Lab 2 Report

ECSE 426 – Microprocessor Systems

Group 7

Gregoire Martin – 260364773

Patrick White – 260353491

October 21st 2013

# Table of Contents

[1. Abstract 2](#_Toc369952480)

[2. Problem Statement 2](#_Toc369952481)

[3. Theory and Hypothesis 3](#_Toc369952482)

[4. Implementation 4](#_Toc369952483)

[5. Testing and Observations 4](#_Toc369952484)

[6. Conclusion 4](#_Toc369952485)

# 1. Abstract

The primary goals of this experiment are to provide a simple graphical output of the processor core’s temperature using the STM32Fx board’s LEDs, to implement a simple pulse width modulation (PWM) algorithm and demonstrate its correctness using the board’s LEDs, and to provide to the user a method of selecting between the modes of operation using a button. This experiment involved the use of sensors, timers, basic input and output, as well as some basic signal processing for the sensors and for a button.   
\*\*\*\* NEED TO ADD DATA AND BRIEF CONCLUSIONS \*\*\*\*

# 2. Problem Statement

The end goal of the experiment is to have a simple LED display indicate whether the board is heating up, cooling down, or maintaining its temperature, as well as demonstrating PWM using the same LEDs and allowing the user to select the mode of operation. The problem can effectively be broken down into five parts:

* Acquiring data from the temperature sensor
  + The data from the temperature sensor must be acquired as a voltage value and converted into a temperature using a formula provided in the datasheet.
  + The data must be sampled at a high enough rate to be useful.
  + The raw voltage readings are provided in analog format, thus the data must be converted to digital format in order for the processor to be able to use it.
* Filtering noise out of the signal
  + The signal is expected to be very noisy. A filter must be used to improve the quality of the signal and get rid of the unwanted noise.
* Updating the LEDs according to the temperature
  + The four main LEDs on the board must light up in a clockwise fashion with only one LED on at a time when the core temperature is increasing, and they must turn on in a counter clockwise fashion with only one LED on at a time. The LEDs must change whenever the temperature changes by 2 degrees Celsius.
* Developing the PWM algorithm
  + Briefly describe what PWM is here
* Providing the user a way of selecting between the two modes of operation
  + The user should be able to select between PWM and temperature tracking by simply pressing a button on the board.
  + The signal from the button will be noisy because of contact bounce, and a way must be devised to obtain a clean reading from the button

These five aspects will allow the board to provide a simple display to the user that describes the temperature trend of the processor’s core, as well as demonstrating PWM on the board’s LEDs and allowing the user to easily select the desired mode of operation.

# 3. Theory and Hypothesis

The values obtained from the temperature sensor will originally be voltages in an analog format. Because the readings are in an analog format, Analog-Digital conversion must be used to convert the data into a format that the processor will be able to use.

From the STM32Fx datasheet (add reference), the formula to convert the voltage reading from the temperature sensor into a temperature in degrees Celsius is

Where V25 is the voltage measured at 25 degrees Celsius, or 0.76 V.

It is expected that the voltage readings from the sensor will have unwanted fluctuations which will make appear that the temperature is varying far more than it actually is. This noise can be caused by many sources; electromagnetic interference, thermal noise, quantization noise from the ADC among other things. The signal must be processed in such a way to filter as much of this noise as possible. A very simple filter we can use is the moving average filter. The moving average filter keeps a buffer of the D most recent samples, and takes the average of those values and outputs that average. When a new measurement is taken, the oldest measurement in the buffer is discarded, the new measurement added to the buffer, and the average recomputed.

In general, a smaller window would be better suited to the task at hand in order to preserve the resources of the system, while reducing the noise in the signal to an acceptable level. For example, a filter depth of 5 may be desirable.

Provide PWM theory and hypothesis for window (add figure probably to demonstrate pwm)

In order to obtain a clean reading from the push button, some basic signal processing will have to be applied to the reading because of contact bounce. (add reference and probably a figure to demonstrate the bouncy signal) Contact bounce occurs because many buttons are made of springy metals. When the spring comes into contact with the electrical contacts, it will result in a “bouncy” signal where the bit value may rapidly pulse between 0 and 1 for a short period of time, preventing the programmer from reliably knowing if the button was pressed. It will be necessary to “debounce” the button using some very basic software functions. To debounce means to correctly detect the button press. A simple way of doing this is to check for both a press and a release. If the bit indicating a press is set, wait a short period of time, and then check if it’s been unset. This gives the signal time to be fully asserted, and fully de-asserted before the bit readings are taken.

