Final Project Report: *Temperature Control System*

Eric Schroeder [Section 4], Nick Scamardi [Section 4], Nick Setaro [Section 3]

Final Project Report

May 8, 2019

1 Design Overview

The objective of this project was to create a temperature regulating system using various systems and control theory. This was done by controlling the PWM of a fan to maintain a constant temperature of a PTAT. A 5V voltage regulator was used in order to heat the PTAT and change its temperature. The desired temperature can be set by the user by turning a potentiometer, with the range going from 0 to 100 degrees Celsius. This system also includes a PID controller for accurate temperature regulation, which was simulated using Matlab and tuned accordingly. Along with the PID control scheme, both a low-pass analog filter and an FIR filter were used to ensure the system stays as consistent as possible.

1.1 Design Features

These are the design features:

- Temperature (C) of a voltage regulator is displayed through UART
- Desired temperature can be set by the user by turning the potentiometer and viewed via UART
- A fan will keep voltage regulator within 1°C of the desired temperature
- Multiple filters will ensure the system stays consistent
- Temperature ranges from 0-100°C

1.2 Featured Applications

While this design was created with specific specifications it can be used in a wide range of applications.

Featured Applications Include:

- Computer cooling system to keep electronics at certain temperatures
- The fan control can be used on a larger scale for room temperature control
- Safety measures to initiate when electronics are nearly over heated

1.3 Design Resources

This is a link to the github repository which contains the code and schematics for the project.

GitHub Link

2 Key System Specifications

The key specifications at which the system can perform are displayed below:

PARAMETER	SPECIFICATION	DETAILS
Set Temp. Range	0°C to 100°C	Range of temperatures that the system is able to operate. Ideally, the system should be able to perform within this range.
Oscillation Range (Steady State)	1°C	When setting a target temperature, this is the maximum error allowed in the temperature oscillations.
Voltage Limit	5V	Maximum operating voltage of the fan.
Current Limits	1.5A	Maximum operating current drawn from the regulator.

3 System Description

The overall design behind this system is based around temperature regulation of a PTAT, which is a temperature sensing device. The idea behind this system is to maintain a constant temperature by varying the PWM of a fan. Using a potentiometer, the

voltage going into the ADC of our microcontroller can be adjusted, which will determine the set temperature of the system. Based on this set temperature, the fan will either turn on, if the temperature is lower than the current temperature, or turn off if the opposite is true. To determine how close the temperatures are, the output of the PTAT is also input into the microcontroller and converted to a temperature reading. Once the actual temperature approaches the set value, the fan will adjust its PWM in order to keep the value as steady as possible. A 5V regulator is used to produce heat, which serves the purpose of heating the PTAT to a higher temperature.

As for the control scheme, PID control was used for this system. This type of control scheme was used in order to stabilize the system and provide the most accurate functionality possible. All three components of the PID control (proportional, integral, and derivative) had to be adjusted separately in order to ”tune” the system. All three components have their own constant, which we set using our simulated values in order to produce the most ideal behavior. The Simulink block that was used for simulation can be seen below or on the GITHUB.

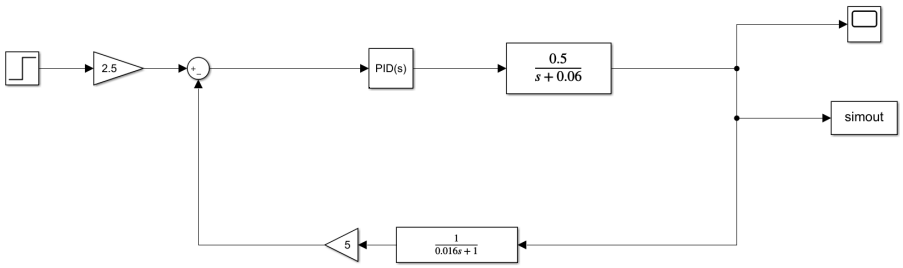

Figure 1: Simulink File Used for Simulation

This simulation was primarily used for determining the PID constant values. These simulations showed the different outputs as each individual variable was changed. Some would impact overshoot more while others would impact settling time. These simulations resulted in us choosing values that would make our system have a balance between overshooting, taking time to settle, and steady state error. This was done because it makes it easier for the user to understand what is happening in the system. Although later more simulations were run in order to decrease the overshoot and settling time as it would not make the system very usable if it overshot the target by 15C or took 3 minutes to get to a desired temperature.

