[0] - start_temp; // get difference between new data and boot

    if(temp_diff <= 1){                                // return LED switch value
        return 1;
    }

    else if ((temp_diff > 1) && (temp_diff <= 2)){
        return 2;
    }
}

```



```

    else if ((temp_diff > 2) && (temp_diff <= 3)){
        return 3;
    }
    else if(temp_diff > 3){
        return 4;
    }
}

```

Here is the instantiation of the read_temp function. It creates a couple of local variables. Next, it reads in the data off the bus. Upon reading it in, it does some comparisons and based on the temperature values (read and the initial value) calculated by the start_temp function, it spits out a 1, 2, 3 or 4. We will use these values to drive LEDs 1, 2, 3 and 4 back and forth.

```

void Set_Temp_Start(){
    unsigned char temp[2];                // storage for data

    I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START); // Start
condition
    SysCtlDelay(20000);                    // Delay
    temp[0] = I2CMasterDataGet(I2C0_MASTER_BASE); // Read first char
    SysCtlDelay(20000);                    // Delay
    I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT); //
Push second Char
    SysCtlDelay(20000);                    // Delay
    temp[1] = I2CMasterDataGet(I2C0_MASTER_BASE); // Read second char
    I2CMasterControl(I2C0_MASTER_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH); // Stop
Condition

    start_temp = temp[0];                // assign starting value
}

```

Here is the final function instantiation. It only runs once. Its only purpose is to generate a starting temperature value. This allows the LEDs to function properly in an air-conditioned room or not. It creates a starting temperature so that when the student puts their fingers on the sensor, they can see the LEDs change.

Debug and run the program. You should exit debug mode to allow for watching on the UART. Remember to not hold onto the sensor when the program begins as that is when the starting temperature is set.

Challenge – Change the text that is displayed, remove the decimal point, turn on the Tiva LEDs at certain times.



Attachment 1: main.c file (unfinished file)

```

/*****
Project : Orbit Lab 9 ATE ( Final Project)
Version : 1.0
Date   : 2/20/2013
Author  : Brian Zufelt / Craig Kief
Company : COSMIAC/UNM
Comments:
The final project adds to Lab 8. Students will add what
was learned in Labs 2-5 to add an LED data bar to show
how the temprature changes as you apply heat (eg. From a finger)

*****/
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
#define PART_TM4C123GH6PM

#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"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"

unsigned char start_screen[31] = "\n\n\r ATE Lab 9 Final Project \n\n\r";
unsigned char log[18] = "\n\n\r Temp reading: ";
unsigned char start_temp = 0x00;          // starting temp

void Print_header();
unsigned short int Read_temp(unsigned char *data); // Read Temperature sensor           // Prints Header
void Set_Temp_Start();                       // Set starting value

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
    unsigned short int LED_value = 0;               // LED Power switch

    // 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
    *****/
}

```



```

// 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));
//*****
*****

// Setup the LEDs see lab 2-5
*****
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC); //PORTC
GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7); // LED 1 LED 2

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB); //PORTB
GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5); // LED 4

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); //PORT D
GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_6); // LED 3
//*****
*****

Print_header(); // Print Header
Set_Temp_Start();

while(1){

    LED_value = Read_temp(temp_data); // Read Data from Temp Sensor
    SysCtlDelay(6000000); // Delay
    for(i=0;i<10;i++){ // Loop to print out data string
        UARTCharPut(UART0_BASE, temp_data[i]);
    }

    /******
    * Using the Return value from Read_temp() use a series of if/else statements
    * to toggle the LEDs to create a status bar for the new temperature readings
    * *****/

}

}

void Print_header(){ // Print Header at start of program

    int i = 0; // general counter

    for(i=0;i<31;i++){ // Print Header at start of program
        UARTCharPut(UART0_BASE, start_screen[i]);
    }
}

unsigned short int Read_temp(unsigned char *data){ // Read Temperature sensor

    unsigned char temp[2]; // storage for data
    signed int temp_diff = 0; // difference between start and new data

    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START); // Start condition
    SysCtlDelay(20000);
    // Delay
    char temp[0] = I2CMasterDataGet(I2C0_BASE); // Read first

    SysCtlDelay(20000);
    // Delay
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT); // Push second Char
    SysCtlDelay(20000);
    // Delay
    char temp[1] = I2CMasterDataGet(I2C0_BASE); // Read second

