TI ARM Lab 7
Temperature Sensor

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 sampling and data passing 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 labs (or 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 become familiar with the concept of the temperature sensor and ways to continue to accomplish simple projects. One idea could be an entertainment center in your home. It might have been poorly designed. As the super sharp designer, you might want to attach a fan but only have it come on when the temperature is really hot. Your handy ARM processor could be used to do just that. Microchip Technology Inc.'s TCN75A digital temperature sensor (like the one on the ORBIT board) converts temperatures between -40°C and +125°C to a digital word, with ±.1°C (typical) accuracy. The TCN75A product comes with user-programmable registers that provide flexibility for temperature-sensing applications. The chip is the small eight pin device on the Orbit board that has an IC4 under it. Some typical Applications include:

- Personal Computers and Servers
- Hard Disk Drives and Other PC Peripherals
- Entertainment Systems
- Office Equipment
- Data Communication Equipment
- General Purpose Temperature Monitoring

The key to remember is that I^2C is nothing new. It was used in Lab 7 for the accelerometer. I^2C is an amazing bus architecture that all individuals that work with microcontrollers or microprocessors need to have a working knowledge of. It is possible to hang hundreds of small sensors off the same pair of wires and to address them individually only by the use of their unique address (shown in their datasheets).

This lab involves introductions to the temperature sensor and then the utilization of the UART to display the temperature on the computer.

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
The Evaluation Board user's manual is on this web site: http://datasheet.octopart.com/EK-TM4C123GXL-Texas-Instruments-datasheet-15542121.pdf and the manuals for the Digilent orbit board are located at http://digilentinc.com/Products/Detail.cfm?NavPath=2,396,1181&Prod=ORBIT-BOOSTER. Here is the datasheet for the temperature sensor http://ww1.microchip.com/downloads/en/DeviceDoc/21935a.pdf . Here is the place to find and download putty www.putty.org.

COSMIAC tutorials found at: http://cosmiac.org/thrust-areas/academic-programs-and-design-services/education-and-workforce-development/microcontrollers/ate-developed-material/

Lab Procedure

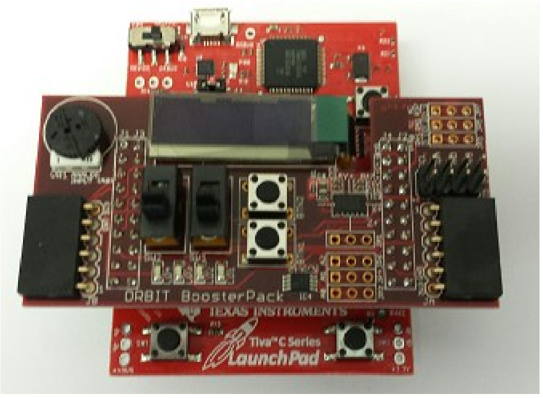


Figure 1. ARM and ORBIT Combination

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).