3.1 Schematics and Implementation

This section contains the system constructed on a breadboard. Along with this a cone was constructed for the fan to ensure that it was concentrated on the regulator. This was wanted to ensure that the system would cool as quickly as possible. The detailed circuit schematics can be seen later in this report.

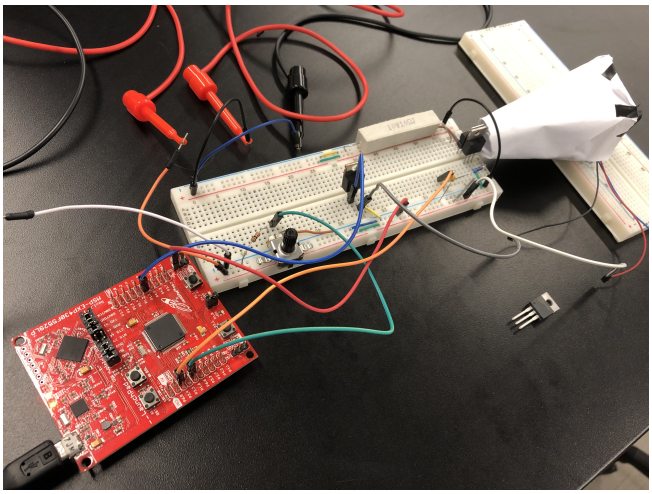

Figure 2: Picture of System Constructed on Breadboard

3.2 Detailed Block Diagram

The following figure shows the detailed block diagram of the temperature control system. Each important component is discussed following the diagram.

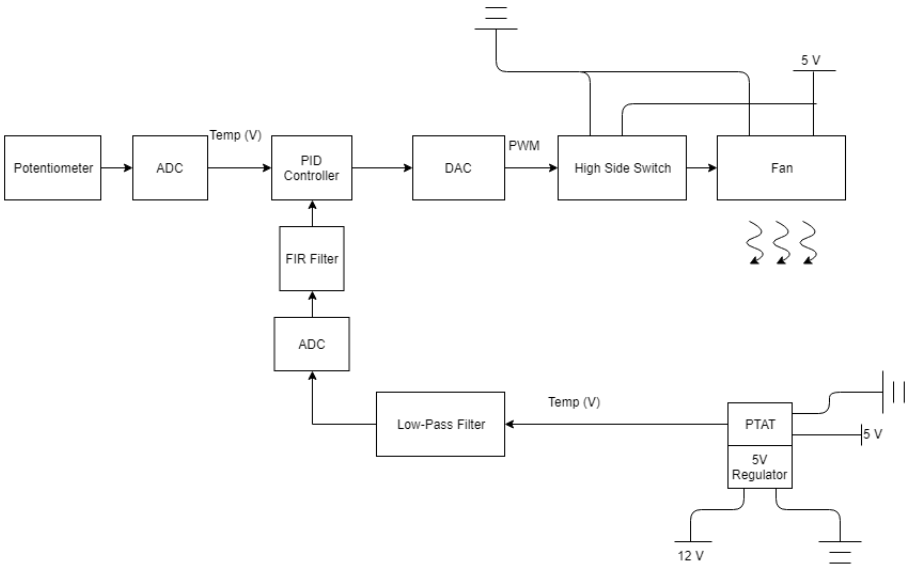

Figure 3: Detailed Block Diagram

3.2.1 PID Controller

The PID controller controls the temperature of the system by changing the PWM signal being sent to the low side switch. The controller takes the current temperature being sent from the PTAT and the set temperature being sent from the potentiometer and computes the error. It then computes the integral and derivative of the error signal to be used in setting the correct PWM. The error signal, its derivative and its integral are each multiplied by constants which determine their individual weight in computing the new PWM. These constants were the values determined in the Simulink simulations.

3.2.2 Potentiometer

The potentiometer is a variable resistor which is used to set the desired temperature of the system. The potentiometer takes 5 volts as its source and is connected to ground. The output pin of the potentiometer is connected to an analog to digital converter on the microcontroller. The highest possible value from the potentiometer is 5 volts and the lowest is 0 volts. In the microcontroller, the signal is multiplied by 30 to convert it to a temperature. This allows a desired temperature range of 0 °C to 100 °C.

