Application Note Template

Tyler M. and Helen P.
Rowan University

December 3, 2018

1 Design Overview

In the following lab, students are required to replicate a cooling system of a computer. The conditions are based in terms of when to turn a fan on or off depending on the desired temperature input from user and on the temperature readings from the temperature sensor. For the first task, students have to program a microprocessor to retrieved the inputs from the thermistor, a temperature sensor, and have it be converted into temperature readings that the users can understand. The second part of the lab requires for students to program their microprocessor to be able to PWM the fan speed in order to cool or heat up to the target temperature from the user. In addition, throughout the process, the current temperature of the thermistor can also be displayed back to the user through UART communication.

Therefore, the microprocessor needs to be able to read the temperature from the thermistor and adjust the fan speed according in order to reach the target temperature from the current thermistor's temperature. Students are required to brainstorm on how to approach these problems, given the constraints on what their microprocessor could physically do. The following report details the methodology and hurdles that were over come in order to complete the project.

1.1 Design Features

The design features are the following:

- Ability to PWM fan speed
  - Automatically adjust fan speed depending on current temperature
- Use the 12 bit ADC (Analog to Digital Converter) to convert the voltage across a thermistor to a digital number

  - Using the ADC value to calculate the value of the thermistor
  - Using the ADC value to calculate the temperature of the thermistor
- UART will return the current temperature every one hundred milliseconds

1.2 Featured Applications

Devices that controls temperature includes:

- Thermostats
- Electronic devices such as computers
- Industrial machinery

1.3 Design Resources

A link to the GitHub repository used for this project can be found directly bellow:

`https://github.com/RU09342-F18/introtoembedded-f18-milestone2-nano-tech`

1.4 Block Diagram

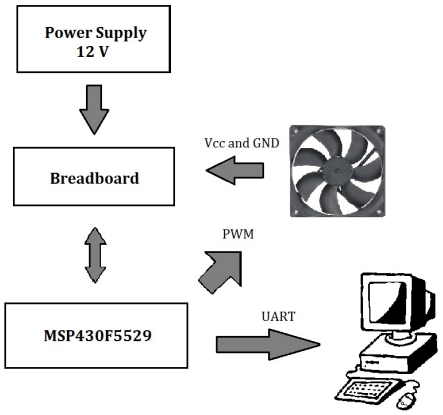

Figure 1: Overall Network of Devices

Figure 1 shows a simple overview of the general idea behind the device. The Power supply runs power to the breadboard, which powers the fan and also houses the voltage regulator. The MSP430F5529 is also connected to the breadboard, and it interfaces with the thermistor on the breadboard. The breadboard itself is segregated to make two sides. The 12 volt side, which has the fan and voltage regulator, and

the 3.3 volt side, which has the thermistor and the MSP430F5529. The MSP communicates to a host computer over UART and can send information like the current fan speed, and the current temperature. The MSP also communicates to the fan to PWM it to the correct speed.

1.5 Board Image

The MSP430F5529 board was shown in figure 2.

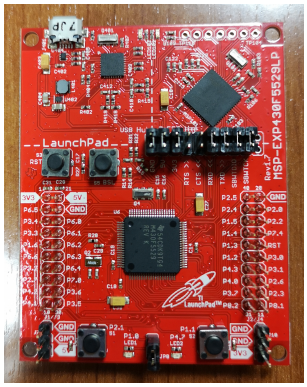

Figure 2: MSP430F5529

In figure 3, the MSP430F5529 was connected to the breadboard using female-male jumper wires. On the breadboard, a 10K ohm resistor is connected in series with the thermistor, a temperature sensor, and two 2 Watt 47 ohms power resistors are connected in parallel to one another. The connection of the fan is based on the colored wires and only three were used: black is ground, yellow is connected to a 12 volt power supply, and blue is for the PWM signal.