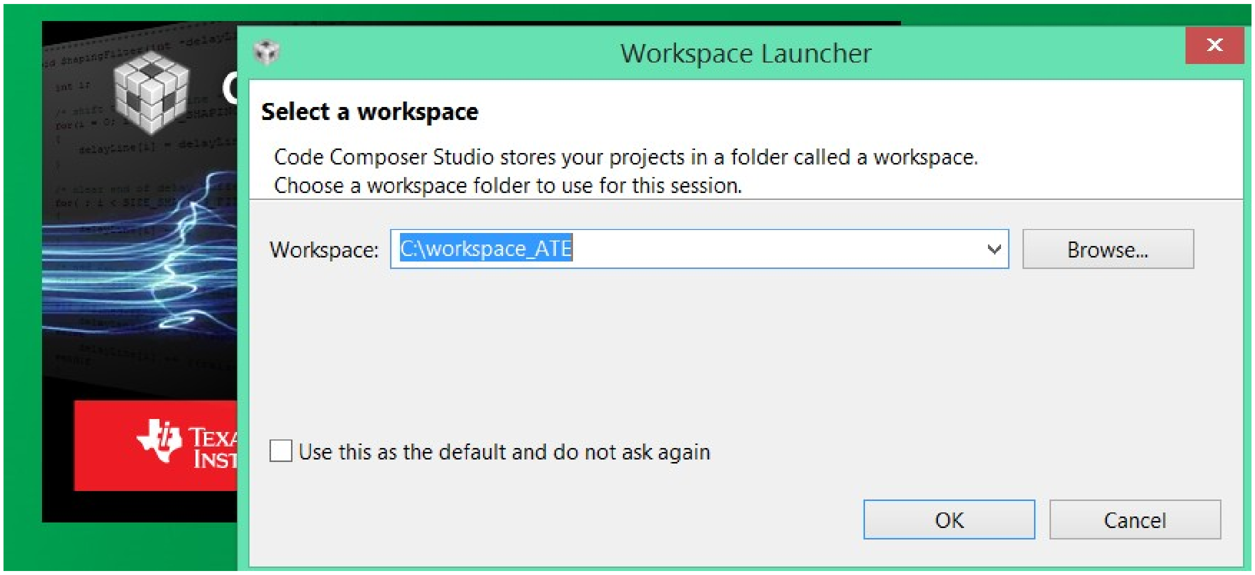


Figure 3. Workspace Selection

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.

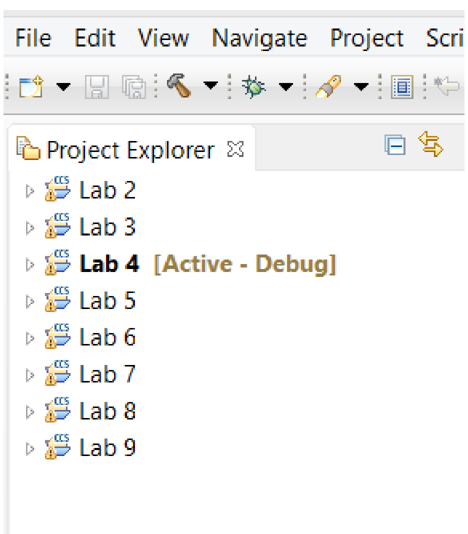


Figure 4. CCS Starting Point

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 8, open main.c. Then either type in all the code from attachment 2 or copy and paste in the code from Attachement 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.

Just as with the accelerometer, the temperature sensor has an I^2C interface. Remember that it is possible to hang hundreds of items onto this bus and only deal with them individually through their unique addresses. This is shown in Figure 5 and is very similar to the one showed in Tutorial 7. I^2C has two wires: SDA and SCL.

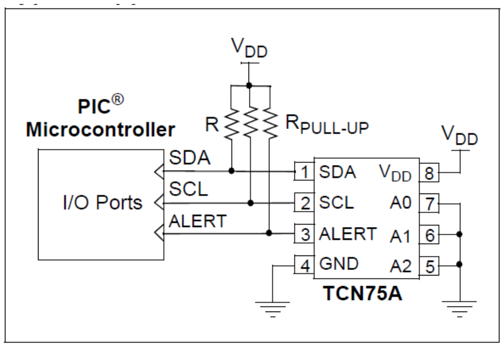


Figure 5. Temperature Sensor Overview

Many parts can share the I^2C bus. Think of it as many grapes hanging on the grapevine. The difference is that you must have a scheme for knowing when each of the pieces is communicated with. This is done with addresses. Each device has a different 7 bit address which is how it is identified. In the case of this temperature sensor, the address is shown below in Figure 6.

3.6 Address Pins (A2, A1, A0)

A2, A1 and A0 are device or slave address input pins.

The address pins are the Least Significant bits (LSb) of the device address bits. The Most Significant bits (MSb) (A6, A5, A4, A3) are factory-set to <1001>. This is illustrated in Table 3-2.

TABLE 3-2: SLAVE ADDRESS

Device	A6	A5	A4	A3	A2	A1	A0
TCN75A	1	0	0	1	X	X	X

Note: User-selectable address is shown by X.

Figure 6. Temperature Sensor Datasheet Excerpt

The portion of the datasheet copied for Figure 6 shows the address for connecting the temperature sensor to the system. The very first line of code is a define: **#define** TEMP_ADDR 0x4F. This is how you create the address in the project to link the I^2C pieces to the main project.

