**Rochester Institute of Technology**

# EE420 Technical Memo

**To:** Prof. Barrios

**TA:** Chris Schwab

**From:** Connor Goldberg

**Date:** 8 December 2014

**Title:** Embedded Systems Design, Final Lab

---

## Abstract

Lab 12 was the final step of a multi-lab project using a microcontroller. An MSP430 was used as the microcontroller to obtain an analog set of data, process the data digitally, and then display the values back to the user using both LEDs and serial communication. The project was broken up into smaller parts, completing the project one step at a time. Over time, the individual parts were combined until the necessary functionality was implemented. In some instances, changes had to be made to the previous parts, however in most cases they remained the same. After the project was successfully designed and implemented using the microcontroller, the project was demoed to the professor to verify that the project passed the requirements of the course.

## Design Specifications

Several steps were taken in the design of the final project. The first step was to complete a flowchart that outlined the intended functionality of the program in a neat, yet detailed manner. The function of this project was fairly simple overall, however each individual part to the project was highly specific and very detailed.

After the program is written to the memory and starts to execute, it checks a capacitive touch board attached to the microcontroller. This board contains 5 separate sensors, one for each standard direction, and one in the center. The program waits and constantly checks the capacitive touch sensor until it detects that one of the buttons was pressed. If any of the four directional buttons are pressed, a corresponding LED that is dedicated to that sensor turns on. When a new directional sensor is touched, the previous LED turns off, and the new LED turns on. This continues until the center sensor is pressed.

Once the center button is pressed, the program starts checking for the up and down directional sensors. If any other sensors are pressed, nothing happens. If the user touches the up sensor, it sets a variable that will make the data acquisition occur at 100 Hz. If the user touches the bottom sensor, the variable is set to make the data acquire at 200 Hz. Next, the program starts checking for the left and right directional sensors. The left sensor will set a variable to make the data values display for 1 second each, the right sensor will set the variable to cause the data values to display for 2 seconds each.

Once these variables are set, the analog to digital converter (ADC) is set up and 20 samples are acquired through GPIO pin 0, on port 1. These 20 values are placed into an array so modifications can easily be made. The data is modified and transformed into thermometer scale values. This scale is a most significant bit representation of the data, where the most significant binary bit of the data is found, and all

bits to the right of the most significant bit are set to 1. For example, 0100 would become 0111. In addition, because the ADC acquires each data point as a 10-bit value, the two least significant bits are cut off and the remaining 8 are shifted right. This makes displaying the values easier; there are 8 LEDs oriented in a circular pattern on the capacitive touch board.

After this conversion of the acquired values to a temperature scale is completed, the program starts to display the values and send them out over UART (serial communication). The value is first transmitted over the Tx line using the UART protocol, then the LEDs will light up that correspond to the temperature scale value. The values will be displayed for either one or two seconds, depending on the initial input from the user, then proceed to the next value of the array. This process will continue indefinitely until the center sensor is pressed. If at any point during the display loop the user presses the center button, the program will start from the beginning.

## Results & Discussion

The program was created with the end goal of being complied onto the MPS430, specifically the MSP430G2553. This board was chosen as the microcontroller for several reasons. It is fairly cheap for a microprocessor, and for the applications that it was to be used for it contained hardware that would suffice; the ADC is 10-bits which provided a good resolution for an input signal, the timers are accurate, the clock speed is fairly fast, and the memory space is ample. In addition, the MPS430G2553 is compatible with the "Capacitive Touch BoosterPack". This contains five separate capacitive touch sensors along with nine LEDs. Eight of the LEDs are located in a circular pattern around the edge of the board, and a ninth LED is located in the center under the center capacitive touch sensor.

The project could have been made using assembly, C, or a combination of both. I decided to write this project completely in C because I thought it would be straightforward, and there were no strong reasons to make the project in assembly or a combination of both. The time it would take to make the program in assembly would outweigh the benefits.

Assembly code, when compiled onto the microcontroller, runs much quicker than C code. This is because assembly code is very close to a one-to-one relationship with machine code. Most of assembly code is a direct translation into a small number of instructions, pertaining to the exact instructions that are represented by the assembly. When writing in C, however, this is much different. When C code is compiled, it goes through more internal steps, much of which is automatic optimization. As a result of these steps, there can be around 1.5X the amount of instructions generated from C code that has the same functionality as assembly code. An additional benefit to assembly code is the level of control. For example, it is much easier to access specific registers. In C however, it is not as simple. There is a header file that includes pre-defined functions that have access to some specific assembly instructions.

Although there are benefits to assembly, it is much harder to write. Assembly code forces the programmer to actively think about each individual instruction, keep track of every variable manually, and be aware of how the memory space is being used. In C code, most of this hard work is taken care of by the compiler and linker. It is also much easier to use control-flow effectively. For example, instead of using compare and jump instructions, if-else statements, loops, and switches can be used. This makes the code much easier to read, and debug. For the specific application of this project, there was no need to conserve clock cycles, the speed of the program, or how many total instructions were placed in memory because the application did not require it. Writing the program entirely in C saved time along with making it easier to debug.

