BEE 425: Microprocessor System Design

Lab 5: Final Lab Project

Winter 2019  
Lab Section : Tuesday 1:15 - 3:15

Justus Bautista & Vivek Moolayil

**\*\*\*Note: All parts of the demo have been checked off as complete by Joe Decuir\*\*\***

**Abstract**

In this final lab project we use all of our gained knowledge from the previous labs to create our temperature controller. We output a PWM signal from lab 4 and used a input ADC converter from lab 3. We learn more about PCB design and had to apply soldering skills to put together the final PCB.

**Introduction**

For this final lab project our objective was to drive and control the temperature of an ohmic heater then display our results on a pair of seven segment displays. This project required us to learn new software such as ExpressSCH and Express PCB in order to create a PCB. Although we did not use our own design, rather we used a PCB designed by Ben Roc.

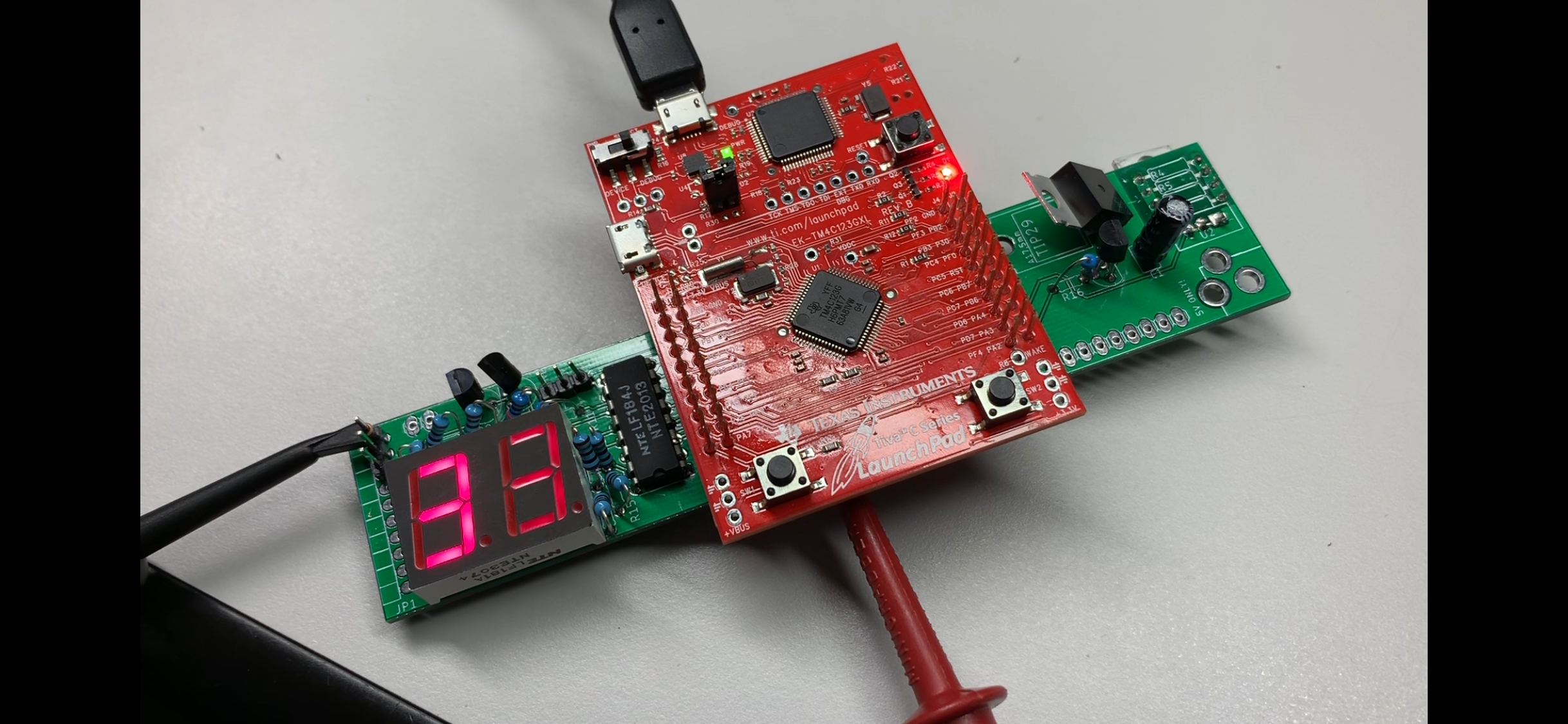
**Procedure**

1. We first create a Schematic of the circuit (Milestone 1).
2. Then we create a PCB layout of the circuit and code (Milestone 2).
3. Next, get all of the parts for the PCB and the PCB itself.
4. After, solder all parts onto the PCB.
5. Then we attach the TIVA board to the PCB as well.
6. We next refine and debug the control program code from milestone 2.
   1. Control Program should display decimal values of temperature on Hex display, drive and regulate the heater it to keep it at a set value, and display on the TIVA LEDs when the heater is too hot or too cold.
7. We download it onto the TIVA board while it is connected to the PCB.
8. Lastly, test and debug the program. This can be done in parts to make sure each aspect of the final project works.