# 4. Implementation

Give brief overview of whole system

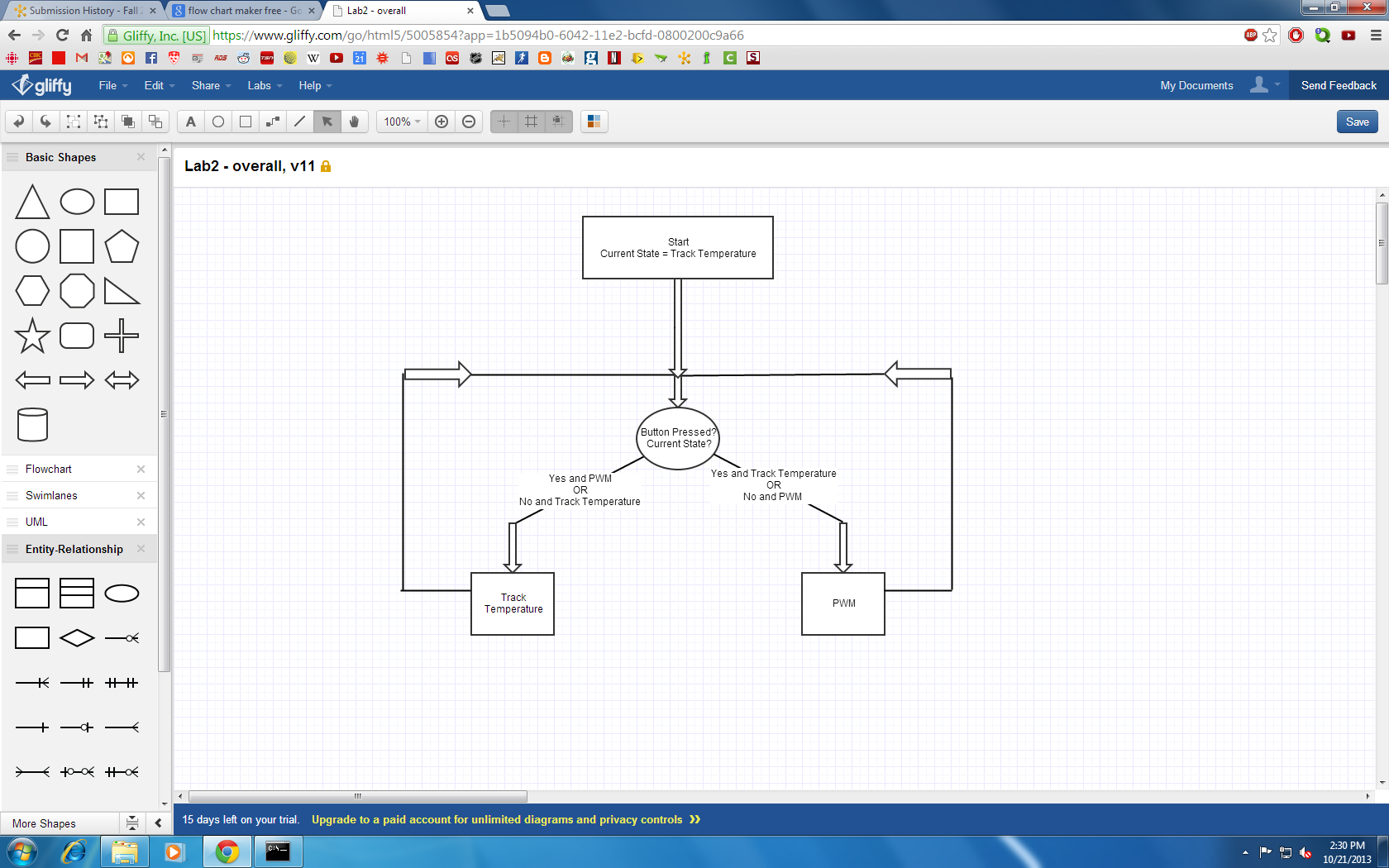


Figure 1: Flowchart of overall system

To obtain the readings from the temperature sensor, the ADC had to be initialized (talk about ADC initialization) In accordance with the specification, the sensor was to be sampled at 20 Hz. To get this sampling frequency, the board’s timer was used. (talk about systicktimer setup here) The timer was set to issue an interrupt 20 times per second.

