Embedded Systems
Project 1

Date of Submission: February 12, 2019

Joel Yuhas : jxy8307@rit.edu
Athaxes Alexandre : axa2012@rit.edu

Overview

The STM32 Discovery board is a microcontroller capable of running "bare-metal(no operating system)" programs. This project interfaced with the STM32's GPIO input pins as well as timers in order to perform input captures from an exterior wave generator. A program was written that initialized hardware components which enabled the STM32 to receive inputs from a pin, count the number of rising edges within a millisecond, and display them using UART. A POST test was created that ran before the main program to test that signals were being received properly. The user was also able to set the bounds from the UART terminal and re-run the test once it had completed. Overall, all of the desired objectives were met and the project was a success.

Areas of Focus

Project Assignments	Team Member
GPIO Initialization	Joel + Athaxes
Timer Initialization	Joel + Athaxes
POST Test Implementation	Joel + Athaxes
Main Test Implementation	Joel + Athaxes
Result Gathering	Joel + Athaxes
Report	Joel + Athaxes

Analysis/Design

The main objective of the program was to measure the "inter-arrival" time between pulses coming from an outside signal generator. The STM's internal clock and the frequency of the signal generator would have slight variations, meaning that a 1 KHz pulse wave would not always send the same amount of pulses every time it was ran. The program would measure how many rising edges it detected in 1.0 millisecond and place the result into "buckets" based on how many edges were counted. The program would repeat this process 1001 times and record its results.

First, the hardware was set up to allow for the reception of signals through a pin. The system clock, LED and UART initalizations had all been provided in a previous project and were used again. A GPIO and Timer initialization needed to be created. After reviewing the data sheet, reference manual, and other class materials, pin PA0 was chosen to receive the signals. The code below demonstrates how the timer was initialized for PA0, and then set to receive inputs.

```
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN ;
GPIOA->MODER &= ~3 ;
GPIOA->MODER |= 2 ;
```

The next set of code was used to tie the pins “capture mode” to the timer and then initialize the rest of the timer.

```
GPIOA->AFR[0] |= 0x1;
RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;
TIM2->PSC = 80;
TIM2->EGR |= TIM_EGR_UG;
TIM2->CCER &= ~(0xFFFFFFFF);
TIM2->CCMR1 |= 0x1;
TIM2->CCER |= 0x1;
```

Then a “POST” routine was created. This “Power On Self Test” would confirm that the GPIO pin could read a pulse at least once in a 100 milliseconds span. This was to verify that the system was working and that the signals were strong enough to be recorded before proceeding to the main test.

After a successful POST test, the main test was started. First the upper and lower limits were displayed to a UART terminal. The lower limit was defaulted to 950 microseconds while the upper limit would always be 100 microseconds above the lower limit. The limits was used to chose the “window” of results. Only results recorded within the window would be displayed.

Next, the user had the option to change the lower limit. If they chose to do so, they could pick a value anywhere between 50 to 9950 microseconds. Attempting to set the bounds outside of this range would prompt the user to enter new bounds again. After the bounds were set, the test would move to the results gathering phase.

In the main test phase, the program counts how many edges it detected within 1 milliseconds as explained above. After 1000 iterations, the test would display all non-zero values using the UART terminal.Two columns were presented, the first being the number of edges counted, and the second being the count of how many times that number appeared in the 1000 samples.

Finally, once the results were received and displayed, the user was given the option to re-run the test. If not the test was completed.

The code below shows a simplified version of the main loop of the code that was executed.

```
int main(void){
      System_Clock_Init();
      UART2_Init();
      GPIO_Init();
      Timer_Init();

      Start_Timer();
      POST_Test();

      While(test_continue = true){
            MAIN_Test();
            FINAL_Print();

            USART_WRITE("Re run?");
            rxByte = USART_Read();
            if(userInput == yes){
                  Test_continue = true;
            } else {

                  Test_contiue = flase;
            }
      }
      Return 1;
}
```

Test Plan

The only major test that had to be incorporated into the project was the POST test. As described in the previous section, the POST test ensured that at least one signal was received in 100 milliseconds, otherwise the user would be prompted to either restart or terminate the test.

Results

The main results from this project came from the UART terminal. The first test that was ran was used to check that the POST test could fail correctly when it did not receive any signals. Figure 1. Shows the results from the UART terminal of this test case.

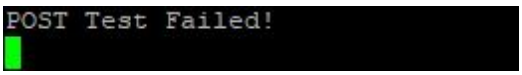

Figure 1. POST Test Fail

Then the POST test had to be sure it could pass. Figure 2. Shows the results of the passing test.