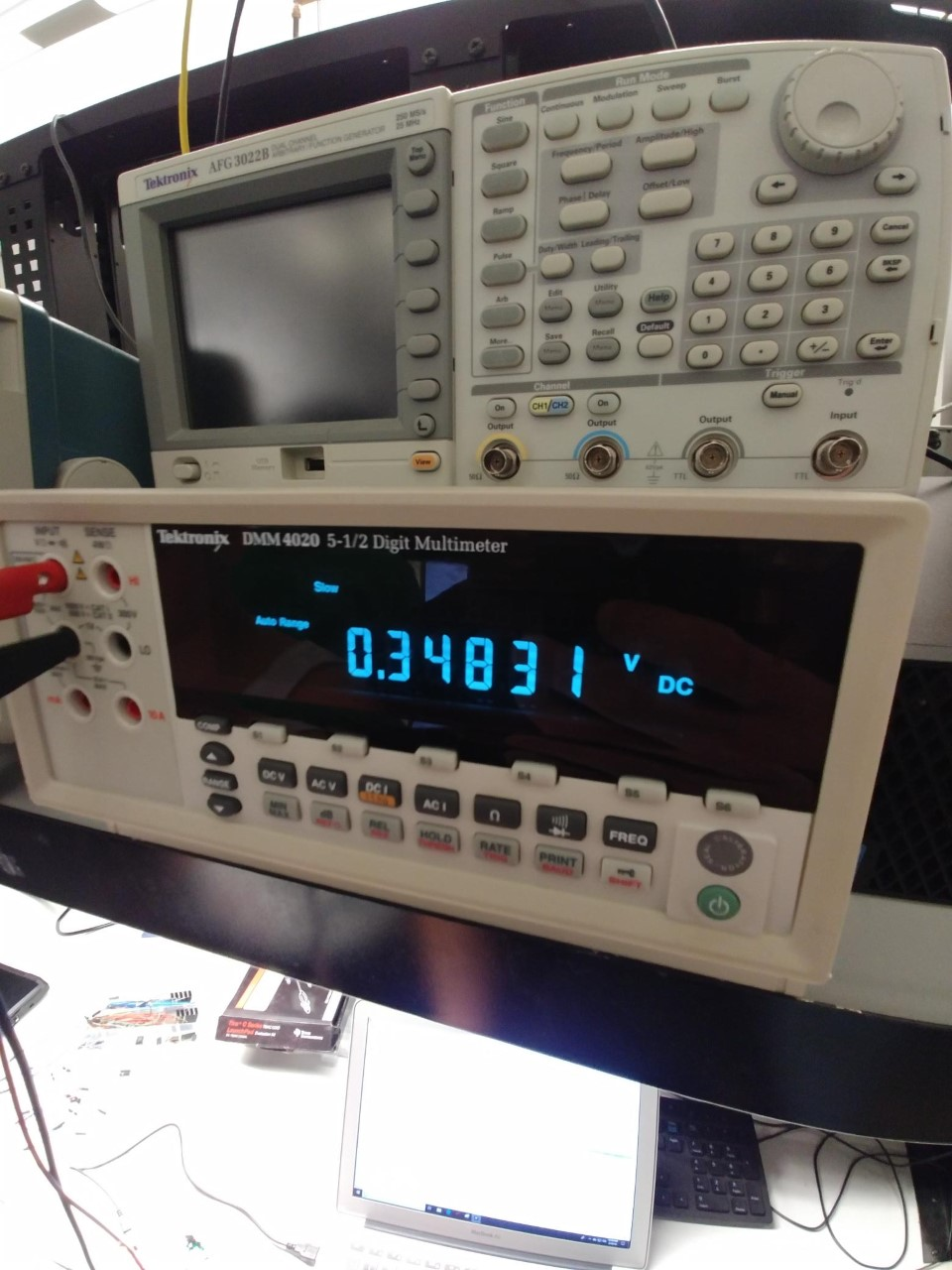
**Results**

We were able to successfully complete all parts of the final lab project. Our TIVA LEDs changed colors depending on if the temperature was too high or low. Our Hex display showed the correct temperature values in decimal. Figure 1 shows the PCB and TIVA board with the Hex display on, and the LED is set to red (too hot). Note that this picture is taken at a single point in time so the right Hex is off currently (value is 35). Figure 2 below shows the max voltage that we set