Rochester Institute of Technology

Real-time & Embedded Systems

SWEN 563 / CMPE 663 / EEEE 663

Project 1

February 12, 2019

Nicholas Breeman | nxb9283@rit.edu

Arthur Fok | axf7314@rit.edu

**Project Overview**

An STM32 Discovery board was used to display lists of counts for a thousand rising edge pulse arrival occurrences. The time between the pulses are around one millisecond for a 1kHz square wave.. To capture our data we read data from the capture/compare register at every rising edge event. The data needed to be confined to the set bounds. These bounds had to be configurable via user input to the USART but the upper limit must always be 100 microseconds greater than the lower limit. In addition to displaying our data, we had to perform a POST (Power On Self Test) before the measurements occurred. Inside the POST, the board must receive a rising edge within 100 milliseconds of starting. If it fails then the user can decide via the USART whether to terminate the program or retry the test.

**Area of Focus**

For this project we were focused on getting it finished as soon as possible so a division of tasks was never really established. Instead we figured out what our strong points were and worked with what we were comfortable with. Nick was more comfortable with writing in C so he took charge in writing the majority of the code. Arthur has experience with register manipulation and worked on configuring the timer. Every task was worked on together.

**Analysis / Design**

We used the UART demo as an outline for our user interface. From there we decided that the best place to start designing would be to configure the timer. We decided to use Timer 2 because it is a general purpose 32 bit timer which would be more than enough for our needs and we would never overflow the timer if reset before each round of measurements. We configured timer channel 1 as input capture so we can find the time span between the rising edges of our signal. The clock is initialized to the timer before any configuration is done. We decided to use the prescaler of 79 because we are using a system clock of 80 MHz and needed to use a one microsecond count. The equation divides the input clock by the prescaler plus one, so our timer frequency was 1MHz to achieve the 1us period.

We then moved on to the POST. First both the timer and the compare/capture register are cleared. In order to pass the POST, timer 2 must receive a capture within 100 milliseconds. We used a loop to continuously check for capture data while our count was less than 100 milliseconds. Inside the loop we checked if our capture/compare register (CCR1) received any value. If not then it would fail our POST.

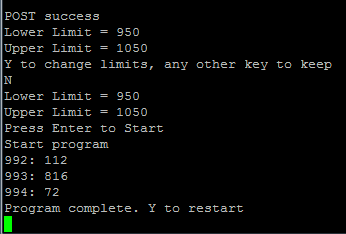
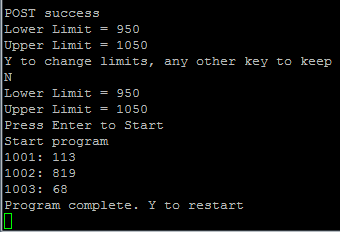
We now have both Timer 2 and POST configured. The next piece we needed was setting our lower and upper bounds. Our default bounds were set to 950 to 1050 microseconds. This needed to be configurable with user input. Using a UART read function, we read the user-inputted value. We received up to 32 characters from the USART and stored them in a character array. We then used the function *atoi()* which converts a string into an integer so we can easily read and set our bounds. Our upper bound must always be greater than the lower limit by 100 microseconds and our lower limit had to be between 50 and 9950 microseconds. If not within that range, the user would be continuously prompted for a new value until a proper one is given.

After configuration, the program waits for Enter to be pressed. Then, the timer and CCR1 and cleared again, and measurements start. To ensure every measurement is taken between a full period of rising edges, the very first value of CCR1 is used as a starting point (in a variable *timeStampPre)*, but not used in calculation. The program waits for CCR1 to be different from *timeStampPre*, as a changing in CCR1 is indicative of a new rising edge occuring. The next value of CCR1 is stored in *timeStampFirst*, then it waits again for CCR1 to be different from *timeStampFirst*. The next CCR1 value is stored in *timeStampSecond*, then the measurement is recorded in an array 100 integers wide (to store the full range of lower limit to upper limit). “*timeStampFirst - timeStampSecond”* represents the time between rising edges, and the entry that should be incremented by 1. Because the array starts at element 0, the measurement is offset by the lower limit, making the full index *“timeStampSecond - timeStampFirst - lowerLim”*. After 999 more recordings, the program prints each non-zero array element along with the period it represents, then prompts the user to either restart the program or terminate it.

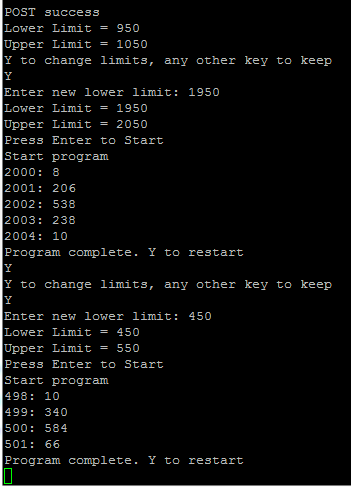
**Test Plan**

To test our program we first needed to pass our POST. We first needed to determine if our CCR1 register was receiving any values. We hooked the signal generator to pin PA0 of our board and ground to ground. Using the debugger we ran our code step by step. Doing so allowed us to see if CNT was incrementing and if CCR1 was capturing data. When we verified that we had some data flowing into our registers we now tested our POST with user interface. We connected our board to PuTTY and our signal generator. When we loaded up our program we were greeted with a successful POST and were able to continue on with the program. We disconnected the signal generator and restarted our code. This time we were not able to pass the POST and were allowed to retry the POST or terminate the program. When we connected the signal generator again, we were then able to pass the POST.

After successfully passing the POST, we went on testing our data using the signal generator and changing our bounds. For our first test we left our bounds as default as 950-1050 and tested it at 1kHz using the signal generator and the oscilloscope. The second case we tested was our bounds as 450 - 550 and at 2 kHz and for our third case we tested the bounds at 1950 - 2050 and a frequency of 500 Hz. We expect our results to be within our bounds and our counts to add up to a thousand.

**Project Results**

The image on the left is with our default bounds and a 1kHz test signal generated from the oscilloscope while the image on the right is with the same bounds but a 1kHz signal generated from the signal generator instead. Our expected results match what we see on our console. The left column which is our time is within our bounds of 950 and 1050 while the columns on the right which is the count values. The count values add up to a thousand.



This image shows the test cases where we changed our bounds and frequency. We first used the bounds 1950 - 2050 and our expected results are met just like our first case. We then continued on with the program without restarting the board and changed our limits to 450 - 550. We displayed our results and they also match our expected results with the time within our bounds and the count values adding up to a thousand.

**Lessons Learned:**

This project was a great learning experience as it was our first time in a new environment. We both became familiar with working with the new software as well and the importance of enabling the clock for the registers we wanted to use. When we were first configuring the timer, we wouldn’t see the registers change like how we configured them to do so. We would get stable zeros or no change whatsoever. We learned that we never enabled our register and through one simple enable, we got out register to update with data. We also became familiar with the STM32 data sheet as we kept referring to it to make sure we were using the correct registers and bits.

Another problem that we ran into was the changing of our bounds. It originally works when we first change the bounds to whatever we want. If it was out of bounds we would get that error but when we ran the program again, we would get an error even if we were in bounds. In the end, we forgot to clear the array that held our previous bounds so we just wrote on top of it and added to it. This would always make us go out of bounds even if we wrote the correct limit.