Milestone 2 - Closed Loop Cooling System

Ian Moffitt and Joey McNatt
Rowan University

December 24, 2018

1 Design Overview

This program is designed to get a system to a desired temperature and maintain it. It is a closed loop system, meaning that it takes an input generated by a sensor (in this case a temperature sensor), and actuates based on that input in order to change the measured quantity. This program is designed for use with the TI MSP430F5529. The goal of the project is to obtain a desired temperature, read the current temperature, and output a variable PWM on a fan to reach/maintain the desired temperature within 3C.

1.1 Design Features

- Ability to control temperature of a device via fan PWM
- Desired temperature sent via UART
- Current temperature received via UART

1.2 Featured Applications

- Cool device down to a specified temperature
- Heat device up to a specified temperature
- Stabilize device within 3C of specified temperature

1.3 Design Resources

All necessary resources can be found in the project GitHub repo at:
`https://github.com/RU09342-F18/introtoembedded-f18-milestone2-gangplank-galleon-gang`

1.4 Block Diagram

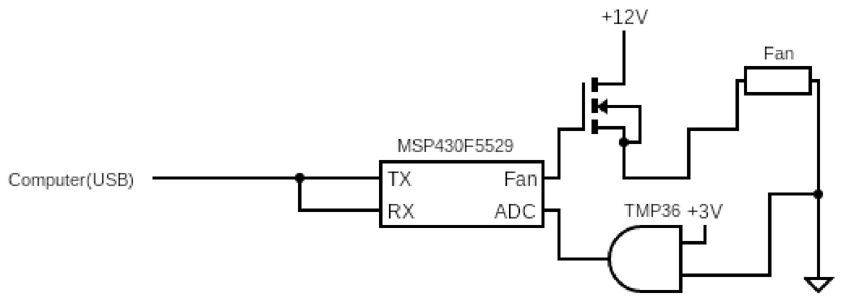

Figure 1: Basic Diagram Showing Major Connections

1.5 Board Image

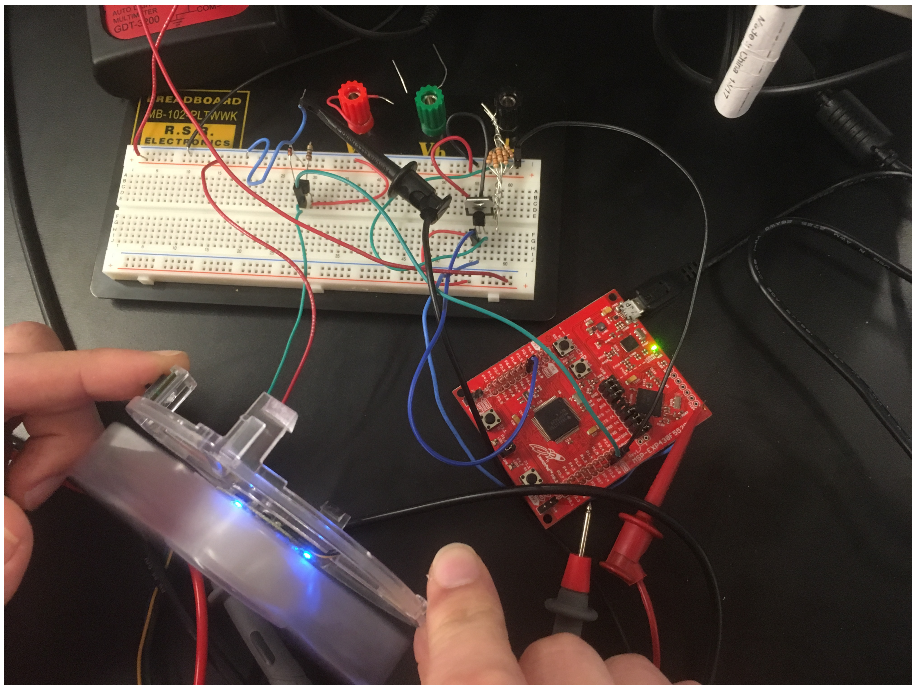

Figure 2: Full Circuit Setup Showing Connections

2 Key System Specifications

Parameter	Specification	Details
Processor	MSP430F5529	This processor was selected due to its general ease of use and 12 bit ADC.
Communication Protocol	UART	9600 baud
Max Temperature	∼60C	This value is dependent on how much current is drawn from the 5V regulator.
Min Temperature	∼25C	This is the temperature the system gets down to with the fan left on 100%.
Temperature Fluctuation	3C	Limit for change once a stable temperature has been reached.