our heater to reach. This is equal to 34 degrees Celsius. Figure 3 shows our oscilloscope measurement of the PWM output. It shows how the PWM turns on and off to keep the temperature at the max value.

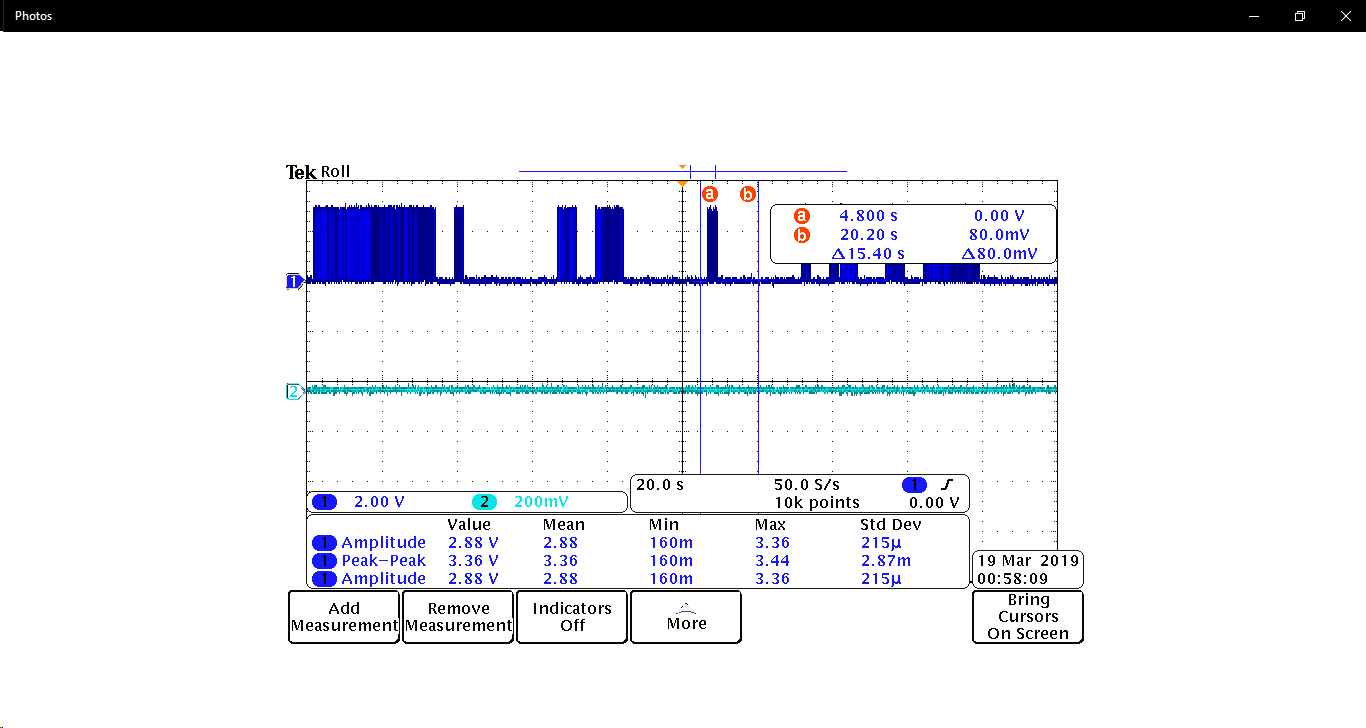
**Figure 1: Board with hex display on and LED red**



**Figure 2: Approximately the max voltage before heater turns off**



**Figure 3: Oscilloscope Screenshot of PWM**



**Figure 3: CODE**

#include "tm4c123gh6pm.h"

#include <stdint.h> // needed to do integer operations, number etc.

