University of Washington
Department of Electrical Engineering
BEE 425, Winter 2019

Lab 5: Temperature Sensor

Report by:
Minh-duc Ho (1774400)
Garrett Padilla(1774422)
Evan Wansa(1774429)

Abstract:

In this lab, we are designing and creating a temperature sensor that can run on a TIVA microcontroller to control the temperature and display measurement values; the temperature will be measured on a ohmic heater and display the values on two seven segment displays. The progress of creating a temperature sensor goes by the following: develop the theory of the sensor functionality and create a schematic, create a PCB layout and write the pseudo-code, then fabricate and demonstrate the operation.

Introduction:

Our temperature sensor will be our final project for this class, and it will demonstrate our knowledge and understanding of microprocessor system design based on the labs we have completed. The system of our sensor we will be implementing is a feedback control system which will indicate the heater low, medium, and high heat. The system will indicate when our power transistor can increase the temperature on the heaters and when to turn off once it gets too hot.

Parts list:

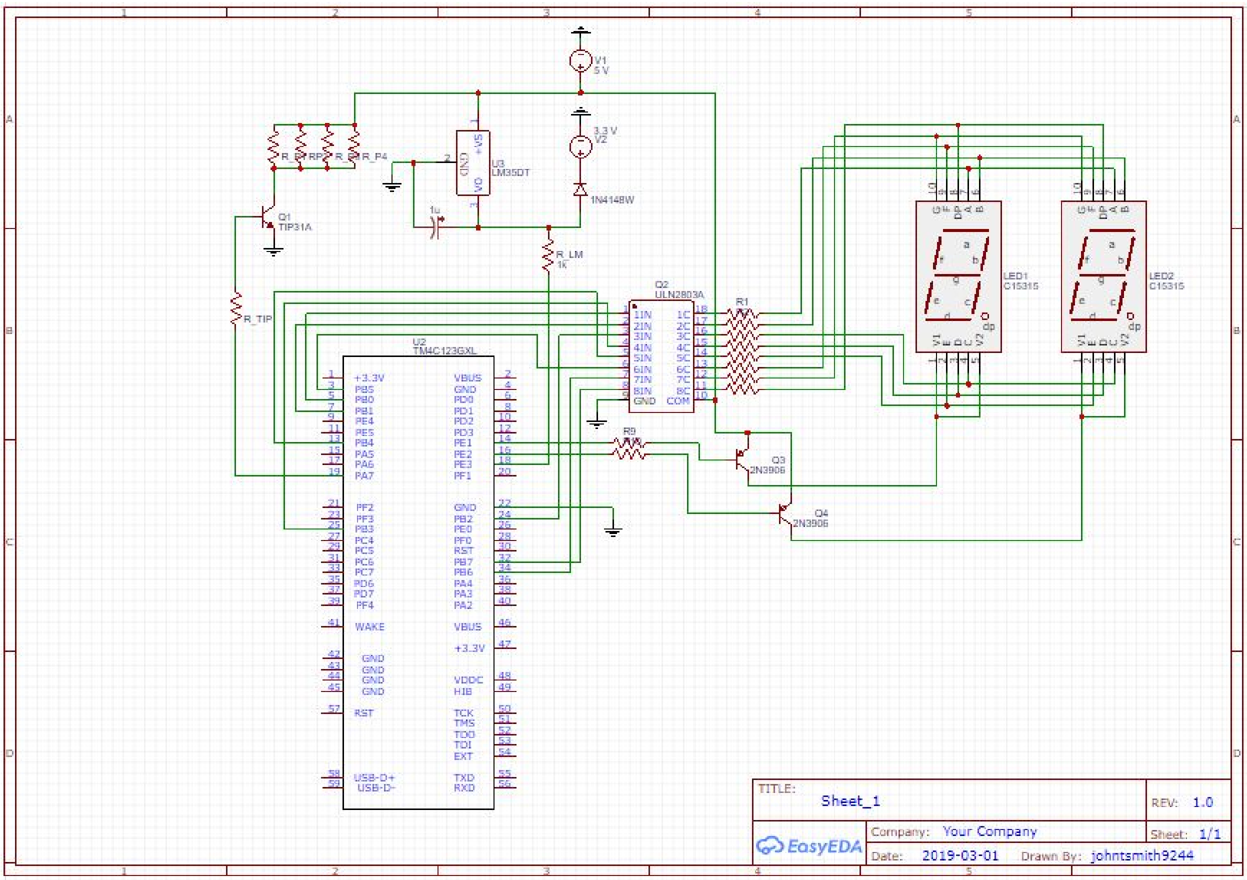
Here is the parts/components we will need to fabricate and perform our temperature sensor:

- TIVA Microcontroller
- Resistive heater: 4x120 ohm resistors in parallel
- TIP3129 power transistor
- LM35 temperature sensor
- 7-segment LED displays (NTE3074)
- 10 uF capacitor
- 3 x 330 ohm resistors
- 10k ohm resistor
- 1N4148 diode
- 2 x 2N3906 BJT
- 2N3904 BJT
- ULN2003A (Seven darlington array)