3.2.3 FIR Filter

The FIR filter is a digital filter that was coded into the project. This filter uses the 10 most recent temperatures and multiplies them by a weight. The newest temperature has the highest weight while the oldest has the lowest weight. All of the weights added up needed to be equal to 1 to ensure that the temperature was not being scaled up in the code. Once these new numbers are determined they are added up to determine the current temperature. This is done so if the PTAT has an error where it reads an incorrect temperature it will not affect the system immediately. If the number in question was an error then the filter ensures the measured temperature will stay at the correct value, due to the 9 other temperatures being correct. The weight on the error would be slowly decreasing which again would lessen any impact. Although, if a jump was desired then the large or small values would repeat which would then begin to change the current temp over time. These jumps will take slightly longer due to the filter although the system needs time to change temperature so it is usually neglected.

3.2.4 Low Pass Filter

The low pass filter is an analog filter that is constructed using a 47 nF capacitor and a 1 kΩresistor. This filter removes the noise from the temperature signal measured by the PTAT. This was done to try and limit any errors that would enter the FIR filter. This allows the system to run as intended consistently.

3.2.5 Low Side Switch

The low side switch is used to change the strength at which the fan blows. The 5 volt fan used in this project does not have a PWM input so a low side switch had to be constructed to change its strength. The low side switch is made using a power transistor to ensure that it could handle the 5 volt range needed for the fan. The PWM outputted by the PID controller is connected to the gate of the power transistor with ground attached to the source and 5 volts attached to the drain.

3.2.6 PTAT

The PTAT is a linear temperature measurement device. This device outputs a voltage that is linearly proportional to the current temperature of the device. The device outputs 250 mV at a temperature of 25 °C and gains 10 mV for every °C that it gains. The temperature signal outputted from the PTAT is then sent to a low pass filter before being passed through an analog to digital converter. This digital signal is then passed through a digital FIR filter before being converted from a voltage to a temperature by the microcontroller.

3.3 Highlighted Devices

In order to complete this project the MSP430F5529 is used as well as a LM35DT PTAT, 5 volt fan, and an 10k potentiometer. These components each are used in different parts of the lab explained above. Their detailed use in this circuit is explained in the next subsections.

- MSP430F5529LP
- LM35DT PTAT
- 5 Volt Fan
- 10k Potentiometer

3.3.1 MSP430F5529

The MSP430F5529LP acts as a PID controller for the system. It accepts input voltage from the potentiometer and converts it to a temperature. This temperature acts as the set point of the PID controller. It also reads the output voltage of the PTAT and converts this to temperature. The current temperature and set temperature are used by the PID controller in the MSP430F5529LP to set the speed of the fan through a PWM output which cools the PTAT.

3.3.2 LM35DT PTAT

The LM35DT PTAT is used to measure the temperature of the 5 Volt regulator. The LM35DT has a linear voltage output with respect to its temperature in °Celsius. The linear output of the PTAT makes it ideal for this application since the large logarithmic calculations needed for a thermistor take a long time for the micro controller. The PTAT provides more efficient performance and more accurate temperature readings.

3.3.3 5 Volt Fan

The 5 volt fan is used to control the temperature of the system by cooling the PTAT. The speed of the fan is controlled by the micro controller by converting the output of the PID to a PWM signal. This fan was chosen because it is a good size and has a good maximum strength for cooling the PTAT.

3.3.4 10k Potentiometer

The 10k potentiometer is a variable resistor which outputs a voltage with respect to the position of a knob. This voltage can range from 0, when the knob is closed, to the

source voltage, when the knob is open. The potentiometer is used to set the desired temperature of the system by adjusting the know. This potentiometer was chosen because it is accurate in the voltage range used by the system.

4 SYSTEM DESIGN THEORY

This section will discuss the overall design process that led to the completion of our project. This includes the Open Loop design, the initial control design, the iterative steps taken to tune the system, and the final design.

4.1 Open Loop System Design

For the open loop system design we determined the voltage outputs of the potentiometer and the PTAT. Since the PTAT is a linear device, converting the voltage to a temperature was fairly straightforward. For the potentiometer, the output voltages were also converted to temperature. These conversions were key for the closed loop feedback system, as the PWM of the fan would depend on the accuracy of these values. The potentiometer voltage was responsible for setting the desired temperature of the system and the PTAT was responsible for measuring the current temperature. The range of the temperatures on the potentiometer varied from 0 to 100 degrees Celsius.