#include <stdio.h>

// Tiva LED Colors

#define RED 0x02

#define BLUE 0x04

// 7 Segment Numbers

#define Nine 0x73

#define Eight 0x7F

#define Seven 0x70

#define Six 0x5F

#define Five 0x5B

#define Four 0x33

#define Three 0x79

#define Two 0x6D

#define One 0x30

#define Zero 0x7E

int hextoDec(int hex)

{

int i = 1;

int dec = 0;

while (i <= 4096) {

dec += (hex & i);

i \*= 2;

}

return dec;

}

void Delay(void) {

unsigned long volatile time;

time = (727240\*50/91); //0.5 sec

while(time){

time--;

}

}

int main()

{

volatile int result; // volatile since result could change at any time

volatile int Decimal;

volatile int Left;

volatile int Right;

volatile int LD;

volatile int RD;

SYSCTL\_RCGC2\_R = 0; // clear the system control clock register

SYSCTL\_RCGC2\_R |= SYSCTL\_RCGC2\_GPIOA; // enable clock on port A

SYSCTL\_RCGC2\_R |= SYSCTL\_RCGC2\_GPIOB; // enable clock on port B

SYSCTL\_RCGC2\_R |= SYSCTL\_RCGC2\_GPIOC; // enable clock on port C

SYSCTL\_RCGC2\_R |= SYSCTL\_RCGC2\_GPIOD; // enable clock on port D

SYSCTL\_RCGC2\_R |= SYSCTL\_RCGC2\_GPIOE; // enable clock on port E

SYSCTL\_RCGC2\_R |= SYSCTL\_RCGC2\_GPIOF; // enable clock on port F

SYSCTL\_RCGCADC\_R = SYSCTL\_RCGCADC\_R0; // enable clock to ADC0

SYSCTL\_RCGCPWM\_R |= SYSCTL\_RCGCPWM\_R1; // enable clock on M1PWM

//Initialize LEDs on port F

GPIO\_PORTF\_LOCK\_R = 0x4C4F434B;

GPIO\_PORTF\_CR\_R = 0x01;

GPIO\_PORTF\_PUR\_R = 0x11;

GPIO\_PORTF\_DIR\_R = 0x0E; // enable the LEDs as output pins

GPIO\_PORTF\_DEN\_R = 0x1F; // enable the LEDs as digital pins

// initialize PA7 for PWM3 output

SYSCTL\_RCC\_R &= ~0x00100000; // Disable pre-divide

GPIO\_PORTA\_AFSEL\_R = 0x80; // Enable AFSEL on PA7, bit 7

GPIO\_PORTA\_PCTL\_R &= ~0xF0000000; //Disable PCTL

GPIO\_PORTA\_PCTL\_R |=0x50000000; // Enable PCTL on PA7

GPIO\_PORTA\_DEN\_R |= 0x80; // Enable PWM3 on PA7

//Setup PWM for PA7

SYSCTL\_RCGCPWM\_R |= SYSCTL\_RCGCPWM\_R1; // Enable clock on M1PWM

PWM1\_1\_CTL\_R = 0; // Disable PWM

PWM1\_1\_GENB\_R = 0x8C; // Set Generator B: Reload -> 1, CMPA -> 0

PWM1\_1\_LOAD\_R = 0xFF; // Set initial load to be FF, 8-bit divisor

PWM1\_1\_CMPA\_R = 0x80; // Set Comparator to roughly 50% duty cycle

PWM1\_1\_CTL\_R = 1; // Enable PWM

PWM1\_ENABLE\_R = 0x8; // Enable PWM on M1PWM3 (PA7)