    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;
    }

temp_diff = temp[0] - start_temp; // get difference between new data and boot
value
    if(temp_diff <= 1){ // return LED switch
        return 1;
    }

    else if ((temp_diff > 1) && (temp_diff <= 2)){
        return 2;
    }

    else if ((temp_diff > 2) && (temp_diff <= 3)){
        return 3;
    }
    else if(temp_diff > 3){
        return 4;
    }
}

void Set_Temp_Start(){
    unsigned char temp[2]; // storage for data

    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START); // Start condition
    SysCtlDelay(20000);
    // Delay
    char temp[0] = I2CMasterDataGet(I2C0_BASE); // Read first
    SysCtlDelay(20000);
    // Delay
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT); // Push second Char
    SysCtlDelay(20000);
    // Delay
    char temp[1] = I2CMasterDataGet(I2C0_BASE); // Read second
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH); // Stop Condition

    start_temp = temp[0];
    // assign starting value
}

```



Attachment 2: main.c file (Solution)

```

/*****
Project : Orbit Lab 9 ATE ( Final Project)
Version : 1.0
Date   : 2/20/2013
Author  : Brian Zufelt / Craig Kief
Company : COSMIAC/UNM
Comments:
The final project adds to Lab 8. Students will add what
was learned in Labs 2-5 to add an LED data bar to show
how the temprature changes as you apply heat (eg. From a finger)

*****/
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
#define PART_TM4C123GH6PM

#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"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"

unsigned char start_screen[31] = "\n\n\r ATE Lab 9 Final Project \n\n\r";
unsigned char log[18] = "\n\n\r Temp reading: ";
unsigned char start_temp = 0x00;          // starting temp

void Print_header();
unsigned short int Read_temp(unsigned char *data); // Read Temperature sensor           // Prints Header
void Set_Temp_Start();                       // Set starting value

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
    unsigned short int LED_value = 0;                // LED Power switch

    // 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
    *****/
}

```



```

// 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));
//*****
*****

// Setup the LEDs see lab 2-5
*****
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC); //PORTC
GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7); // LED 1 LED 2

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB); //PORTB
GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5); // LED 4

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); //PORT D
GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_6); // LED 3
//*****
*****

Print_header(); // Print Header
Set_Temp_Start();

while(1){

    LED_value = Read_temp(temp_data); // Read Data from Temp Sensor
    SysCtlDelay(6000000); // Delay
    for(i=0;i<10;i++){ // Loop to print out data string
        UARTCharPut(UART0_BASE, temp_data[i]);
    }

    if(LED_value == 1){
        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 ON
        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00); // LED 4 ON
    }

    else if (LED_value == 2){
        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0xFF); // LED 1 ON, LED 2 ON
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 OFF
        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00); // LED 4 OFF
    }

    else if (LED_value == 3){
        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0xFF); // LED 1 ON, LED 2 ON
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0xFF); // LED 3 ON
        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00); // LED 4 OFF
    }

    else if(LED_value == 4){
        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0xFF); // LED 1 ON, LED 2 ON
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0xFF); // LED 3 ON
        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20); // LED 4 ON
    }

}

}

void Print_header(){ // Print Header at start of program

    int i = 0; // general counter

    for(i=0;i<31;i++){ // Print Header at start of program
        UARTCharPut(UART0_BASE, start_screen[i]);
    }

}

```



```

unsigned short int Read_temp(unsigned char *data){ // Read Temperature sensor

    unsigned char temp[2]; // storage for data
    signed int temp_diff = 0; // difference between start and new data

    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START); // Start condition
    SysCtlDelay(20000);
    // Delay
    char temp[0] = I2CMasterDataGet(I2C0_BASE); // Read first
    SysCtlDelay(20000);
    // Delay
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT); // Push second Char
    SysCtlDelay(20000);
    // Delay
    char temp[1] = I2CMasterDataGet(I2C0_BASE); // Read second
    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;
    }

    temp_diff = temp[0] - start_temp; // get difference between new data and boot
    if(temp_diff <= 1){ // return LED switch
        value
        return 1;
    }

    else if ((temp_diff > 1) && (temp_diff <= 2)){
        return 2;
    }

    else if ((temp_diff > 2) && (temp_diff <= 3)){
        return 3;
    }
    else if(temp_diff > 3){
        return 4;
    }
}

void Set_Temp_Start(){

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

    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START); // Start condition
    SysCtlDelay(20000);
    // Delay
    char temp[0] = I2CMasterDataGet(I2C0_BASE); // Read first
    SysCtlDelay(20000);
    // Delay
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT); // Push second Char
    SysCtlDelay(20000);
    // Delay
    char temp[1] = I2CMasterDataGet(I2C0_BASE); // Read second
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH); // Stop Condition

    start_temp = temp[0];
    // assign starting value
}

```




Attachment 3: Block Diagram of the Pins Used in Projects

