TI ARM Lab 8
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 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 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 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

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://www.cosmiac.org/Microcontrollers.html . 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 Stellaris 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 .

Lab Procedure : Install/Connect board to computer

Plug in the supplied USB cable to the top of the Evaluation Kit and Digilent Orbit board as shown in Figure 1. Ensure the switch on the board is set to "DEBUG" and not "DEVICE". It is assumed that the evaluation board is properly installed and operating. Launch the Code Composer software. The easiest way to find it, is to go to Start, All programs and then look in the area identified in Figure 2.

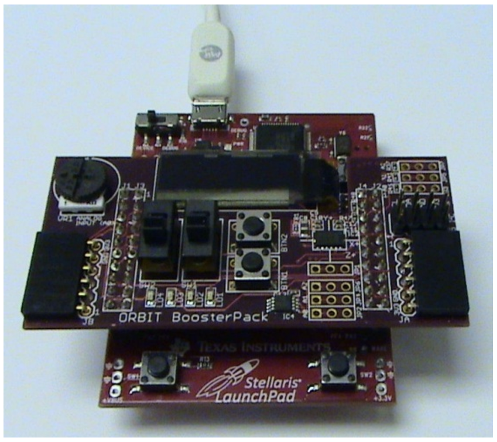


Figure 1. Stellaris and Orbit Combination

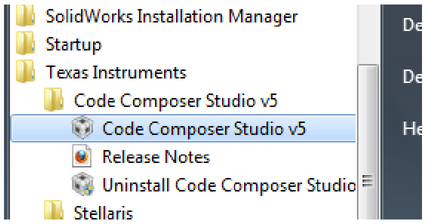
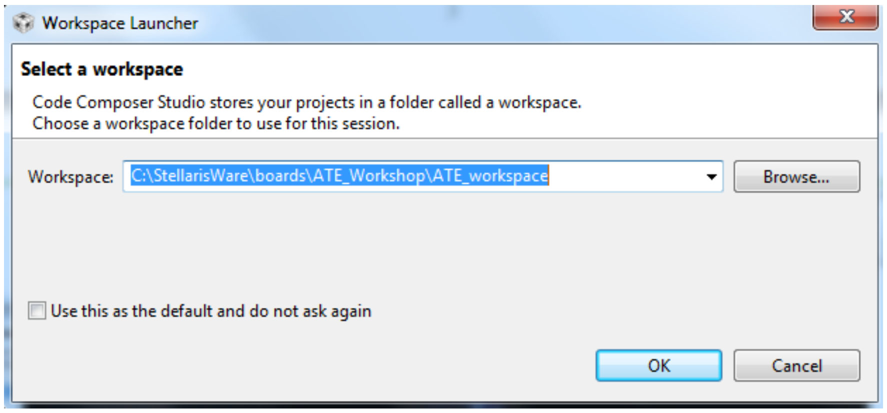


Figure 2. Code Composer



Workspace Launcher

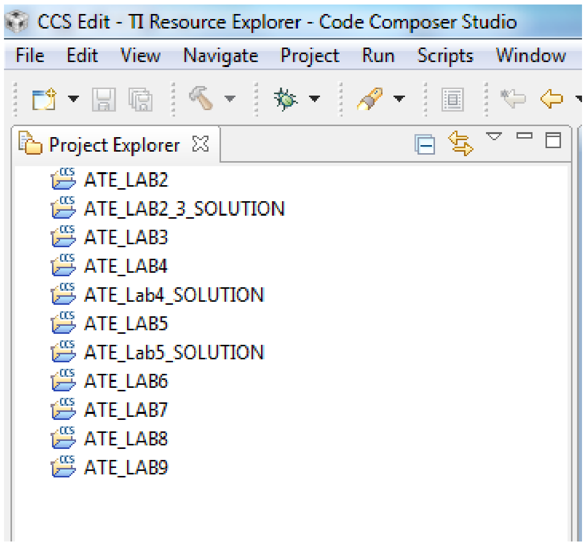
Select a workspace

Code Composer Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\StellarisWare\boards\ATE_Workshop\ATE_workspace

Figure 3. Workspace Launcher

The designer will be presented with the choices shown in Figure 3. The Code Composer wants to know where the workspace is located. If the auto installer was used to preload all the tutorials, it will automatically load up shells that are used for the workshop and these tutorials. In that case, use the path as shown in the picture in Figure 3. The student will know if they have used the auto installer program as they will be presented with a picture as shown in Figure 4 where the shells of all lab projects are installed. The "Solution" directories were put in during development but then removed before the installer was released. What is desired is to have the student step through the Lab_8 exercise first by just reading the document and then by following the procedures.