// Initialize PORTB for 7 segment display, turn on segments

GPIO\_PORTB\_DIR\_R = 0xFF; // enable Port B pins as outputs

GPIO\_PORTB\_DEN\_R = 0xFF; // enable Port B as digital pins

// Initialize PORTC for 7 segment display, choose digit

GPIO\_PORTC\_DIR\_R = 0xFF; // enable Port C pins as outputs

GPIO\_PORTC\_DEN\_R = 0xFF; // enable Port C as digital pins

// initialize PE3 for AIN0 input, temp sensor

GPIO\_PORTE\_AFSEL\_R |= 8; // enable alternate function

GPIO\_PORTE\_DEN\_R &= ~8; // disable digital function

GPIO\_PORTE\_AMSEL\_R |= 8; // enable analog function

// initialize ADC0 for sampling sensor

ADC0\_ACTSS\_R &= ~8; // disable SS3 during configuration

ADC0\_EMUX\_R = (0xF<<12); // software trigger conversion

ADC0\_SSMUX3\_R = 0; // get input from channel 0

ADC0\_SSCTL3\_R |= 6; // take one sample at a time, set flag at 1st sample

ADC0\_IM\_R = (1<<3); // Interrupt Mask for SS3 must be on in order to interrupt

ADC0\_ACTSS\_R |= 8; // enable ADC0 sequencer 3

//Setup SysTick

NVIC\_ST\_RELOAD\_R = 15999; // Initial Load value of SYSTICK(16000-1)

NVIC\_ST\_CTRL\_R = 5; // Start SYSTICK

while(1)

{

//Read temperature sensor

ADC0\_PSSI\_R |= 8; // start a conversion at sequence 3

while((ADC0\_RIS\_R & 8) == 0); // while the conversion is not complete, do nothing

result = ADC0\_SSFIFO3\_R; // read the result from the SSFIFO3 register

Decimal = hextoDec(result);

Decimal /= 10;

Decimal -= 9;

Left = Decimal / 10;

Right = Decimal % 10;

printf("Result : %d\n", Decimal);

if (result > 0x1A9) {

//if temperature is too hot , greater than 35 deg C

GPIO\_PORTA\_DEN\_R = 0x00;

GPIO\_PORTA\_DIR\_R = 0x00;

GPIO\_PORTA\_DATA\_R = 0x00;

GPIO\_PORTF\_DATA\_R = RED; //LED is red to indicate too hot

}

else {

GPIO\_PORTA\_DEN\_R = 0x80;

GPIO\_PORTA\_DIR\_R = 0x80;

GPIO\_PORTA\_DATA\_R = 0x80;

GPIO\_PORTF\_DATA\_R = BLUE; //LED is blue to indicate too cold

}

// Display on HEX display

switch(Left){

case 0:

GPIO\_PORTC\_DIR\_R = 0x80;

GPIO\_PORTC\_DEN\_R = 0x80;

GPIO\_PORTC\_DATA\_R = 0x80;

GPIO\_PORTB\_DATA\_R = Zero;

Delay();

break;

case 1:

GPIO\_PORTC\_DIR\_R = 0x80;

GPIO\_PORTC\_DEN\_R = 0x80;

GPIO\_PORTC\_DATA\_R = 0x80;

GPIO\_PORTB\_DATA\_R = One;

Delay();

break;

case 2:

GPIO\_PORTC\_DIR\_R = 0x80;

GPIO\_PORTC\_DEN\_R = 0x80;

GPIO\_PORTC\_DATA\_R = 0x80;

GPIO\_PORTB\_DATA\_R = Two;

Delay();

break;

case 3:

GPIO\_PORTC\_DIR\_R = 0x80;

GPIO\_PORTC\_DEN\_R = 0x80;

GPIO\_PORTC\_DATA\_R = 0x80;

GPIO\_PORTB\_DATA\_R = Three;

Delay();

break;

case 4:

GPIO\_PORTC\_DIR\_R = 0x80;

GPIO\_PORTC\_DEN\_R = 0x80;

GPIO\_PORTC\_DATA\_R = 0x80;

GPIO\_PORTB\_DATA\_R = Four;

Delay();

break;

case 5:

GPIO\_PORTC\_DIR\_R = 0x80;

GPIO\_PORTC\_DEN\_R = 0x80;

GPIO\_PORTC\_DATA\_R = 0x80;

GPIO\_PORTB\_DATA\_R = Five;

Delay();

break;

case 6:

GPIO\_PORTC\_DIR\_R = 0x80;

GPIO\_PORTC\_DEN\_R = 0x80;

GPIO\_PORTC\_DATA\_R = 0x80;

GPIO\_PORTB\_DATA\_R = Six;

Delay();

break;

case 7:

GPIO\_PORTC\_DIR\_R = 0x80;

GPIO\_PORTC\_DEN\_R = 0x80;