3 System Description

When electrical components are being used in a system, a lot of them generate heat. This can cause problems with the components themselves, or other components in the system. It is often desired to keep components at or below a certain temperature for optimal performance, however it can sometimes be annoying or power consuming to keep cooling solutions powered on all the time, so it is best to use them only when necessary. This can be facilitated through an embedded system, and will be explored in this project, where the overall goal was to keep the temperature of a heated component stable at a specified temperature with minimal fluctuation above or below the desired temperature. This way, the system could cool to a certain level without keeping the fan on constantly once it got there, or the system could be allowed to heat to a certain level without being allowed to overheat.

3.1 Detailed Block Diagram

(1).png

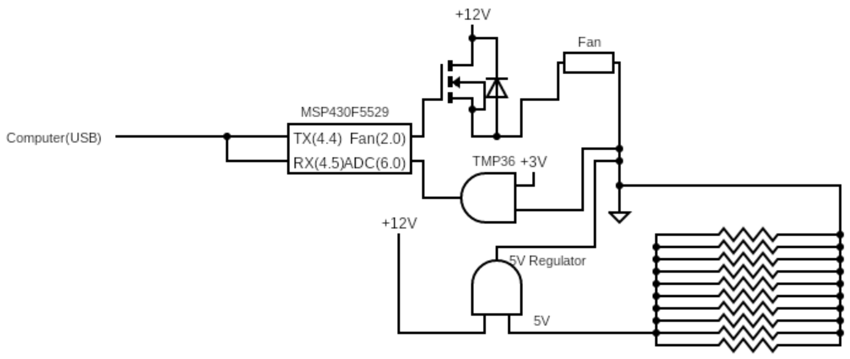

Figure 3: Detailed diagram showing connections (port specific) and heating element. A flyback diode is added across the NMOS to protect the board and computer.

3.2 Highlighted Devices

- MSP430F5529 - Accepts target temperature over UART, reads temperatures using ADC, and PWMs a fan.
  User Guide: `http://www.ti.com/lit/ug/slau208q/slau208q.pdf`
  Datasheet: `http://www.ti.com/lit/ds/symlink/msp430f5514.pdf`
- TMP36 Temperature Sensor - Outputs a voltage to be read by the ADC.
  Datasheet: `https://www.analog.com/media/en/technical-documentation/data-sheets/TMP35_36_37.pdf`

3.3 MSP430F5529

The MSP430F5529 is the main control unit for the rest of the system. It was selected due to its general ease of use compared to some of the other boards available. It also has a built in 12 bit analog to digital converter (ADC). It has built in UART functionality for receiving desired temperatures, and timer modules capable of producing a PWM square wave on its I/O pins.

3.4 TMP36

The TMP36 is a proportional to absolute temperature (PTAT) sensor that outputs temperature information as a linear voltage. With a supply voltage of 3V, the device produces 10mV/$^\circ$C with a 500mV offset (750mV at 25C). This stays well under the MSP430F5529's 2.5V reference voltage for the ADC even at high temperatures, making it an excellent choice for this application. This device was chosen over a thermistor due to its ease of use. While a thermistor is non-linear, this device is linear and therefore does not require characterization.

4 System Design Theory

The system operates using 3 major components; the board itself, the PTAT, and the fan. The board essentially takes a reading from the PTAT on a timer, converts it to the temperature, compares it to the desired temperature, and does something with the fan based on the difference between the two. How the system functions will be detailed in the following subsections.