The open loop design was key for the operation of the overall system. Without having characterized it and made the correct conversions, the system would not have operated as desired. This is due to the fact that the fan speed is completely reliant on these two temperature readings and the difference between them. For example, if the set temperature was lower than the actual measured temperature, the fan should operate at full speed until it approaches the value. Then, the fan's PWM should decrease in order to avoid overshoot and maintain the target temperature. This same behavior is true in the opposite direction. If the set temperature is higher than the current temperature, the fan should turn off and allow the voltage regulator to heat the temperature sensor. Once it is heated to the target, the fan will come on to keep it at that level.

4.2 Initial Control Design

The initial control design used was a PID controller using an Arduino Nano. This system featured a low pass filter at the output of the PTAT and utilized a PID package in Arduino. The system also took the input from a potentiometer in order to set the target temperature for the PID system. However, due to soldering issues on the Arduino Nano the PWM output to the fan was not clean enough to be used for this design. Also, the Arduino interface did not allow us to debug the system to the level we needed, which is why we chose to utilize the MSP430F5529 for our design.

4.3 Iterative Steps

To fix the issue with the Arduino Nano an MSP430F5529LP was used to act as the PID controller instead. After switching boards it was found that the digital readings from the PTAT were not very steady and had large jumps at random intervals. To fix this issue, an FIR filter was implemented in the microcontroller. This created smoother values for the current temperature which resulted in a better calculation of error from the PID controller.

The current design for the system had a very large overshoot, a very large settling time and had a very large steady state error. To fix this, the gains for the derivative and integral controls were changed. The derivative gain was increased to increase the response time of the system and reduce the overshoot. The integral term was reduced to prevent the steady state error and reduce it to 1 °C. These constants were tuned in until the system performed as close to ideal as possible. We continuously changed the values and tested the system behavior until we were satisfied with the result. The values that were changed each time were determined from the simulations performed with Simulink. We were lowering the time in the simulation while also lowering the overshoot. These new values needed to be tested each time as they needed to account for the systems error and the time taken for the temperature to change.

With this design the fan was still rather slow on cooling down the system and would sometimes be on when it needed to allow the system to heat up. To fix this, the PID controller was only activated when the error was less than 3 °C. This ensured that the fan would be fully on or off to correct for large amounts of error very quickly. With the combination of these alterations, the system performed as desired.

4.4 Final Design

The final design for the temperature control system uses two filters, a microcontroller and a fan. The first filter is an analog low pass filter put at the output of the PTAT to filter noise of the current temperature signal. The second filter is a digital FIR filter that is implemented within the microcontroller. This filter is used to create a very clean input to determine the error for the PID controller. To compute the error, the microcontroller must also have a desired temperature. This is done by turning the knob on a potentiometer which is scaled between 0 °C and 100 °C and connected to the microcontroller. The error is then sent to the PID controller where it computes the derivative and integral of the signal. The signal, as well as its derivative and integral are each multiplied by constants and then summed. This signal is then used to set the PWM of the fan to appropriately cool the system. By default, if the error is greater than 3 °C, the fan is either set fully on or fully off.

5 Code Review

This section will cover the main portions of the code used in the micro controller. It will discuss details of certain aspects of the code while also explaining the theory in certain sections. The base of this code was taken from a similar code that was used for temperature regulation and was written for a previous class by all the authors of this report. After this code was taken it needed to have a PID controller implemented in the control of the fan's PWM. Along with that an FIR filter was coded as well to further the accuracy of this system.

