Real-Time and Embedded Systems

Project 01: Timing

Author: Liam Sullivan
Email: lds7381@rit.edu
Submission Date: 09/14/2022

Analysis and Design

This exercise required the STM32 microcontroller to take a square wave input at varying frequencies and measure the period of 1000 pulses. These pulses' periods would be measured in microseconds and added to a "bucket." A bucket contains a period and the amount of times that period has been measured from the square wave. Once 1000 square wave pulses were measured, all of the buckets that have a period that was inputted from the square wave at least once were printed to the terminal.

To do this, the main peripheral on the STM32 used was the 32-bit General Purpose Timer, specifically Timer 2 (TIM2). This timer was set up to "tick" or increase once every microsecond. This timer base of increasing once per microsecond was implemented by changing the prescaler of the timer. Since the internal clock of the microcontroller runs at 80MHz the required prescaler to get a 1 microsecond increase every tick would be 80 - 1, or 79. This causes the Auto-Reload Register (ARR) to increase once every microsecond. Once the base timer was set up, the first channel on TIM2 was set up to perform an Input Capture. This meant that the timer would receive a rising edge input on its pin that tells the timer to capture the current timer's count in microseconds. This pin, PA0, was set up as an Alternate General Purpose Input/Output (GPIO) and mapped to the TIM2 using NVIC. When a capture is performed the current time from the ARR and puts the value into the Compare/Capture Count Register 1 (CCR1). CCR1 is the register that holds the time value for channel 1, which was selected to be the input capture channel. Once this value is put into the CRR1 register the clock continues to increase every microsecond still. How this capture is exactly performed with the ARR and CRR1 register can be seen in the hardware diagram in Figure 1.

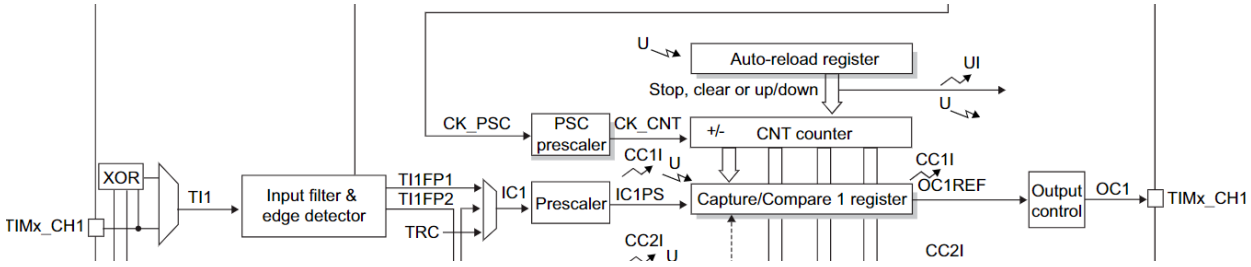

Figure 1: Hardware Diagram of Capture from Edge Detection on the Input Pin

To know when an input capture had been performed, a bit in the Status Register (SR), bit 9, corresponding to the Capture/Compare 1 overcapture flag is set 'High'. While the microcontroller is reading pulses from the wave it also is polling this flag to see if an input capture has been performed. If it has then that value from CCR1 is saved, then on the next 'High' that next value is saved. Next, the difference between the two time values is then saved into a Bucket. Bucket is a custom struct made to store the period that is read from the microcontroller

as well as the amount of times that period has been read in the 1000 pulses. This Bucket struct can be seen in Figure 2.

```
// Bucket structure to hold Bucket information
typedef struct Bucket {
    uint32_t period;
    uint32_t count;
} Bucket;
```

Figure 2: Source Code of Bucket Struct

These Bucket structs were put into a length 100 array which contained the frequency of (base - 50) to (base + 50). The base is inputted by the user into the terminal and received over UART which allows the program to know the expected period it will receive. When a period is read from the square wave input, the Bucket with the same period has its count increase by one. Once all of the 1000 pulses had been read the period and count of each Bucket was displayed to the terminal through UART given that they had at least one pulse of that period. To ensure that the periods read are correct, the count is totalled up and made sure to be at 1000 pulses or the expected amount. The program can then be run again, at the same expected period or a new expected period.