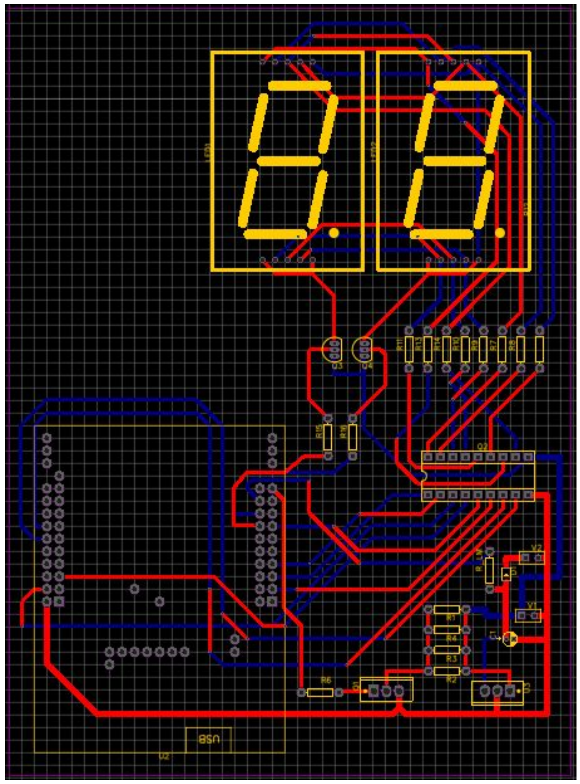
Procedure:

Milestone 1:

Tier 1 hardware and software schematics



Circuit PCB Design



Circuit PCB Layout

Milestone 2:
PCB schematic capture and tier 2 software

Milestone 3:
Prototype demo-

- Driving heater
- Thermal regulation
- LED indicator
- LED display

Results:

Figure 1 - Blue LED (too cold)

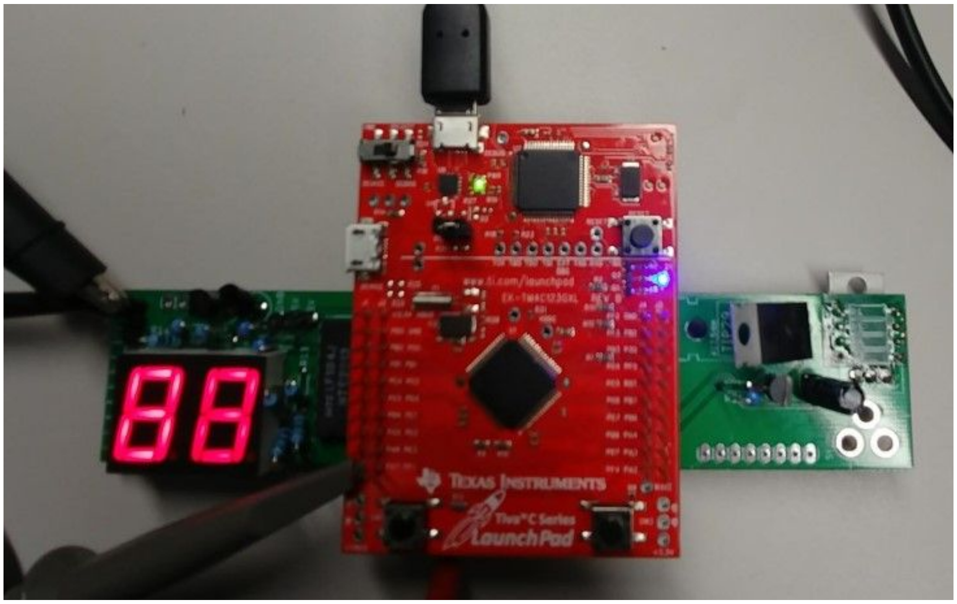


Figure 2 - Red LED (too hot)