CCS Edit - TI Resource Explorer - Code Composer Studio
File Edit View Navigate Project Run Scripts Window
Project Explorer

ATE_LAB2
ATE_LAB2_3_SOLUTION
ATE_LAB3
ATE_LAB4
ATE_Lab4_SOLUTION
ATE_LAB5
ATE_Lab5_SOLUTION
ATE_LAB6
ATE_LAB7
ATE_LAB8
ATE_LAB9

Figure 4. Shell of Projects

Just as with the accelerometer, the temperature sensor has an I^{2}C interface. This is shown in Figure 5 and is very similar to the one showed in Tutorial 7.

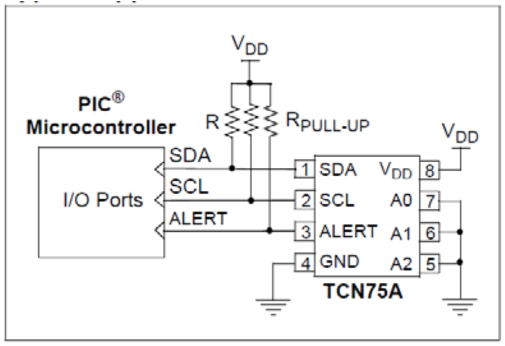
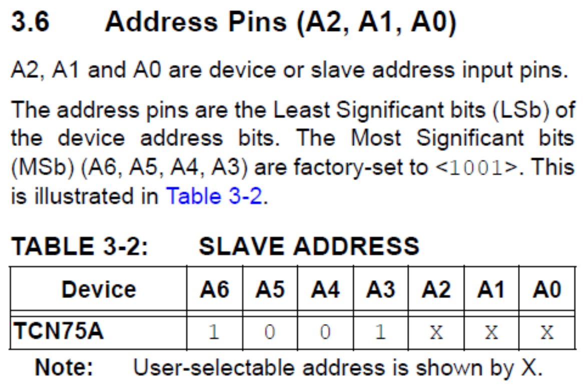


Figure 5. Temperature Sensor Interface

Many parts can share the I^{2}C 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.

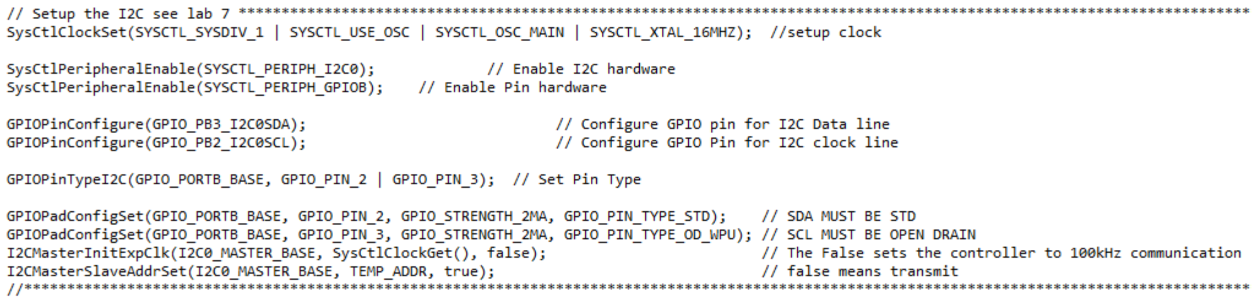
Figure 6. Temperature Sensor Datasheet excerpt

The portion of the datasheet copied for Figure 6 shows the address for connecting 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^{2}C 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. 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. 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.