Test Plan

One of the biggest tests performed on the STM32 microcontroller was the Power-On Self Test (POST). The POST was configured to first make sure that there is an input into the PA0 Pin or the pin set up for TIM2. This checked to make sure that a rising edge was being detected on the pin from the inputted signal. Second, the POST made sure that the period of this inputted singal did not exceed 100 milliseconds. If either of these conditions failed then so did the POST, but if both passed the POST would succeed and the program would start and ask for the expected period.

Another test performed on the STM32 microcontroller was the test to make sure that expected period inputted into the microcontroller was within the range of 50Hz - 9950Hz. This was done by making sure the inputted period was between 100 and 20000. Outside of this range the microcontroller is not able to read the inputted square wave signal. If an input was outside of this range, then the microcontroller would run at the default expected period of 1000.

For testing the overall program on the STM32 microcontroller, square waves with varying periods were inputted into the microcontroller. Depending on the period being inputted into the microcontroller, first the expected value had to be changed so that the program knows what

period to expect. The square from a function generator was then put onto PA0 and the program ran. Once the 1000 pulses were read they were outputted to the terminal. The bucket's output should be in the range of +/- 50 from the expected period, as well as having the counts of all of the buckets add up to 1000. When this is true, then the program ran and read the pulses from the square wave successfully.

Project Results

When the project was fully completed, the STM32 was wired to the signal generator with a 1000 Hz frequency or an expected period of 1000. This is also the default period that the microcontroller reads. Once the POST was successful, the pulses were read at an expected period of 1000 and the results outputted to the terminal. The results from the terminal that contained the buckets can be seen below Figure 3.

```
Enter expected period or Press Enter if no change (DEFAULT 1000)
Expected period set to 1000
Reading Periods...

 BUCKETS
Period: 996, Count: 1
Period: 997, Count: 1
Period: 999, Count: 56
Period: 1000, Count: 856
Period: 1001, Count: 86
```

Figure 3: Bucket Results from 1000 Hz Square Wave Input

Once these results were printed to the screen the function generator was changed to make a square wave input with a Frequency of 500 Hz. This corresponds to a period of 2000 microseconds, which was inputted as the expected period into the microcontroller. Once the POST was successful the results from the terminal that contained the buckets were printed and can be seen below Figure 4.

```
Enter expected period or Press Enter if no change (DEFAULT 1000)
Expected period set to 2000
Reading Periods...

 BUCKETS
Period: 1996, Count: 1
Period: 1997, Count: 68
Period: 1998, Count: 483
Period: 1999, Count: 419
Period: 2000, Count: 29
```

Figure 4: Bucket Results from 500 Hz Square Wave Input

The main difference between the expected output and the actual output is that there is a lot more variation in the singal than what was expected. In the handout for this project there are only a total of 3 Buckets being printed. However, in both of the cases seen above in Figure 3 and 4, there are more than 3 buckets being printed to the terminal. This shows that the signal inputted into the STM32 microcontroller was more variable than what was expected on the handout for this project. This could also be due to the signal generator that was being used at the time of these captures. One expected result that is an actual result is that the number of pulses read from the signal generator on both frequencies is equal to 1000. Showing that the microcontroller is able to get 1000 pulses from the signal regardless of what the input frequency is. From these results above we can conclude that this project is a success and that the STM32 microcontroller can read a square wave input's period as long as it is within the frequency range.

Lessons Learned

One of the biggest lessons that was learned from this project was that the Timer peripheral is a very important and powerful tool. The timer has many different functionalities that can be used for many different purposes other than what was needed for this project. The use of these general purpose timers will be used throughout the course with different functionality. It was good to learn about how these timers work and the different functionalities each type of timer can provide.

This project also introduced the IOC which was used in this project to initialize many of the core peripherals used by the microcontroller. This IOC was good to learn because it will write its own code for you with your desired functionality allowing for seamless implementation.

A difficulty that emerged when creating this project was an issue with STMCube IDE. When using a flash drive device for the STM project I came across numerous problems. The first being that many of the project files like the .location or .project files were being deleted randomly and not opening the project so the project had to be remade many times. Second, it is important to not unplug the Flash Drive from the USB connector until STM Cube has fully been closed. Without doing this you risk the chance of having the entire Core folder with the source and include files deleted (This was not fun, I had to remake this project the night before due). From this project on, files for STM Cube IDE will only be stored on the SSD on my laptop to avoid further issues.