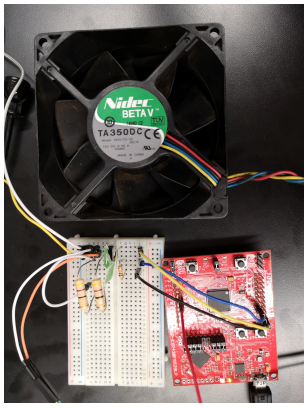

Figure 3: Milestone 2, using MSP430F5529 to PWM the fan speed with temperature

2 Key System Specifications

The most important specifications, the ADC and the voltage to temperature conversion, are featured at the top of the table. While a non-core function like predictive measurements are shown at the bottom.

Parameter	Specification	Details
12 bit ADC	Convert analog signals to digital value	ADC reads in a analog value (the voltage) and converts that to a digital, 12 bit number. This number is later used to calculate the value of the thermistor and then, later the temperature of the thermistor.
Linear approximated thermistor functions	Calculate temperature of thermistor	A linear approximation of the thermistor resistance to temperature calculation. Approximated to save computation time at a very small cost of accuracy (less than 1^{o} Celsius at most)
Timers	Pin 1.2 generates a PWM signal to control the fan	Every time the processor wants to change the PWM rate for the fan it calls the function to do so while the new PWM duty cycle (In percent out of 100) This function creates the new duty cycle for the PWM and immediately sets it into motion
Predictive measurements	Guessing what the next temperature will be before measuring it	The board stores the last 5 measurements taken and uses that to generate a trend line. This is trend line is used

Table 2 Features some of the key design features of the system.

3 System Description

3.1 Problem

The device needs to be able to regulate the temperature of an object (For testing purposes, a voltage regulator was used). This does not just mean that the MSP needs to cool off the device, but it must also recognize when the object has been cooled too much and either stop cooling off the object or not work as hard at cooling it off. This requires the device to closely monitor the temperature of the object and react to changes it detects.

3.2 Solution

The problems that the MSP have to solve are divided into sections and tackled individually. The first problem is knowing the temperature of the object. This was easily solved by attaching a thermistor to the object. A thermistor is a type of resistor whose resistance value changes with temperature. In this case, the thermistor's resistance decreases when ever the temperature increased. Therefore, since the temperature of the thermistor correlates to a known resistance value, the MSP only has to know the value of the thermistor in order to know the temperature. To do this, the thermistor was used in a voltage divider setup.

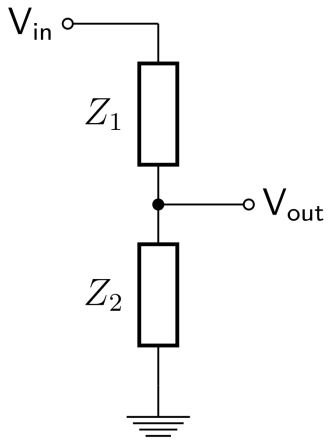

Figure 4: Example of a voltage divider from Wikipedia. Z_1 is the thermistor and Z_2 is the 10K resistor

Figure 4 shows the basic setup for a voltage divider. In this example, V_{out} goes to pin P6.0 on the MSP. Pin P6.0 is connected to the ADC on the chip. This chip uses a 3.3 volt to ground reference and will convert the input voltage to a digital, 12 bit number. This 12 bit number represents the voltage at that point, and since this is just a basic voltage divider, the voltage at that point will change based on the current resistance value of the thermistor. Since the voltage at the point changes with

the thermistor, and the resistance of the thermistor is directly tied to the voltage at the point, and the voltage at the point is known, the value of the thermistor can be calculated.

$$R_{thermistor} = \frac{33000}{V_{out}} - 10,000 \tag{1}$$

Equation 1 shows the formula for getting the resistance value of the thermistor. Since many of the original values cancel at the top and bottom of the fraction, the formula becomes very simple, making it quick to calculate on the relatively slow processor the MSP430F5529 is armed with.

With the resistance value of the thermistor calculated, the only thing left is to use that to calculate the temperature. The temperature of the thermistor corresponds to a logarithmic graph provided on the data sheet and in figure 5.