When the interrupt is received, it signals that a new reading from the temperature sensor should be taken. The first thing to happen is that the timer is reset to begin counting until the next interrupt. At this point, the value returned by the temperature sensor is read in from the ADC. The value returned by the ADC is a voltage value, and thus it must be converted appropriately to get the actual temperature detected by the sensor. The voltage is divided by 4095 (or 212 – 1) in accordance with the bit resolution of 12 to obtain the fraction of the reference voltage detected by the sensor. This fraction is then multiplied by 3 V, to obtain the actual voltage detected by the sensor. The actual voltage is then applied to the formula (add reference) to obtain the temperature reading. The temperature value needs to be cleaned up using some basic signal processing in order to increase the reliability of all the temperature readings.

To filter the noise from the readings of the temperature sensor, a moving average filter was employed. The filter was implemented as a C struct. A ring buffer technique was employed to avoid having to shift all the buffer values at the addition of each new temperature reading. The struct hold an array of length D to represent the buffer, the sum of all the values in the buffer, the average of all the values in the buffer, and the index to keep track of the next position to insert a value. Every time a new value was read from the sensor, it would be written to the buffer, the sum and average would be recomputed, and the index incremented or reset in order to wrap around when it reached the end of the buffer.

To optimize the moving average filter, a Matlab model was used to compare different buffer lengths and the resulting quality of data. The moving average filter was applied to several data sets from the temperature sensor, and the buffer length was varied. After running several data sets through the Matlab model, it was determined that a buffer of length 10 would be used. For the specifics of the testing and optimization, see [Section 5: Testing and Observations](#_5._Testing_and).

The filtered temperature readings were then applied to the problem of implementing the rotating LEDs to show the temperature trend. The variables tracked for this problem were the current LED illuminated, the new filtered temperature reading, and the base temperature. The LEDs are numbered 0 to 3, with clockwise representing increasing values. In other words, LED 1 is one position clockwise from LED 0, and accounting for the wraparound, LED 0 is one position clockwise from LED 3. The inverse is true for the counterclockwise sequence. The base temperature is the value that last caused a change of LED. After the new temperature value is filtered, it is passed to a function that updates the LEDs.

