UNIVERSITY OF WATERLOO

ECE 750 Real-time Embedded System

Lab #2

Name: Shaocong Ren

Student number: 20478300

E-mail: s7ren@uwaterloo.ca

Q1. How the software loop(s) were implemented and calibrated. What were the values used?

A1. Null statement in C language cost the least time, I use for loop to control the number of times executing null statement. If the number of times N is right, the waiting time will just be 10ms as we desired. Actually, it is hard to evaluate the time for each null function, so it is hard to explain how many null statements we need.

Q2. How the hardware timer was implemented. What were the configuration parameters used and why?

A2:

1) I choose the **TIM2** among the 14 available timers on the STM32F4 and it is a general purpose timer whose default frequency is 84 MHz.
2) Then **prescaler** is set as 30, so the timers will count at a frequency of: **default frequency / prescaler = 84 MHz / 30 = 2.8 MHz**.
3) Parameter **period** is set as 28000, so the counter time will be: **period / real frequency = 28000/2.8MHz = 10ms**.
4) In the function TIM2_IRQHandler () GPIO Port A Pin 1 is set as low and high every time, such that the period between two rising edge is **10ms**.

Note that because of the system error, a parameter in the real project is different from the analysis above. Through the error correction, we get the real output the same as the expected one.

Q3 Pros and cons of each timing method. When would you use software loop(s) vs. hardware timers?

A3:

1) **Software pro**: The program is easy to understand and modify. When the program is written by assemble language, the resolution is relatively high. There is no upper limitation of the timing duration. Besides, the power consumption is lower than hardware timer since it doesn't use other hardware device.

   Software con: The accuracy is not high enough if the program is written by advanced language e.g. C language. It is hard to time accurately simply by modifying the times of loops. In a single process program, the CPU cannot do any other works when it is executing the loop timing method, which is a waste of computing resources, so that the system performance will become worse.

2) **Hardware pro**: It is very flexible for programmers to set the resolution and duration of the timer, just by modifying a few parameters. Different timers have different resolutions but most of the hardware timers have really high resolutions, which is independent from the CPU frequency. When the hardware is counting, the CPU is not occupied since timer and CPU are different devices they are

working independently as well. So the performance of CPU is much higher than that using software timing method.

Hardware con: Hardware timer is a device outside of the CPU, so both of the devices working will increase the power consumption. The hardware timer program contains many key bits and flags which requests that programmers must be familiar with the peripherals and instruction sets very well. Reloading the register will cost some time which decrease the accuracy of timing.

3) Software timing method is usually used when the system have some relatively lower requirements, e.g. timing resolution, accuracy, and system performance are not influenced by the loops very much. In these scenarios it is not worthy to use the hardware timers.

   Hardware timers are used widely, almost all the scenarios using software timing method can be implemented by hardware timers as well. When the system requests accurate timing and counting, high speed running of CPU, and no limitation of power consumption, hardware timer is the better method than using software one.

Q4. Were the rising edges being sent at exactly 10ms intervals? What are the sources of timing inaccuracies for each method?

A4. No, the periods are not exactly 10ms.

For the software timing method, the resolution is not high enough. If the parameter of for loops is changed by 1, the duration might be influenced significantly. Such that if the test period is large enough the error will be seen, specifically, if we want 1,000 rising edges and we get the same as that number, but if we expect 100,000 rising edges, in practice we might get 100,005. The testing criteria is another source of timing inaccuracy, since assume that the testing period is exactly 5s and 500 rising edges are expected, we can be sure that there are 499 intervals between the 1^{st} and 500^{th} edges but we cannot be sure the time consumption outside of the two edge which maybe 0 to 2 cycles.

For the hardware timer method, even though we calculate the parameters to get exact 3,000 rising edges, we may get 3,001 in practice using the corresponding parameters. Because 10ms is the timers counting period from beginning to it triggering interrupt, while the duration between the rising edges contains not only the timers counting time but also the time that program jumping from main function to the ISR and jumping back to main function. Besides, the timer overflows and triggering interrupts also need some time, this time is little but hard to calculate. Hardware timers can auto-reload

register, so the time cost during reloading the register is a source of inaccuracy. Furthermore, just like the analysis of software method, testing criteria is another source of error.