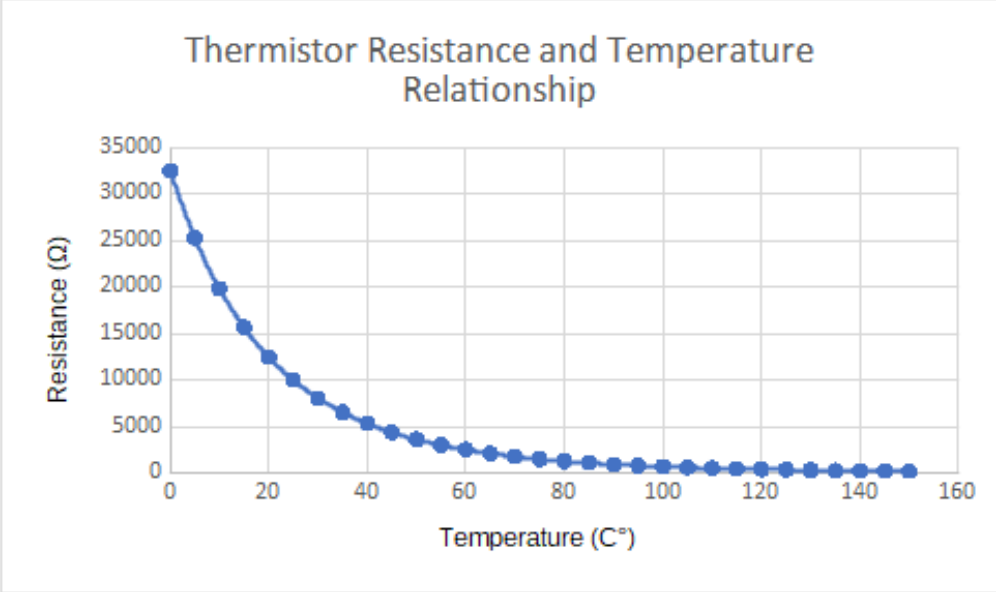

Figure 5: The graph shows how the resistance value of the thermistor changes as it gets hotter.

There are multiple ways to calculate the temperature from the resistance value. The first option is to just derive a logarithmic function from figure 1 to get a function, and then use that function get temperature. This was decided against however, as logarithmic functions are very taxing and take a long time to compute. Another problem with this approach is that additional C libraries would have to be imported to use log functions, and there is very limited RAM on the MSP430F5529.

A second option was to take each value on the graph and make a look up table. This table would then be referenced each time a value needs to be calculated. This was also shot down due to the size of the look up table exceeding the available space in RAM.

A third option was proposed to make a linear approximation of the graph and use that to calculate the temperature. Originally this was seen as a poor idea because a linear approximation if a logarithmic function is inaccurate in all but a few (Sometimes one) place(s). This idea was instead adapted. The function was broken up into 7 separate linear approximations, each covering a separate portion of the graph. When the temperature needs to be calculated, the function first acts as a look up table. The MSP finds which linear approximation is closest to the resistance value it has. Once the closest linear approximation is found, the resistance value is plugged in and calculated. This method proved to be fast, compact, and accurate.

3.3 Detailed Block Diagram

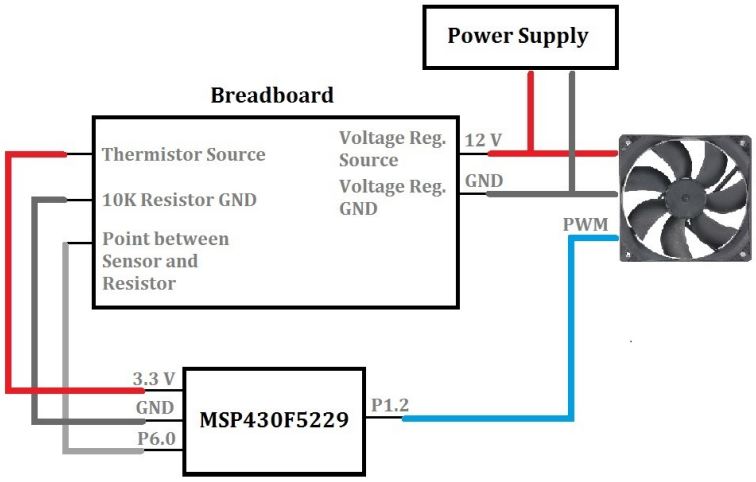

Figure 6: Working on image right now

3.4 Highlighted Devices

- MSP430F5529
  - Essential the ”Brains” of the project. It is the location for where code is compiled and run. In addition, it send controls to PWM the fan speed according to the temperature feedback.