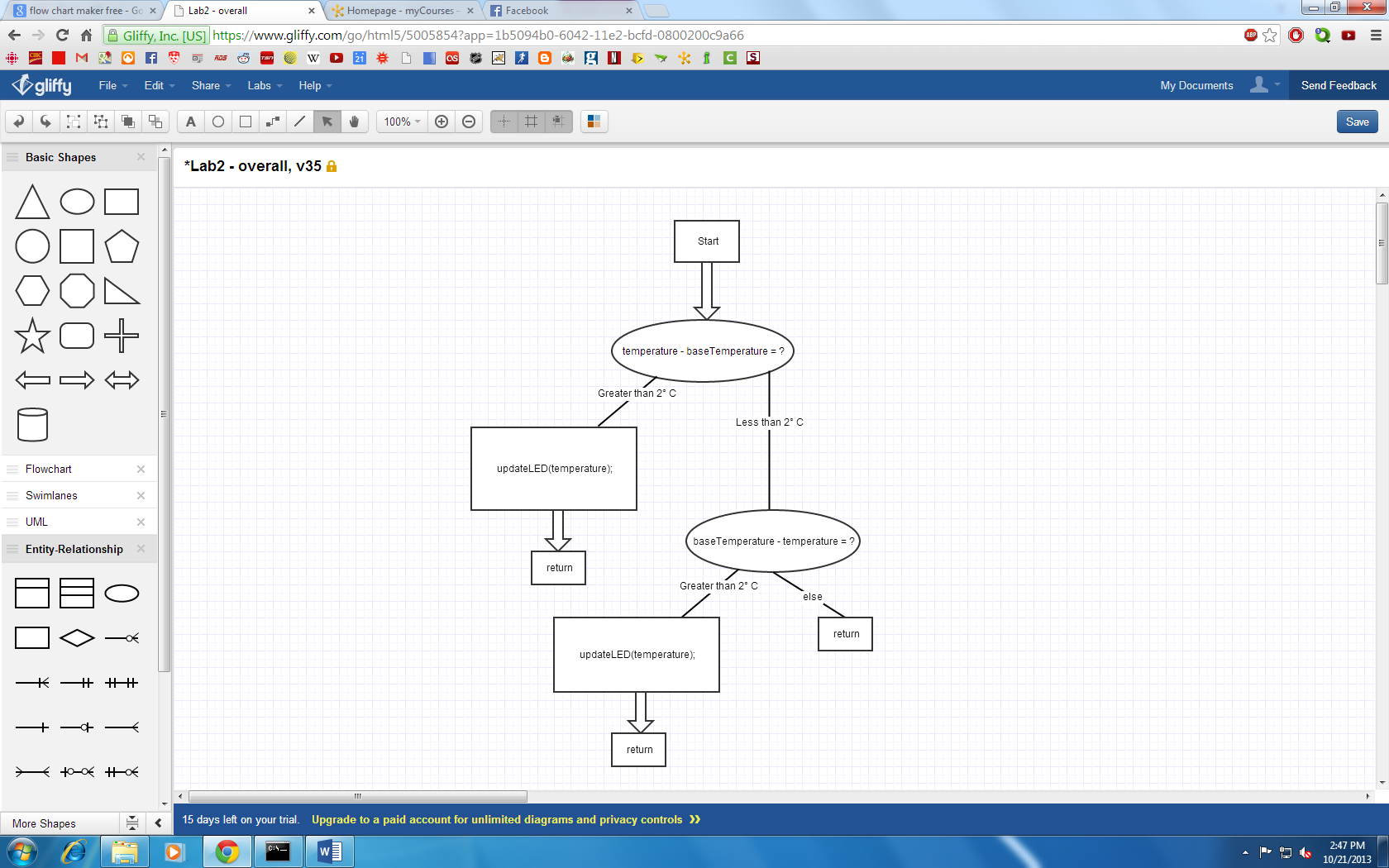


Figure 2: Temperature Tracking Flowchart

The function initially checks if the new temperature is at least 2 °C greater than the base temperature (the temperature that last caused the LEDs to change). If the new temperature satisfies this condition, it means that the LEDs must change with the current LED being turned off and the LED immediately clockwise of the current LED must be turned on. At this point, the function checks which LED is currently illuminated, writes a bit value of ‘0’ to GPIO pin of the current LED, and then writes a bit value of ‘1’ to the GPIO pin of the next LED clockwise from the current one. Finally, the function updates the current LED variable to the LED that was just turned on, and updates the base temperature to the temperature value that caused the change in LED.

In the case where the new temperature is not at least 2 °C greater than the base temperature, the function next checks if the new temperature is at least 2 °C less than the base temperature. If this condition is satisfied, this means that the LEDs must change with the current LED being turned off and the LED immediately counterclockwise of the current LED must be turned on. The function checks which LED is currently illuminated, writes a bit value of ‘0’ to the GPIO pin of the current LED, and then writes a bit value of ‘1’ to the GPIO pin of the next LED counterclockwise from the current one. Finally, the function updates the current LED variable to the LED was just turned on, and updates the base temperature to the temperature value that caused the change in LED.

In the case where neither of the two conditions are satisfied, the function simply returns.