An additional piece of hardware that was integral to this project was the timer block of the MPS430. The timer block is called TimerA, and has two separate timers. This allows for the concurrent use of two timers. Each timer contains a 16-bit register, and a wide range of support for inputs, outputs, and interrupts. The timer was used for two different functions in this project. The first use is in the detection of the capacitive touch sensor. The timer is used to help determine which sensor is being pressed. The second timer is used in the display loop, when the temperature scale values are being displayed on the LEDs. The timer used in the capacitive touch sensor function is never on for very long periods of time, and sub-main clock (SMCLK) is used as an input. This clock is taken from the main clock of the microprocessor and operates at a frequency of roughly 1.1 MHz. This is the fastest clock built in to the microprocessor (external clocks

can be added); therefore it is the most precise. For the display loop, the timer has to account for either one or two seconds at a time. At first it seemed obvious to use ACLK, which has a frequency of 32 kHz. This could be used without a divider to obtain both one and two-second times, however later it was decided that using the SMCLK would be superior.

The timers use interrupts to function. Interrupts are a very important concept to understand and they are important in the design of many embedded systems. Interrupts are generated by various pieces of hardware for many different reasons. In the instance of the timers, the interrupts are set up to generate after the timer has counted to its value. When an interrupt is generated, the program immediately jumps to a separate section of code specific to the interrupt that was generated. These sections of code are called interrupt service routines (ISR). A multitude of tasks can be completed inside of each ISR. The way that they were used in this project was by setting a software flag, clearing the interrupt flag, then returning to where the program was previous to the generation of the interrupt. An interrupt flag is a value that is stored inside of a register specific to that interrupt, while a software flag is a variable that is placed in memory. When waiting for an interrupt, a while loop was used that detected a software flag. Once it was detected, the while loop was passed, the software flag was reset, and the program continued to execute. Once set up, the interrupts were implemented effectively.

One problem that occurred during the project involved using the LEDs in certain sections of the program. Even though the LEDs would be set to display, nothing would occur. The reasoning for this was the setup of the I/O ports. The same I/O ports are used for three different functions in this program. They control the LEDs, take in inputs from the capacitive touch sensors, and take in inputs for the ADC. For each of these functions, the port needs to be setup much differently. For example, when controlling the LEDs, the port has to be setup as an output, but for both other uses it has to be setup as an input. It is extremely important to check how the control registers are setup for each port before they are used, every time. The problem occurred because at first the port was not reset after the ADC was used. This caused interference with the LEDs. Once the ports were set back to their correct control settings, this problem was fixed.

The display loop was the most difficult part of this project and it caused the most problems. The first issue arose when trying to only use one timer. The timer was being used to measure the period of time that each value would be displayed for, however it was also being used in the capacitive touch function. This caused issues at first, but was resolved by using the second timer and splitting the functionality. As a result, one timer was only ever being used in the capacitive touch function, and the other timer was solely being used to measure time in the display loop.

The second problem that occurred in the display loop was with the duty cycle of the LEDs. The LEDs on the board are multiplexed, therefore only four of the LEDs can be on at any given time. Therefore, to display more than four LEDs at the same time, the program has to alternate between the two sets of four fast enough such that the human eye cannot detect the alternating back and forth. This is easily achieved with a fast enough speed, but as a result the LEDs appear dimmer because they are not on for as long. The problem arose when trying to check the capacitive touch sensor during the loop that controlled the duty cycle of the LEDs. The capacitive touch sensor has to be constantly checking for a press of the center button as per the project requirements. This was a problem because the check for the capacitive touch sensor took a relatively long time compared to how fast the LEDs needed to be switched back and forth. As a result, the majority of the one/two seconds in which the LEDs were supposed to be displaying the temperature value, it was checking the capacitive touch sensor instead. This resulted in the LEDs being extremely dim, almost to the extent that the human eye could not detect that they were on. To solve this problem, instead of checking the capacitive touch sensor as often, it was only checked every 100ms. This allowed the LEDs to be on for a much longer amount of time relative to the amount of time the program was checking the capacitive touch sensors. After this fix the LEDs were all adequately visible. Other problems that arose were minor and were fixed easily.