- MSP430 LaunchPad
  - The LaunchPad provides pin outs for numerous useful I/O pins, some of which are used in this project for connecting to ground, power source, and communicating over to UART and ADC.
- Breadboard
  - The Breadboard allows for the connection of numerous electronic components in a neat and organized fashion, without the risk of two components touching.
- Desktop Computer Fan
  - The fan is used to cool down the temperature of the thermistor. The fan port contains four connections, black to ground, yellow to power source, green to signal, and blue to control or PWM.

3.5 MSP430F5529

3.5.1 ADC

The ADC is the backbone of the project. Without it, the temperature would not be able tot be read and then the fan would not be able to be PWM'd correctly. The ADC is used to convert analog signals into a digital number. The ADC used in this device is the 12 bit variant. This ADC by default uses the 3.3V to ground reference, however it can be changed if necessary. For the purposes of this project, the default reference was used. The ADC was setup such that it constantly polls Pin 6.0 for a new voltage reading. It then converts that signal to a digital number and stores it in it's local register. Since the ADC is constantly getting a new sample, it isn't necessary for the code to request the ADC to do so, instead, the code just reads the value of that register, and uses that for it's calculations.

3.5.2 Processor and RAM

The processor is what allows the code to actually be run. The code is constructed such that it calculates the temperature and PWM's the fan every 100 milliseconds. Since the processor has a finite speed, all the instructions must be able to be run within this time frame. This isn't a horribly limiting factor, as the processor is fast enough to do these instructions far faster than the current rate, and the code was written to run efficiently and not take up too much CPU time.

The on board RAM is also a very limiting feature. There is only 8KB of available on board RAM, meaning that the code and all of it's variables and data it manages, must be able to fit inside of this space. writing ”compact” code isn't a challenging task itself, however making sure that all of the information that your code manages can fit inside this space is another problem (This is briefly talked about in section 3). Since

there are numerous logic used in intelligently determining what to set the fan PWM to, the code does take up a decent amount of the available space, however it is not full.

3.5.3 Timers

There are two instances of timers used in this project. The first instance is purely so that the code knows when 100 milliseconds has passed. The program is set up such that every 100 milliseconds it reruns it's routine, this means getting and calculating the current temperature, predicting what the next temperature will be, and setting the duty cycle for the PWM to bring the temperature closer to the target. The second timer is used for the PWM itself. This timer is set up to count to 99 (Because counting starts at 0) and then reset. The sends 0 volts over pin 1.2 until it hits a specified value (The duty cycle %) then pin 1.2 is 3.3 volts. This is because the fan turns off whenever it receives power on the PWM pin, and turns on when there is no power.

3.6 MSP430 LaunchPad

The MSP430 LaunchPad contains the breakout pins for the device which allows direct connection of pre-labeled pins to the breadboard. This is extremely helpful in terms of easy access and identification of each pins. The LaunchPad also provided options for output voltages of 3.3 volt or 5 volt, programmable LEDs, and button inputs.

3.7 Breadboard

The bread board is critical to the functionality of the device, as it allows for safe and organized construction of the circuitry. The main components on the breadboard are: a thermistor, a 10K ohm resistor, a voltage regulator, and two 2 watt 47 ohm resistors. The thermistor contains two pins, one is connected to the 3.3V pin on the MSP430F5529 and the other is connected to the 10K resistor in series to GND. The voltage regulator contains three pins, left to Vcc, middle to GND, and the right pin is to regulate voltage. It is pulling 12V from the power supply and is connected in series with the power source pin, yellow wire, of the fan. The GND, black wire, of the fan is connected in series with the voltage regulator GND to power supply. The PWM of the fan, blue wire, is connected to pin P1.2 of the MSP430.

4 SYSTEM DESIGN THEORY

This project has a few design requirements briefly covered in section 2. Namely the ADC has to be set up correctly. The functions for calculating the temperature have to be derived and implemented. The timers have to be set up, so that they can keep track of time and also PWM the fan.