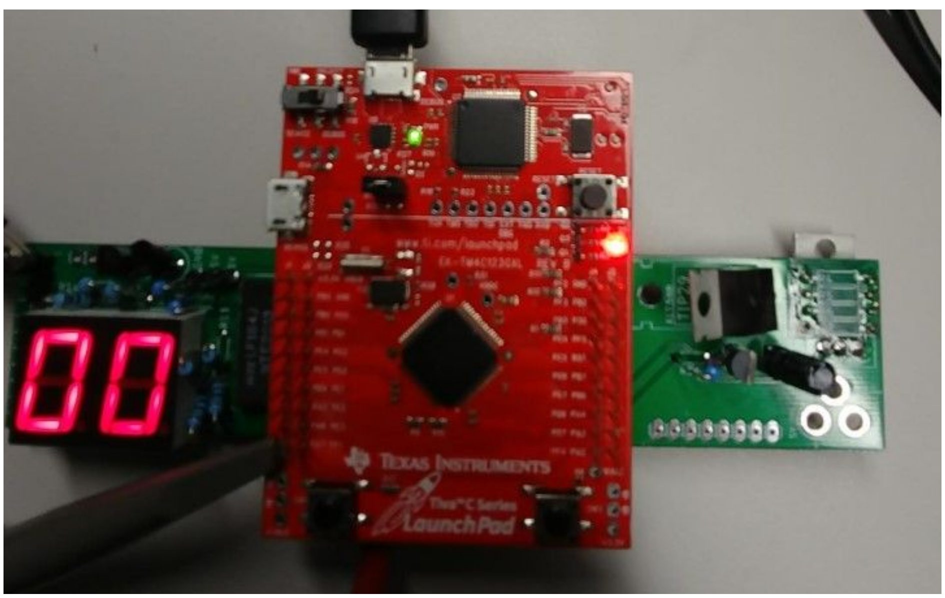
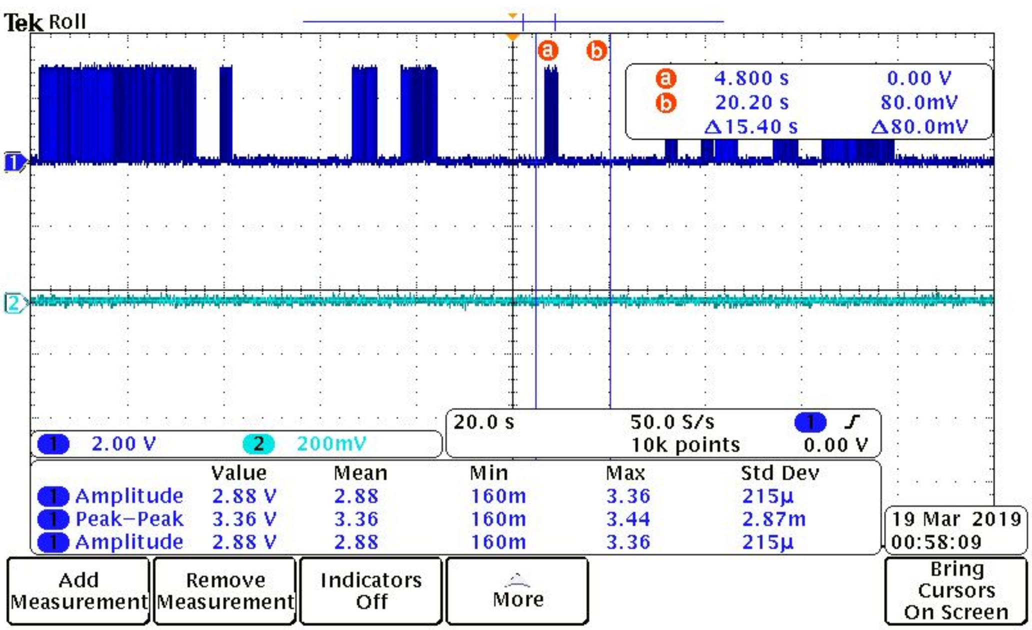


Figure 3 - temperature regulated around 34 degrees C

Figure 4 - Thermal regulation waveform



CODE:

```
//Garrett Padilla
//Evan Wansa
//Minh Ho
//Vivek Mooyalil
//Justus Bautista
#include "tm4c123gh6pm.h"
#include <stdint.h> // needed to do integer operations, number etc.
#include <stdio.h>
```

```
// Tiva LED Colors
#define RED 0x02
#define GREEN 0x08
#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 via a
mask.
	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 34 degrees C)
```

```
//reduce duty cycle
        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 {
       //if temperature is too cold
       //(less than 30 degrees C)
      //increase duty cycleq
        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:

In conclusion, we were able to run our temperature sensor off the TIVA microcontroller and display the results. It was difficult for us having one lab period to assemble our components onto the PCB, including when it came to soldering; we didn't have enough experience to be able to finish soldering all the components in time. We ended up burning part of the PCB due to removing and reassemble specific components to our board. However, we were thankful of having another group to work beside us that has a working PCB board; One of the members, Justus, successfully soldered his PCB board and taught us his technique of soldering.

We worked together for the code, and it took us a while to get the code to function the way we wanted. We successfully able to run the driver, indicate the temperature range based on the LED which determines hot and cold and stops driving heat when the heater gets too hot.

Bibliography:

We used Ben's schematic and his PCB layout for our final milestone. Joe's file Lab5 R2 gave us the background knowledge and approach for this lab. Justus and Vivek helped us out for using their PCB board, and we worked together with getting the codes to work. We reference a page we found, (https://github.com/sphanlung/TivaC/blob/master/SevenSegment1.c), by sphanlung, to get the 2-digit seven segment to display numbers.