This is also a good place to introduce “global” versus “local” variables. In the code below, there are four variables declared. The first two are character (char) strings: start_screen and log. The [29] designates them as a character string that is 29 bytes long. The first two are done before and outside of the main function that runs the project. These are called global variables and can be seen by any function in the project.   The second two are declared within the main function: temp_data and i. These are local variables and can only be seen from within the main function. From a design perspective, always use local variables when possible. It will reduce the debugging time later.

```
unsigned char start_screen[29] = "\n\n\r ATE Lab 8 Temp Sensor \n\n\r";  //29 bytes long
unsigned char log[18] = "\n\n\r Temp reading: ";  //18 bytes long

//two function headers that will be defined later
void Print_header();                                        // Prints Header
void Read_temp(unsigned char *data);            // Read Temperature sensor

//start of main program
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;
```

The next section sets up the I^{2}C. There is nothing different here than was used in the last lab. This section must be included any time there is a desire to use I^{2}C. The informational part here is to explain where these functions come from. By googling "tivaware device driver users guide" you will be taken to this page as an option: http://www.ti.com/tool/sw-tm4c . Open this file: TivaWare™ Peripheral Driver Library for C Series User's Guide. It is actually a driver library. According to this document, the Texas Instruments® Tiva® Peripheral Driver Library is a set of drivers for accessing the peripherals found on the Tiva family of ARM® Cortex™-M based microcontrollers. While they are not drivers in the pure operating system sense (that is, they do not have a common interface and do not connect into a global device driver infrastructure), they do provide a mechanism that makes it easy to use the device's peripherals. As an example, when this pdf is opened, if the designer searches for "SysCtlPeripheralEnable" then they will be taken to the page shown in Figure 7.

- void SysCtlIntDisable (uint32_t ui32Ints)
- void SysCtlIntEnable (uint32_t ui32Ints)
- void SysCtlIntRegister (void (*pfnHandler)(void))
- uint32_t SysCtlIntStatus (bool bMasked)
- void SysCtlIntUnregister (void)
- void SysCtlMOSCConfigSet (uint32_t ui32Config)
- void SysCtlPeripheralClockGating (bool bEnable)
- void SysCtlPeripheralDeepSleepDisable (uint32_t ui32Peripheral)
- void SysCtlPeripheralDeepSleepEnable (uint32_t ui32Peripheral)
- void SysCtlPeripheralDisable (uint32_t ui32Peripheral)
- void SysCtlPeripheralEnable (uint32_t ui32Peripheral)

April 11, 2013

Figure 7. Hyperlink for Drivers

The key to remember is just how powerful this ARM processor is. If you glance through this document's table of contents then it is clear to see just how powerful it is. There are functions for using ADC, CAN, I^{2}C, UART, USB, ….. For the purposes of this tutorial (and the other tutorials), the projects are being reduced to the bare minimum.

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

Figure 8. Enabling and Configuring Ports and Pins

As shown above in Figure 8, first the peripherals are enabled. Next the General Purpose IO pins are configured. The final line is where the I^2C address is assigned through the variable "TEMP_ADDR".

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

Figure 9. Enabling and Configuring the UART

The next set of code that is shown above is the UART code. This sets the normal variables that designer are used to seeing with a UART (115200, 8, none, 1). Once again, each of these functions can be found in the driver user guide. Take the time to compare the two sets of code above. Look at how similar the first four lines of each section are. These four lines are mandatory for any time there is a desire to use a peripheral. The first two lines enable the peripheral. The second two lines configure the GPIO pins.

```
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]);  //Print to UART here
        }
}
```

The next set of code shown above is nothing more than a print function. It takes the global variable called start_screen that was declared as a global variable at the top of the program and sends it out to the UART screen.

```
void Read_temp(unsigned char *data){            // Read Temperature sensor

        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

		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;
		}

}
```

The final set of code is shown above. This code is designed to read temperature from the temperature sensor and format it the right way. The BURST_RECEIVE_START and BURST_RECEIVE_FINISH are constants that are used to identify the START and END conditions on the I^2C line. For the displayed temperature values, the plan is xx.x where the xx is a whole number, the decimal point is a fixed value and the final x is a value that is either .5 or .0. Temp[0] is the value on the left side of the decimal point and temp[1] is the value on the right hand side of the decimal point. The assignment of the data [0, 1, and 3] are done next. The crazy looking code there is to convert the information from HEX to ASCII. Data 0 and 1 are the left two positions/numbers. Data 2 is the fixed decimal point and data 3 is the value on the right hand side of the decimal point (0 or 5).