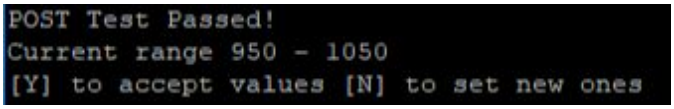

Figure 2. POST Test Pass

Afterwards, two main test were executed. One where the input was set to around 1 KHz and another where it was set to around 2 KHz. The expected results of 1 KHz was projected to be tightly clustered around 999, with only about 3-4 different results in total. However, it should be noted that the actual values received are highly reliant on the input signal, and small differences could impact the results. Figure 3 shows the results that was gathered from the 1 KHz test.

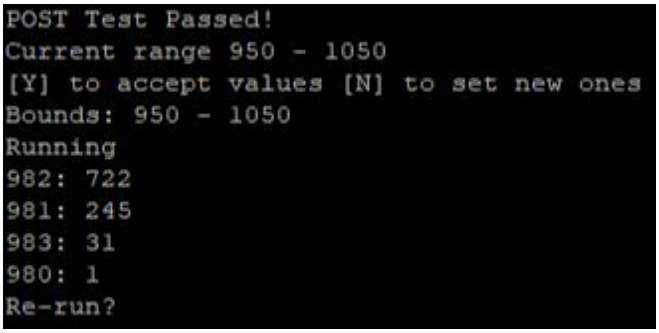

Figure 3. 1KHz Test

The gathered results fell exactly in with what was predicted, where only between 3-4 different results were recorded. However in this test specifically, it should be noted that only 999 total iterations were detected, as seen from adding all the numbers in the 2nd column. After several test this seemed to happen from time to time but more than often produced all 1000 results. Figure 4 shows the test ran again but this time at 2 KHz.

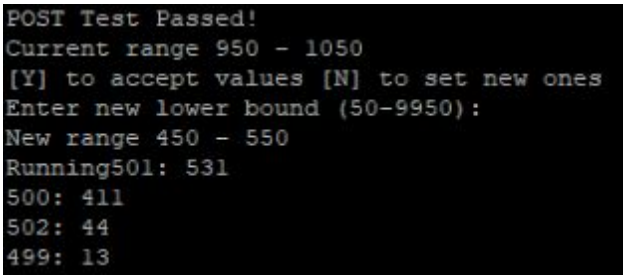

Figure 4. 2KHz Test

Notice how the 2 KHz test produced results much lower than the 1 KHz test, around the 450-550 range vs the 950-1050 range. This is due to the fact that the frequency is inversely proportional to the time between counts. These results are also tightly paired between 499-502.

Finally, the test had to be confirmed that it could be ran a second time after a test. This is demonstrated in Figure 5.

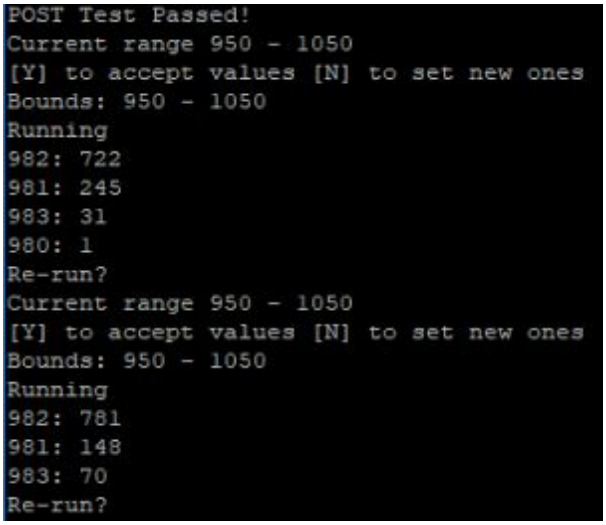

Figure 5. Test Re-Run

Lessons Learned

The most amount of trouble actually came from fully understanding what it was the project wanted. Initially there was confusion around when the project really meant by “buckets” and it wasn't known that the code was suppose to be tested against a waveform generator. The initial idea thought it would be ran against another clock. The only other area that caused major trouble was being able to properly initialize the timer and GPIO pin. During the first stages of testing, it wasn't apparent that the GPIO pin was not initialized correctly, which added significant time to finishing the project.

Additionally there were a few small issues here and there, one being having the wrong implementation of the UART write at one point which caused the write procedure to print the wrong results. The issue was fixed by simply setting the buffer size correctly. However, once those two main issues were addressed, the rest of the project went very smoothly.