GPIO\_PORTC\_DATA\_R = 0x80;

GPIO\_PORTB\_DATA\_R = Seven;

Delay();

break;

case 8:

GPIO\_PORTC\_DIR\_R = 0x80;

GPIO\_PORTC\_DEN\_R = 0x80;

GPIO\_PORTC\_DATA\_R = 0x80;

GPIO\_PORTB\_DATA\_R = Eight;

Delay();

break;

case 9:

GPIO\_PORTC\_DIR\_R = 0x80;

GPIO\_PORTC\_DEN\_R = 0x80;

GPIO\_PORTC\_DATA\_R = 0x80;

GPIO\_PORTB\_DATA\_R = Nine;

Delay();

break;

}

switch(Right){

case 0:

GPIO\_PORTC\_DIR\_R = 0x40;

GPIO\_PORTC\_DEN\_R = 0x40;

GPIO\_PORTC\_DATA\_R = 0x40;

GPIO\_PORTB\_DATA\_R = Zero;

Delay();

break;

case 1:

GPIO\_PORTC\_DIR\_R = 0x40;

GPIO\_PORTC\_DEN\_R = 0x40;

GPIO\_PORTC\_DATA\_R = 0x40;

GPIO\_PORTB\_DATA\_R = One;

Delay();

break;

case 2:

GPIO\_PORTC\_DIR\_R = 0x40;

GPIO\_PORTC\_DEN\_R = 0x40;

GPIO\_PORTC\_DATA\_R = 0x40;

GPIO\_PORTB\_DATA\_R = Two;

Delay();

break;

case 3:

GPIO\_PORTC\_DIR\_R = 0x40;

GPIO\_PORTC\_DEN\_R = 0x40;

GPIO\_PORTC\_DATA\_R = 0x40;

GPIO\_PORTB\_DATA\_R = Three;

Delay();

break;

case 4:

GPIO\_PORTC\_DIR\_R = 0x40;

GPIO\_PORTC\_DEN\_R = 0x40;

GPIO\_PORTC\_DATA\_R = 0x40;

GPIO\_PORTB\_DATA\_R = Four;

Delay();

break;

case 5:

GPIO\_PORTC\_DIR\_R = 0x40;

GPIO\_PORTC\_DEN\_R = 0x40;

GPIO\_PORTC\_DATA\_R = 0x40;

GPIO\_PORTB\_DATA\_R = Five;

Delay();

break;

case 6:

GPIO\_PORTC\_DIR\_R = 0x40;

GPIO\_PORTC\_DEN\_R = 0x40;

GPIO\_PORTC\_DATA\_R = 0x40;

GPIO\_PORTB\_DATA\_R = Six;

Delay();

break;

case 7:

GPIO\_PORTC\_DIR\_R = 0x40;

GPIO\_PORTC\_DEN\_R = 0x40;

GPIO\_PORTC\_DATA\_R = 0x40;

GPIO\_PORTB\_DATA\_R = Seven;

Delay();

break;

case 8:

GPIO\_PORTC\_DIR\_R = 0x40;

GPIO\_PORTC\_DEN\_R = 0x40;

GPIO\_PORTC\_DATA\_R = 0x40;

GPIO\_PORTB\_DATA\_R = Eight;

Delay();

break;

case 9:

GPIO\_PORTC\_DIR\_R = 0x40;

GPIO\_PORTC\_DEN\_R = 0x40;

GPIO\_PORTC\_DATA\_R = 0x40;

GPIO\_PORTB\_DATA\_R = Nine;

Delay();

break;

}

}

return 0; // terminate while(1) statement

}

**Conclusion**

After debugging the code and with help from various classmates we were able to demonstrate to Joe Decuir that we could drive the heater, regulate the temperature to under 35 degrees Celsius, convert the temperature to a two digit decimal number, and display this number on the seven segment LEDs. From this lab we learned how to use embedded C programming, the TIVA microcontroller, creation of a schematic and PCB layout, as well as assembling and soldering our own PCB board. By far the most difficult parts of this lab were trying to learn new software such as ExpressSCH and ExpressPCB as well as understanding/ implementing what we needed to code in order to drive, regulate, and display the temperature from our resistive heater.

Classmates we received assistance from:

Justin M, Ryan G, Garrett P, Chase M, Mike M, Evan W, Kenny L, Nate V