5.1 Initialization

This section simply covers the basics initialization of the code for this project. The first of these declarations included the PID constants which can be seen below as Kp, Ki, and Kd respectively. These were the constant values obtained from simulations discussed earlier in this report. Later on throughout this section comments may be cut off, the full comments may be viewed in the GITHUB link provided in this report.

```
const float Kp = 0.5;
const float Ki = 0.07;
const float Kd = 0.3;
```

After these values were declared then some simple variables were defined and assigned values. A few of the main variables from this portion include the MaxOut and MinOut which controlled the maximum and minimum duty cycle of the fan. Along with these all of the weighted constants were set for the FIR filter, later in the code the reasoning for these values will be described with the FIR section of the code.

```
const int MaxOut = 998, MinOut = 0; // Maximum/Minimum Duty Cycle
float Up;
int Out;
int uart_count = 0;
int ADC_count = 0;
float k1 = 0, k2 = 0, k3 = 0, k4 = 0, k5 = 0, k6 = 0, k7 = 0;
float k8 = 0, k9 = 0, k10 = 0, y;

//ai constants set for FIR filter
const float a1 = 0.25;
const float a2 = 0.20;
const float a3 = 0.15;
const float a4 = 0.1;
const float a5 = 0.05;
const float a6 = 0.05;
const float a7 = 0.05;
const float a8 = 0.05;
```

```
const float a9 = 0.05;
const float a10 = 0.05;
```

Some more variables were created and defined after this, but the next big initialization was that of the timer. This was done to set certain pins to output direction while also enabling the timer A. Timer A will be used to constantly updated the PWM of the fan from the output created from PID controller. This basic set up was taken from a previously written code that also utilized a timer A for a similar purpose of temperature regulation.

```
void TimerA0Setup(){
    //Set P1.0, P4.7, P1.2 to the output direction.
    P4DIR |= BIT0;
    P4DIR |= BIT7;
    P1DIR |= BIT2;
    P1SEL |= BIT2;
    TA0CCTL1 = OUTMOD_7;          // Enable interrupt in compare mode
    TA0CCR0 = 999;                // Set period of CCR0 1000 microseconds.
    TA0CCR1 = 500;                // Set duty cycle to 50%
    TA0CTL = TACLR;               // Clear flag
    TA0CTL = TASSEL_1 + MC_1;     // TimerA0 Control: SMCLK, UP Mode
}
```

Finally, there us quite a bit of UART initialization. Although these were taken from a sample code found through googling. The purpose of those lines of code is to allow the set and actual temperatures to be sent over UART for the user to view.

5.2 Sensor Acquisition

For this portion the code that involves the sensors will be discussed. The major sensors involved with this system were both the PTAT that was measuring the current temperature along with the potentiometer which was creating the set temperature. The code that implemented both of these can be seen in the listing below.

```
// Temperature calculations
    pot_Voltage = ADC12MEM2;     // Stores Potentiometer voltage in MEM2 of
    temp_Voltage = ADC12MEM0;    // Stores PTAT voltage in MEM0 of ADC

    pot_Voltage = ((pot_Voltage*3.3)/4096);      // Calculation for potentio
    setTemp = ((pot_Voltage)*30);                // Conversion from PTAT rea
    temp1 = ((temp_Voltage*3.3)/4096);
    temp1 = (temp1*100);
```

The first line takes the converted value from the ADC that was measured from the potentiometer and sets it equal to a variable. The potentiometer voltage has to be

converted according to a formula found in the data sheet for this specific potentiometer. Once this was done it was multiplied by 30 in order to properly display the set temperature from 0-100°C. We knew the final voltage had to be multiplied by to equal a temperature value although the final value of 30 was determined by trail and error.

The PTAT voltage that was converted in the ADC was set equal to another variable. This value needed to also be converted to temperature similarly using another equation found on the data sheet for this specific PTAT. Once this was done the resulting voltage needed to be multiplied by 100 to output the temperature measured. Now that both the set and actual temperatures are able to be read and displayed these values need to be used in order to control the system and make the system perform as intended.

5.3 Error Calculation

This section discusses how and why the FIR filter was implemented and how the error was calculated for this system. The FIR filter was implemented to ensure that the system would not have random spikes in temperature. This is due to the filter taking the past 10 temperatures and weighting each one of these and adding them together. This prevents spikes in temperature from glitches as the glitch will be going against 9 correct values each weighted differently. These weights were the values assigned to the variables in the subsection about initialization. The newest temperature has the highest weight and it decreases as they get older. One important note is that the weights of all of the temperatures must add up to 1 as to not incorrectly amplify the temperature reading. The code for the FIR filter can be seen in the following listing.

```
// Samples shifted up by 1 and temp1 is shifted in
   k10 = k9;
   k9 = k8;
   k8 = k7;
   k7 = k6;
   k6 = k5;
   k5 = k4;
   k4 = k3;
   k3 = k2;
   k2 = k1;
   k1 = temp1;

   // FIR filter input: x  output: temp
   temp = (k1*a1) + (k2*a2) + (k3*a3) + (k4*a4) + (k5*a5) + (k6*a6)
   + (k7*a7) + (k8*a8) + (k9*a9) + (k10*a10);
```