4.1 MSP430F5529 Setup Code

This section will be dedicated to all the pieces of setup code necessary to get the system functioning. First is the code to set up the ADC:

```
void ADCSetup(){
    REFCTL0 &= ~REFMSTR; //reset REFMASTER to hand over
        control of internal reference voltage to ADC_12
    //initialize control register ADC12CTL0
    ADC12CTL0 = ADC12SHT0_4 + ADC12REFON + ADC12REF2_5V +
        ADC12ON;
    //initialize control register ADC12CTL1
    ADC12CTL1 = ADC12SHP + ADC12SSEL_3;
    //set control register of conversion memory
    ADC12MCTL0 = ADC12SREF_1 + ADC12INCH0;
    //configure the port to be used as an ADC input
    P6SEL |= 0x01;
    //enable interrupt
    P6DIR &= ~BIT0;
    ADC12IE = 0x01;
    //enable conversion
    ADC12CTL0 |= ADC12ENC;
    //start sampling
}
```

The reference voltage of the ADC is set to 2.5V to ensure the PTAT voltage cannot go over the reference voltage, even though a 1.5V reference would provide greater accuracy. With a 2.5V reference and 12 bits, the system is accurate to .00061V, which is more than accurate enough given that a degree change in temperature is represented by 10mV on the PTAT. This system really only needs to be accurate to about a degree to work correctly, so this is plenty of wiggle room. The ADC interrupt is also enabled to make it easy to perform actions when a sample is taken, and conversion is enabled at the end because the other control registers cannot be modified while conversion is enabled. The ADC is set up to sample on pin 6.0.

Next is the UART setup code:

```
void UARTSetup(){ //configure UART for 9600 baud
    UCA1CTL0 = UCMODE_0;
    UCA1CTL1 = UCSWRST;
    UCA1CTL1 |= UCSSEL_2;
    UCA1BR0 = 104;
    UCA1BR1 = 0;
    UCA1MCTL = UCBRS_1;
    UCA1CTL1 &= ~UCSWRST;
    UCA1IE |= UCRXIE;
}
```

This function works to configure the UART control registers to produce a baud rate of 9600. It also sets the serial mode to UART, and enables the UART interrupt to enable functionality as soon as a new temperature byte is received. It should be noted that UCA1 is used as opposed to UCA0 to make the device USB compatible through the Launchpad.

The following code sets up the timers:

```
void TimerSetup(){//sets up a timer for PWM and a timer for
    sampling
```

```
        TA1CTL = TASSEL_2 + MC_1 + TACLR;
        TA1CCTL1 = OUTMOD_3; //places timer in Reset/Set mode
            for easy PWM
        TA1CCR0 = 100;
        FanControl = 100;
        TA2CTL = TASSEL_2 + MC_1 + TACLR + TAIE;
        TA2CCR0 = 100;
}
```

Timer A1 is used for a PWM output for the fan, and is configured in up mode with CCR0 set to 100 to make it easy to give PWM as a percent duty cycle. It is also placed in Reset/Set mode so that it just provides a PWM output with no further coding necessary. The variable FanControl is defined as TA1CCR1, and is initially set to 100 to have the fan off. Timer A2 is interrupt enabled, as it is the source of the ADC's sampling, as will be shown later.

The rest of the setup is completed in the main function:

```
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;    // stop watchdog timer
    P4SEL |= Recieve + Transmit;     //set up Port 4 pins as I/
        O for USB UART
    P2SEL |= FanOut;     //set up Port 2 PWM pin as I/O
    P4DIR = Transmit;    //set TX pin as an output pin
    P2DIR = FanOut;      //set fan PWM pin to output
    P2OUT = 0;

    TimerSetup();
    UARTSetup();
    ADCSetup();
    __bis_SR_register(GIE); //enable global interrupts
    while(1)
    {

    }

}
```

The setup seen in main begins with stopping the watchdog timer, then proceeds to set up the pins needed to supply functionality outside of the chip itself. TX and RX are on pins 4.4 and 4.5, respectively, and the fan PWM is found on pin 2.0. It then calls the functions shown above to set up the other components, enables global interrupts, and enters an endless loop.