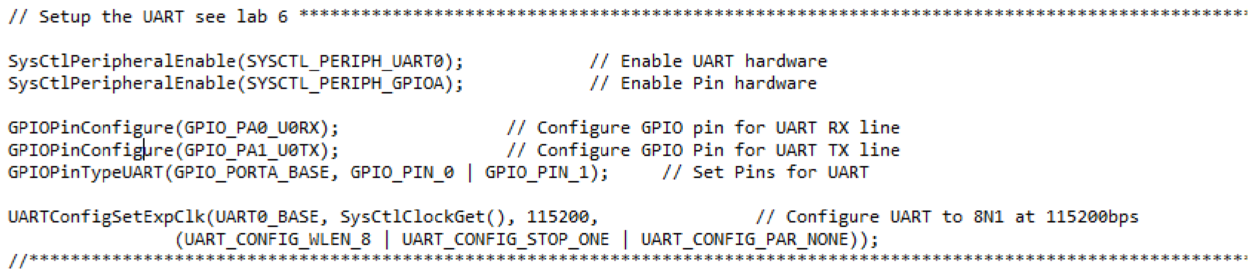
```
unsigned char start_screen[29] = "\n\n\r ATE Lab 8 Temp Sensor \n\n\r";  //29 bytes long
unsigned char log[18] = "\n\n\r Temp reading: ";  //18 bytes long

//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. On the community portal website (http://cosmiac.org/Community_Portal_Micro.html) there is a 3M 518 page pdf document called the Driver_userguide.pdf. It is actually a driver library. According to this document, the Texas Instruments® Stellaris® Peripheral Driver Library is a set of drivers for accessing the peripherals found on the Stellaris 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 on page 363 under section 24.2.2.30, an entire section is designated to this item. The key to remember is just how powerful this ARM processor is. If you glance through this documents 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_WPU); // SCL MUST BE OPEN DRAIN
I2CMasterInitExpClk(I2C0_MASTER_BASE, SysCtlClockGet(), false);                         // The False sets the controller to 100kHz communication
I2CMasterSlaveAddrSet(I2C0_MASTER_BASE, TEMP_ADDR, true);                               // false means transmit
//***************************************************************************************************************
```

As shown above, first the peripherals are enabled. Next the General Purpose IO pins are configured. The final line is where the I^{2}C address is assigned.



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

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 that are shown in blue enable the peripheral. The second two lines that are shown in green configure the GPIO pins.

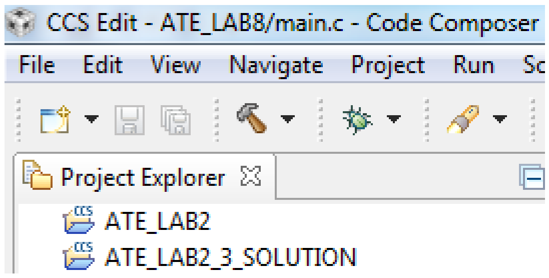
```
void Print_header(){                            // Print Header at start of program
        int i = 0; // general counter
        for(i=0;i<29;i++){ // Print Header at start of program
                UARTCharPut(UART0_BASE, start_screen[i]);  //Print to UART here
        }
}
```

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^{2}C 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 1. When all the final code is typed in, click on the debug icon/bug as shown in Figure 7.


CCS Edit - ATE_LAB8/main.c - Code Composer
File Edit View Navigate Project Run Sc
Project Explorer
ATE_LAB2
ATE_LAB2_3_SOLUTION
Figure 7. Debug

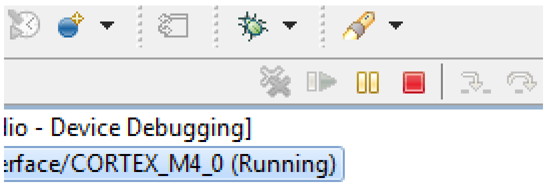
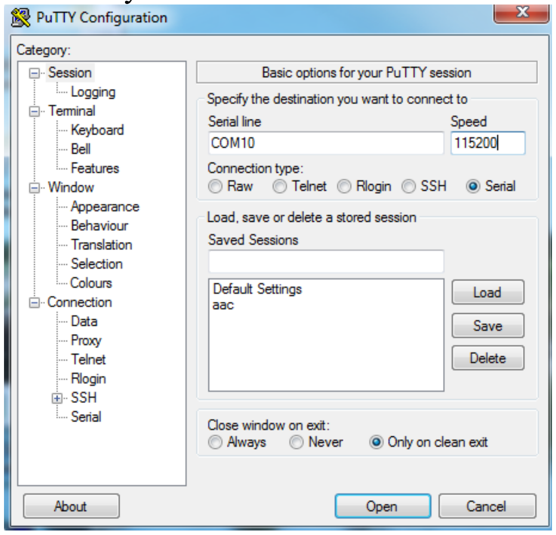

lio - Device Debugging]
:rface/CORTEX_M4_0 (Running)

Figure 8. 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.


PuTTY Configuration

Category: Session, Logging, Terminal, Keyboard, Bell, Features, Window, Appearance, Behaviour, Translation, Selection, Colours, Connection, Data, Proxy, Telnet, Rlogin, SSH, Serial

Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM10

Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

Saved Sessions

Default Settings
aac

Close window on exit: Always Never [✓ Only on clean exit]

Figure 9. Putty Configuration

Launch the Putty program with the configuration settings shown above in Figure 9. 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.