The next part is dependent on if you have the word file or the pdf in a classroom. If you are doing this as part of the workshop, you should now look at main.c. If you are part of the workshop, you will now see parts that need to be typed in. All of the final code is shown in Attachment 2. When all the final code is typed in, click on the debug icon/bug as shown in Figure 10.

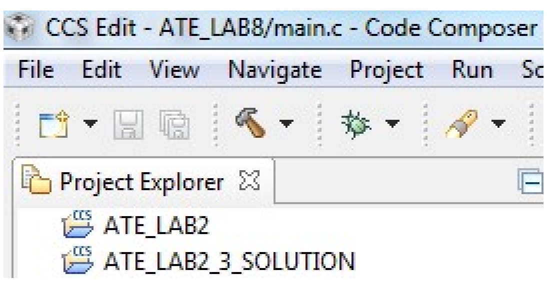


Figure 10. Debug Window

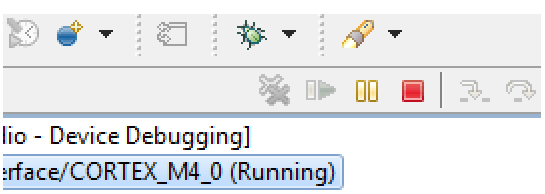


Figure 11. Run and Stop

Click on the green angle to load the program and then on the red square to exit debug mode. It is important to exit the debug mode to allow Putty and the UART to work correctly by freeing up the port.

Start Putty.

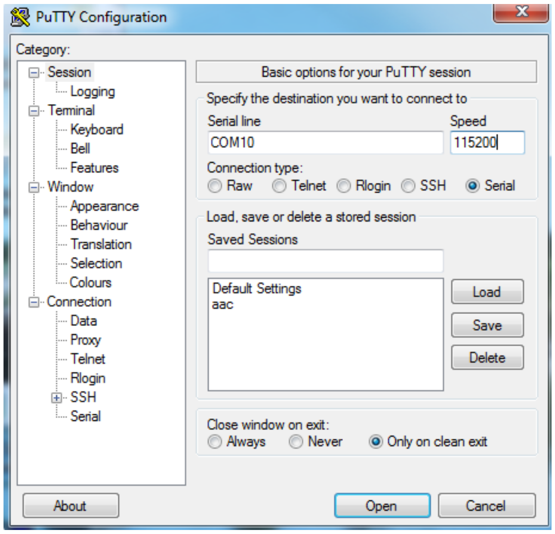


Figure 12. Putty Configuration

Launch the Putty program with the configuration settings shown above in Figure 12. Note that your comm port will most likely be different than this one. You will have to go to your Device Manager to ensure you have the correct comm port for your computer.

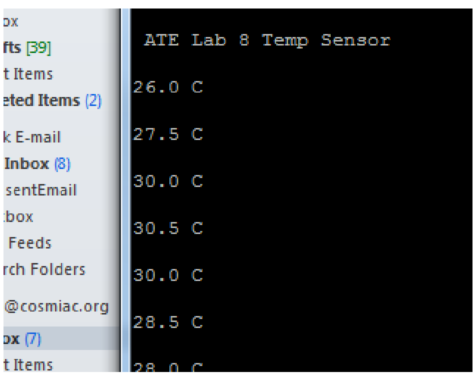


Figure 13. Temperature Output

As shown on Figure 13, once the program is running the designer can visualize the output on Putty. The temperature is displayed. Now, by putting your finger on the IC4 chip on the Orbit board, it is possible to raise the temperature. This same system could be used to control fans or other items in a house based on specific temperatures. By pressing the reset button on the Tiva board it is possible to restart the program and see the ATE Lab line.

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

Attachment 1: main.c file (starting file)

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

Project : Orbit Lab 7 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
#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[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
	//***********************************************************************************************************************
***********