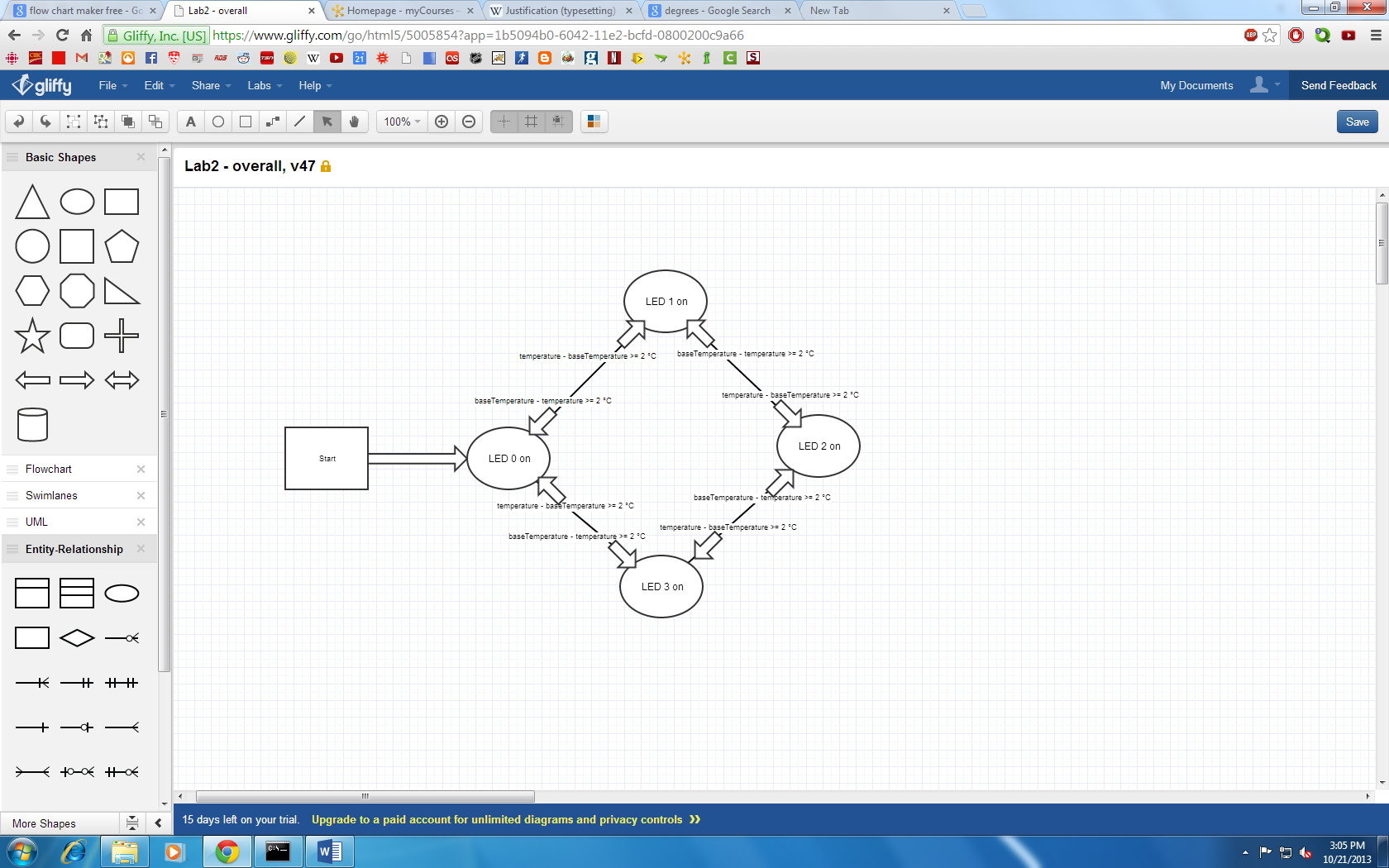


Figure 3: LED updating algorithm

In order to select between the two modes of operation, the system needed to be able to detect a button press. This required a simple debounce function to be implemented. Every time the main loop restarts, the function is called. The function first checks to see if the button’s GPIO pin is set. If the pin is not set, it means that the button is not pressed, and the function returns 0. If the bit is set, the function then waits 100 ms (using the SystickTimer) and then checks the bit again. If the bit is not set, this means that the button has been fully pressed and then released, and the function returns 1. If the bit is still set, this means that the button has not been released yet, and the function waits again for 100 ms. The same check is then done again, and if the bit not set, it means that the button has been released, and the function returns 1. If the bit is still set, it means that the button has not been released, and it returns 0.

With a functioning button, it was possible to implement a mode selection. The main function uses an integer to keep track of which mode of operation the system is in. The function checks for a button press, and if it detects one, it will update the SystickTimer frequency, and then it will execute the correct routine (PWM if the system is in temperature tracking mode, and vice versa).

# 5. Testing and Observations

Talk about testing that temperature sensor was sending real values

To test and optimize the moving average filter, a Matlab model was used to compare different buffer lengths. Four data sets were used, all data was temperature values output by the temperature sensor. For each data set, the raw data was plotted. A moving average filter was then applied to this data, with buffer lengths of 5, 10, 15 and 20. The filtered data was then plotted and compared with the raw data. As expected, the moving average filter showed a significant reduction in the random fluctuations and noise. The various buffer lengths were then compared against each other, and it was determined that a buffer length of 10 offered the best trade-off between noise reduction and memory usage.

Add Matlab figures

The rotating LEDs were tested by inspection. A “printf” function was added to the start of the function that updates the LEDs. The printout displayed the filtered temperature value each time the function was run, so that the temperature value was observable. Then, the board was turned on, and the temperature tracking enabled. The processor was warmed up by placing a finger on it, and the LEDs observed. After the processor had warmed up and the LEDs were observed rotating clockwise as expected, the processor was allowed to cool down and the LEDs were observed rotating counterclockwise, as expected.

Talk about PWM testing

The push button debounce was tested simply by pressing the button, and observing if the LEDs changed from the rotating LEDs to the PWM. Various “types” of button presses were tested, such as pressing it quickly, holding the button down for a long period of time, and a normal press. It was found that with the double-check debounce routine, all of these pressed were detected correctly.

# 6. Conclusion

# Appendix

## Appendix A - References