4.1 Design Requirement: ADC

The ADC is the most important component of the device. The ADC is what allows the MSP to read and understand the voltage output from the voltage divider (See figure 4). The ADC in this project is set up to do continuous single channel conversions. This means that the ADC is doing conversions as fast and often as it can, and stores those conversion in it's local register (In this case 'ADC12MEM0' was the one used). This setup allows the program to not have to interface with the ADC outside of the initial setup. Anytime a function wants to know the current voltage across the thermistor, it just has to read the ADC value from that register.

4.2 Design Requirement: Functions

The functions used in the code represent the brains of the program. There a re a few important functions that are used that help the program accomplish it's goals. One such equation was talked about briefly in section 3. Another example of functions that were necessary would be the function to convert the ADC reading back to voltage. JUST because the ADC converts analog signals to a digital number doesn't mean that digital number is exactly the value it reads. The ADC has a range of 3.3 volts and 12 bits to describe what signal it is reading. This can be used to determine the bits per volt seen in equation 2.

$$Bits\,per\,volt = \frac{ADCMEM0 \times 3.3}{2^{12}} \tag{2}$$

This simple equation can turn the ADC reading back into a voltage reading. The value is stored in a float, as to not lose the decimal accuracy.

The last main function used in the program was to convert the calculated resistance value to a temperature. This function was slightly more complicated than the others, it can be seen in it's entirety in snippet **??**.

```
float Convert_RtoT(float R2_value){      // function for converting
                                         // the resistance value to
  float resist;                          // temperature
  float temperature;
  resist = R2_value;

  if ((resist <= 32554) && (resist > 19872)){
    temperature = - 0.0008 * resist + 25.311;
  }
  else if ((resist <= 19872) && (resist > 10000)){
    temperature = - 0.0015 * resist + 39.319;
  }
  else if ((resist <= 10000) && (resist > 4372)){
```

```
    temperature = -0.0035 * resist + 59.061;
  }
  else if ((resist <= 4372) && (resist > 1753)){
    temperature = -0.0094 * resist + 84.646;
  }
  else if ((resist <= 1753) && (resist > 786)){
    temperature = -0.0256 * resist + 113.46;
  }
  else if ((resist <= 786) && (resist > 442.6)){
    temperature = -0.0579 * resist + 139.74;
  }
  else if ((resist <= 442.6) && (resist > 182.6)){
    temperature = -0.1334 * resist + 171.64;
  }
  return temperature;
}
```

This function essentially creates a piece-wise linear approximation of figure 1. When the function is passed a resistance value that needs to be converted, the value is compared to the options in each of the if statements. The if statements represent the range where their function is accurate. When the correct if statement is found, the equation inside is used to calculate the temperature. This approach allows for quick and accurate temperature calculation, without wasting all of the MSP's RAM space.

4.3 Design Requirement: Timers

The timers are used for two main tasks. The first task is simple, keeping track of time. The first timer is setup such that it fires an interrupt every 100 milliseconds. This is done by driving the timer with a clock running at a specific frequency, and then finding how many cycles that clock runs through in 100 milliseconds, and setting the timer to fire an interrupt after that many cycles. It also allows the program to have a very accurate timer that has one simple job, which is important because it allows the program to do all of it's task in just once every 100 milliseconds. This way the MSP isn't doing these calculations as fats and as often as possible.

Normally, it might be considered desirable for the program to run as many calculations as possible, however, for the program being used this would be counter productive. The code tries to predict what the next temperature will be based on the last 5 recorded temperatures. If it is just getting and calculating temperatures as fast as possible, not enough time will have passed for a meaningful change to have occurred. This timer allows some time to pass before getting a new temperature value.The second timer is used to PWM the fan. This timer is setup in OUTMODE 2, reference figure 7 for details on this.