The code shows how the FIR filter was implemented into this specific code. It is very similar to the description mentioned before as each temperature is weighted and once weighted they are all added up to get the current temperature. This may seem like

a simple filter design although it is very crucial to ensuring that the system does not randomly spike and trigger the fan to turn on. Along with this filter this section will discuss how the errors were calculated to be used later in the code. These errors will be important during the PID control of the system. The code for these two error calculations can be seen below.

```
p_error = error;        // Track previous Up and Current Up
error = temp - setTemp;        // Calculation for the error
```

This code calculates error in a very standard way of subtracting the actual value from the set value. This is repeated for every value that comes through the system;however, before the new value is calculated a variable is set equal to that of the previous error. This is required for the implementation of the PID controller which can be seen in the next subsection.

5.4 Control Calculation

This section of the code consists of the most important aspect of this system the PID controller. The PID controller is what calculates the new PWM for the fan with every new temperature reading. It needs to calculate Up, Ui, and Ud to be used in the actuating section of this report. The first of these is the Up which is the calculation for the proportional control this simply take the error calculated from the current temp minus the set temperature and multiplies it times the Kp value set previously from the simulations.

After this the Ui was calculated, rather than do integration in the micro controller a rolling sum was used. The Ui takes takes the error and multiplies it by 0.025. This was done in an example code we found on the internet. The Ki value determined from the simulations is then multiplied by this. Once this is calculated it is added to the previous calculation in order to act as an integral term. Also, the Ui term is set back to 0 when the set temperature changes by more than 3 degrees. This was done due to reduce the time taken to calculate the new Ui.

Finally, the Ud value needed to be calculated. In the previous subsection about error it was mentioned that not only was the error calculated but the previous error was also being stored. That previous error is required for the derivative control. We did not want to perform derivations inside of the micro controller so we took the difference between the error and the previous error then divided by 0.025. The same code was referenced and used 0.025 so it was used for our code. Finally, that was multiplied by the Kd constant which again was determined from the simulations of the system. The code implementation of all of this can be seen in the following listing.

```
p_error = error;                    // Track previous Up and Current Up
error = temp - setTemp;             // Calculation for the error
Up = Kp * error;                    // Calculation for Proportional Control
Ui += (Ki * (error * 0.025));       // Calculation for Integral Control
Ud = Kd * (((error - p_error)) / 0.025); // Calculation for Derivative
```

These calculated values of Up, Ui, and Ud will be used in the next section to actuate the PWM of the fan. These variables may seem to have random calculations to determine them but they all have a specific purpose. The Up has to deal with the current steady state error of the system. While the Ui deals with the magnitude of this error and the duration of it. The Ud deals with the settling time of this error. All of these calculated together allow a system to be controlled to very specific specifications.

5.5 Actuation

Finally, this section will cover how the system actually actuates and changes the PWM of the fan. The PID controller and filter have all been part of a function updatePWM. The previously discussed Up, Ui, and Ud are added up and multiplied by 30 in order to give the output. Now that the output (Out) was determined from the PID controller it can simply be returned from this function. This function is then set equal to the CCR1 in the micro controller which was initially setup in the Timer A. This will set the output calculated from the PID controller equal to that of CCR1 which is the PWM of the fan. The listing for all of this code can be seen below.

```
Out = (Up + Ui + Ud)*30;
  return Out;

#pragma vector=ADC12_VECTOR
__interrupt void ADC12ISR (void)
{
    TA0CCR1 = updatePWM();
}
```

This interrupt changes the CCR1 value for each time the PWM needs to be updated with the newly calculated value from the PID controller. When all of this was implemented the system was performing as intended. Although, it took long periods of time for the system to heat up as the fan would never fully turn off. In order to counteract this we implemented a simple bang bang control along with the PID. This bang bang would turn the fan completely off if the set temperature was greater than 3 degrees of the actual temperature. The code for this is listed below.

```
if(setTemp-temp>3){
    Out = MinOut;
} else if (setTemp-temp<-3){
    Out = MaxOut;
} else {
    Out = Out;
}
```

This final piece of code made the system function in a more user friendly manner. The time to heat up and cool down the system was now greatly decreased while also keeping the full PID intact to be able to keep the temperature steady. The CCR1 value

is then entered into the low pass filter on the breadboard which then allows it to control the PWM of the fan. This was due to the fan not having a specific PWM pin rather we needed to modify the voltage entering the fan to control its speed.