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

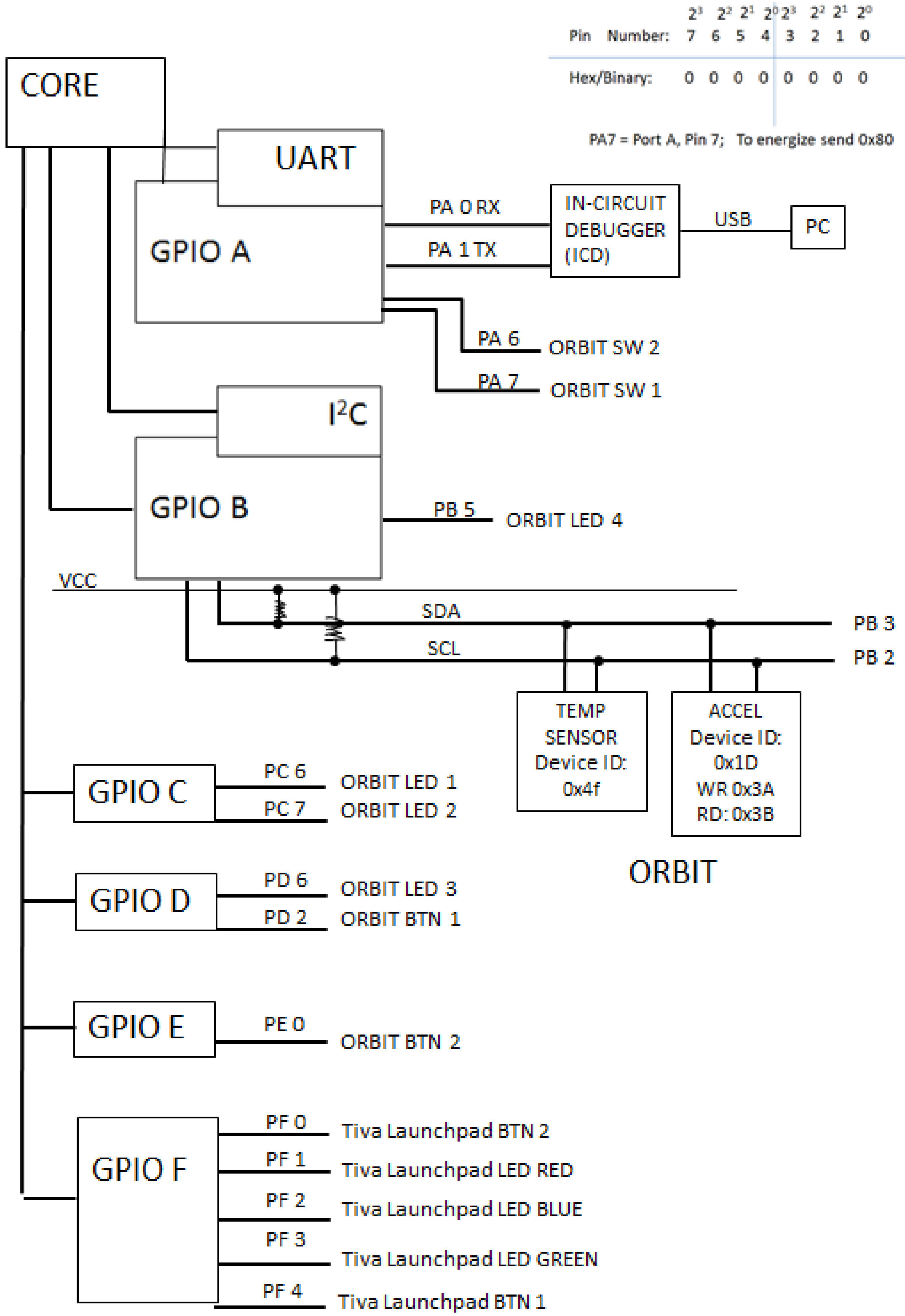
```
	SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);		// Enable UART hardware
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);		// Enable Pin hardware

	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
		SysCtlDelay(6000000);					// Delay

		/*************************************************************************
		 * The value of the temperature sensor is placed in the array temp_data. Printout
		 * this string using a FOR loop.
		 *************************************************************************/

	}

}

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

	/*****************************************************************************************************
	 * The Temperature sensor provides 2 bytes that are the C degrees. Create code to read these bytes and place
	 * them in temp[0] & temp[1].
	 *****************************************************************************************************/

	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;
	}

}
```

Attachment 2: main.c file (solution)

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

Project : Orbit Lab 7 ATE (Temp With UART)
Version : 1.0
Date    : 2/20/2013
Author  : Brian Zufelt / Craig Kief
Company : COSMIAC/UNM
Comments:
 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
#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[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
        //*****************************************************************************************************
***********

        // 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
                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;
        }

}
```

Attachment 3: Block Diagram of the Pins Used in Projects



Attachment 4: ASCII Table

Table ASCII -I

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□