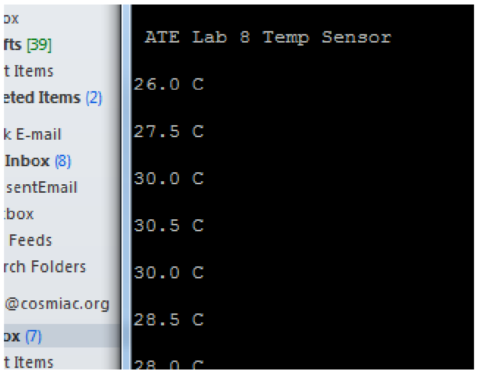

```
 ATE Lab 8 Temp Sensor

26.0 C

27.5 C

30.0 C

30.5 C

30.0 C

28.5 C

28.0 C
```

Figure 10. Temperature Output

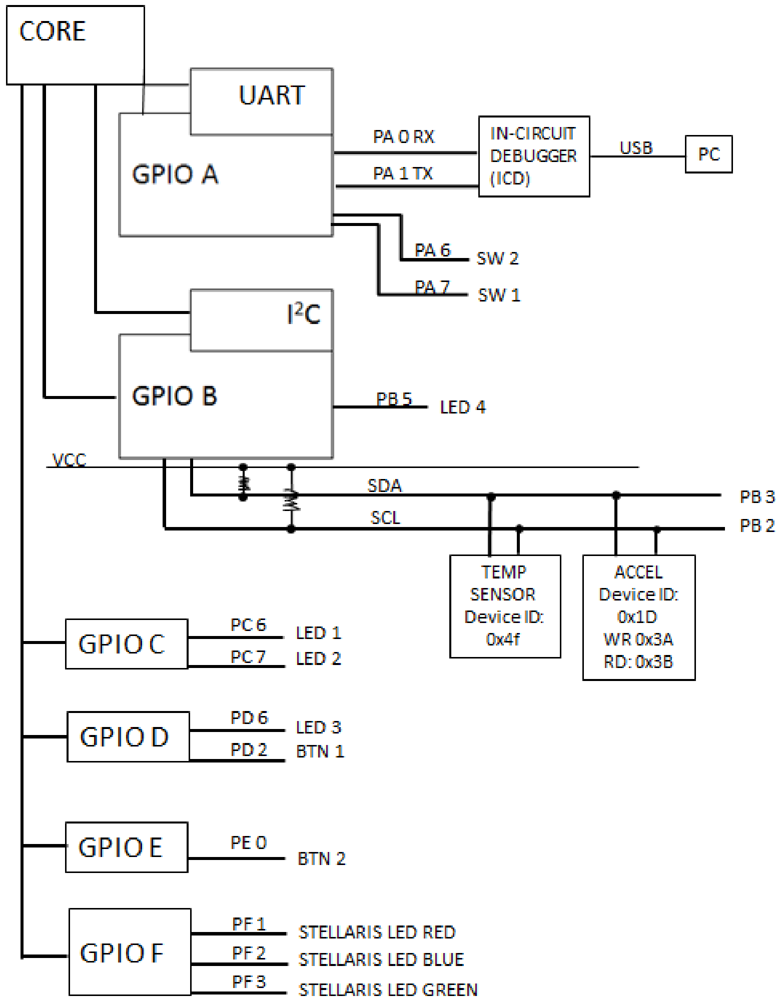
As shown on Figure 10, once the program is running the designer can visualize the output on Putty. The temperature is displayed. Now, by putting your finger on the IC4 chip on the Orbit board, it is possible to raise the temperature. This same system could be used to control fans or other items in a house based on specific temeratures. By pressing the reset button on the Stellaris 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

```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 LM44F120H5QR
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_LM4F120H5QR

#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_WPU);         // SCL MUST BE OPEN
DRAIN
        I2CMasterInitExpClk(I2C0_MASTER_BASE, SysCtlClockGet(), false);
        // The False sets the controller to 100kHz communication
        I2CMasterSlaveAddrSet(I2C0_MASTER_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_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;
        }

}
```

Attachment 2: Block Diagram of the Pins Used in Projects


CORE
UART
GPIO A
PA 0 RX
PA 1 TX
IN-CIRCUIT DEBUGGER (ICD)
USB
PC
PA 6
SW 2
PA 7
SW 1
I²C
GPIO B
PB 5
LED 4
VCC
SDA
SCL
PB 3
PB 2
TEMP SENSOR Device ID: 0x4f
ACCEL Device ID: 0x1D WR 0x3A RD: 0x3B
GPIO C
PC 6
LED 1
PC 7
LED 2
GPIO D
PD 6
LED 3
PD 2
BTN 1
GPIO E
PE 0
BTN 2
GPIO F
PF 1
STELLARIS LED RED
PF 2
STELLARIS LED BLUE
PF 3
STELLARIS LED GREEN