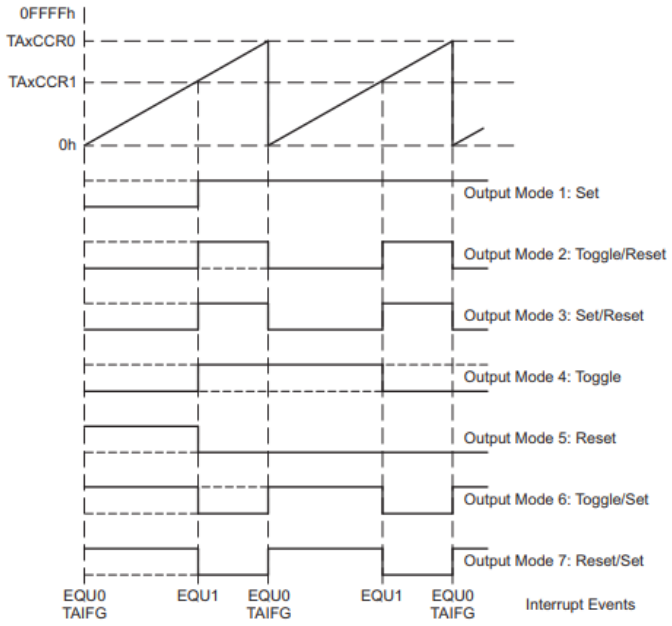

Figure 7: Displays the different OUTMODES and what they do

The CCR0 value was set to 99 and CCR1 was set to what ever duty cycle percent is desired. OUTMODE 2 was used because it would create the inverse of the desired duty cycle percent. This means that if CCR1 was set to 60% the OUTMODE would actually make it 40%. If CCR1 was set to 23% the OUTMODE would output 77%. This is an important feature, because the fan used would turn off when ever it received a high value, and would stay on when it receives a low value. By using this OUTMODE, math has to be used to get the correct number (Although it would just be a simple subtraction problem) This saves power and also CPU time.

5 Getting Started/How to use the device

5.1 Hardware Overview

Figure 2 shows the MSP430F5529 from a top down perspective. The pins used for this project are (Starting on the left) 3.3V, Gnd, and 6.0, followed by (On the right) 1.2. Pin 3.3V and Gnd are a 3.3 volt rail and Ground rail respectively. Pin 6.0 is the input for the ADC. Pin 1.2 is the output for the fan PWM. The launchpad used is excellent for testing because these pins are directly exposed (As opposed to using just the chip and having to map them out directly). In addition to the MSP430F5529, a bread board, power supply, and fan will also be required, The breadboard allows for easy setup of the components and connections. The Fan will be used to cool the object of interest, and the power supply will be used to power the fan. The fan used in laboratory testing

was a 12 volt fan, and as such, a power supply capable of delivering at least 12 volts is required.

Some smaller components are also required for testing. A thermistor is require din order to get the temperature of the object of interest. The thermistor used was the NTCLE100E3 10K ohm variant. Using this device will allow for no necessary code modifications, since all of the equations used are tuned to this exact model. A 10K resistor is also required to be in series with the thermistor to create a voltage divider.

In laboratory testing, a voltage regulator was used to generate heat. The voltage regulator was hooked up in parallel to the fan, meaning that it had 12 volts across it. The exact model of voltage regulator isn't important, because they all make heat, and that is all that matters. Firmly attaching the voltage regulator to the thermistor allows for optimal heat transfer, apply thermal compound if necessary to improve heat transfer further.

5.2 Communication

To communicate with the device, a program that can communicate over UART is required. If the host operating system is a Linux based device, the user just has to navigate to the /dev/ directory. The user then finds their device, and pipes all the commands they want to send directly to that device using their terminal of choice.For windows users, it is recommended that they utilize RealTerm.

6 Getting Started Software/Firmware

6.1 Software

There are two pieces of software required for this project. The first is Code Composer Studio, which is available for the three major operating systems and is free to download for all users. It also comes pre-packaged with all the resources needed to compile the board used in this project. The second piece of software required is some form of UART compliant program to communicate with the board while it is running the software. For this purpose, there are two recommendations: RealTerm, and PuTTY. The in house testing used RealTerm, but PuTTY would work just the same.

6.2 Firmware

In order for the device to run properly, an updated copy of the MSP430G2553 firmware must be installed to the board. To update the firmware, simply load up Code Composer Studio and compile the project to the board. Code Composer Studio will automatically detect that the boards firmware is out of date and prompt you to update it. Never unplug the device while updating firmware, as this could permanently damage the device.