4.2 MSP430F5529 Running Code

The first thing the system needs is target temperature, which it obtains through the UART interrupt:

```
#pragma vector = USCI_A1_VECTOR
__interrupt void USCI1RX_ISR(void)
{
    target = currentByte;
    transferByte = target;
}
```

In this interrupt function, currentByte is a macro for UCA1RXBUF amd transferByte is a macro for UCA1TXBUF. Essentially, the system takes a target value from UART and stores it in a global variable, and echoes it back to let the user know that the interrupt fired.

Next are the pair of interrupt vectors for Timer A2, which cause the system to take a sample:

```
#pragma vector=TIMER2_A0_VECTOR
__interrupt void TIMER_A0_ISR()
{
    ADC12CTL0 |= ADC12SC; //start sampling and conversion
}

#pragma vector=TIMER2_A1_VECTOR
__interrupt void TIMER_A1_ISR()
{
    ADC12CTL0 |= ADC12SC; //start sampling and conversion
}
```

This process causes the ADC12 interrupt to go off, which is where the system's heavy lifting takes place:

```
#pragma vector=ADC12_VECTOR
__interrupt void ADC_ISR(void)
{
    adcIn = ADC12MEM0;
    currentTemp = convertTemp(adcIn);
    changeFan(currentTemp, target);//switches fan speed based
        on last and current value
    char out = (char)currentTemp;
    transferByte = out;
}
```

First, the system loads the current ADC reading into a variable, adcIn, and passes that variable into a function to convert it to a temperature. This is a simple linear function, so the code will not be provided here. It can be found in the GitHub repo. It then runs the changeFan function (casted to a char to omit the decimal), which is the most important part of the system, and will be explained next. It then outputs the current temperature reading.

The changeFan function:

```
void changeFan(float current, int target){
    if(current >= target){
        if (current - target > 1){
            FanControl = 0;
        }
        else if(current - target <= 1){
            FanControl = 50;
        }
    }
    else if(current < target){
        if(target - current > 1){
            FanControl = 100;
        }
        else if(target - current <= 1){
            FanControl = 50;
        }
    }
}
```

This function takes in the current temperature (in the form of a float) and the target temperature (in the form of an int). The target temp is an int due to it being easier to transmit over UART, and because the system doesn't need to be accurate to more than a degree C. The function itself is actually quite simple. If the temperature difference is more than 1C, the fan either goes completely on, or completely off. When the temperature gets within 1C of the target, the fan cuts down to 50 percent duty cycle. This variable nature allows the system to fluctuate a small amount, but it works equally well for any temperature within the range that the system can handle.

5 Getting Started/How to use the device

To set the device up for use, ensure that pin 6.0 is connected to the output pin of the PTAT, pin 2.0 is connected to the gate of the fan MOSFET, all grounds (3V source, 12V source, and MSP430G2553) are tied together so the voltage is from the same reference. Make sure the USB port of the Launchpad is connected to the computer's USB port.

5.1 Communicating with the Device

Once the device is started, open its port in your serial software of choice (in testing RealTerm was used). Send down a target temperature. It should be echoed back, indicating a functional connection. The device should then begin spitting out temperature readings.

6 Test Results

In testing, the system performed exactly as it was intended to. For a heat source, a 5V regulator was placed in parallel with the fan (but not the MOSFET) to be on a 12V source, with a large bundle of resistors on the end of it to have a low equivalent resistance drawing significant current. Temperatures were stable to about 2C, and the system was able to heat and cool quickly and efficiently.

7 Design Files

7.1 Bill of Materials

- MSP430F5529 Launchpad
- TMP36 temperature sensor
- Breadboard
- Assorted Jumper Wires
- 5V Regulator
- 12V DC fan
- Laptop running CCS and RealTerm
- Resistor bundle (made by Russell)
- NMOS
- Diode