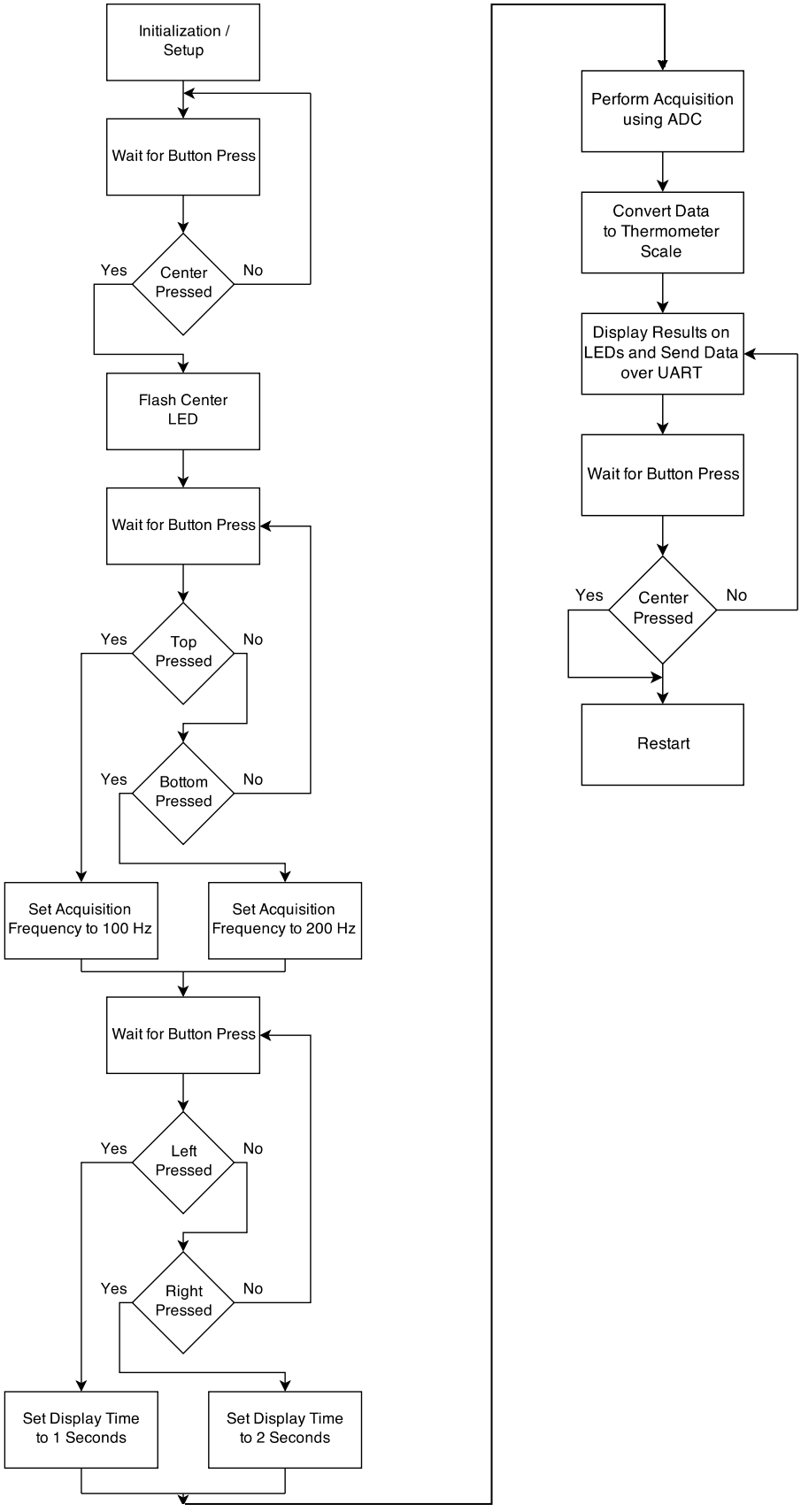
One of the easiest parts of the project was using UART. This section only required sending the temperature scale value over the serial line. This involved simply converting one of the nine possibilities (0-8) to an ASCII code, then placing this code into a buffer, which would output the value out over the serial line. The serial line could be read in on the computer via a terminal to verify that the values were being sent correctly. The only difficult part to the serial functionality was setting baud rate manually. Due to

variances in the silicon of every chip, the BAUD rate (rate at which each bit is sent) has to be set manually. This was solved by monitoring the serial line on an oscilloscope, then performing some algebra to find how much the BAUD rate needed to be delayed.

## Conclusion

This final project was an excellent way to learn how to use assembly and C to create an embedded system. This embedded system, though fairly simple, the concepts that were used can be used in an endless number of applications. This project clearly showed how to effectively use timers, ADCs, GPIO pins, serial communication, interrupts, and more. This information is used many embedded systems, and can be easily expanded upon. After the completion of this project, embedded system design is now clear and understood.

## Flowchart


Initialization / Setup
Wait for Button Press
Yes
Center Pressed
No
Flash Center LED
Wait for Button Press
Yes
Top Pressed
No
Yes
Bottom Pressed
No
Set Acquisition Frequency to 100 Hz
Set Acquisition Frequency to 200 Hz
Wait for Button Press
Yes
Left Pressed
No
Yes
Right Pressed
No
Set Display Time to 1 Seconds
Set Display Time to 2 Seconds
Perform Acquisition using ADC
Convert Data to Thermometer Scale
Display Results on LEDs and Send Data over UART
Wait for Button Press
Yes
Center Pressed
No
Restart