6 Design Files

6.1 Schematics

The Schematics for the circuit used in this system can be seen in the following four figures. The first of these shows the circuit created to control the PWM for the fan. This was done by creating a low side switch. This switch allows the fan to be left floating (off) when the regulator is at the desired temperature.

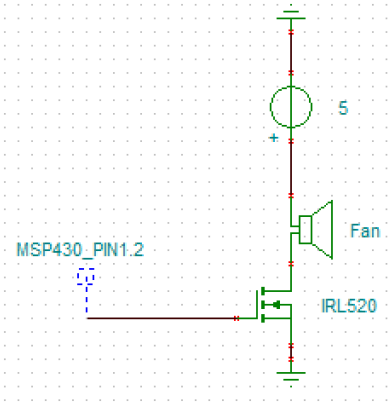

Figure 4: Fan Schematic

The input to the low side switch gate for the fan control is pin 1.2 on the MSP430 micro controller. The next schematic shows the circuit used to read the voltage from the PTAT. This can be seen in the following figure.

The PTAT model shows the voltage input being 5V from the source. The middle pin connects to common ground. While finally the voltage output is connected to pin 6.0 on the micro controller. This is passed through a low pass filter before it is connected to the pin though. This is done to ensure the temperature values from the PTAT do not spike randomly from errors but rather increase or decrease at normal rate. This value is then put into the ADC to determine the temperature of the system. The next circuit created is the circuit to heat the voltage regulator. The circuit for this can be seen in the following figure.

The voltage regulator circuit is the simplest of all the circuits for this system. This simply connects the 5V regulator to a 12 volt power supply and the common pin to ground.

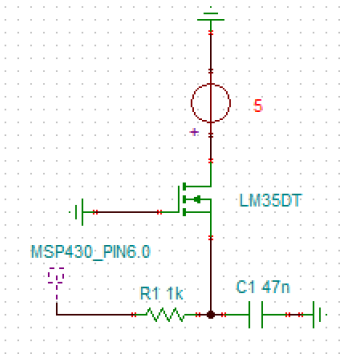

Figure 5: PTAT Schematic

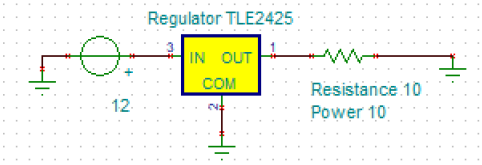

Figure 6: Regulator Schematic

Finally, there is a power resistor connected to the output of the regulator to ground. The power resistor is a 10 Ohm 10 Watt power resistor. The final circuit created was the portion that controlled the temperature of the system with a potentiometer. This can be seen in the following figure.

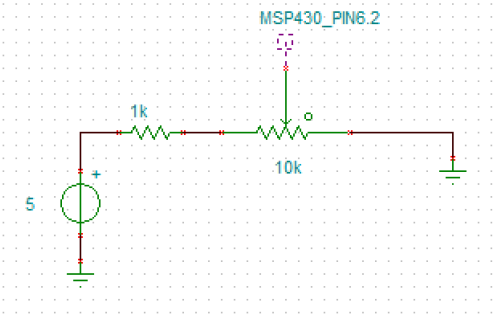

Figure 7: Potentiometer Schematic

The potentiometer circuit uses a 1k ohm resistor to connect the potentiometer to the 5V source. Then the micro controller pin 6.2 is connected across the potentiometer. This pin goes to another ADC to allow the temperature to be set for this system by turning the potentiometer.

6.2 Bill of Materials

- 1 MSP430F5529 Micro Controller
- 1 10k Potentiometer
- 1 1k Ohm resistor
- 1 10 Ohm 10 Watt power resistors
- 5v Regulator
- 1 LM35DT PTAT
- 1 IRL520 Power Transistor
- 1 5V Fan
- Power supply capable of 12V output

6.3 Code

Our code can be found on GitHub at the following link: GitHub Code