ENGIN 346: Microcontrollers

Engineering Department

University of Massachusetts, Boston

Semester: Fall 2020

Instructor: Dr. Honggang Zhang

Project Report for:

Project #4: Timer

By

Name: Tyler McKean

Student ID: 01098154

Date: November 24th, 2020

TABLE OF CONTENTS

NOTE: To automatically update ToC, right click above and select “Update Field”

Introduction

For Project 4, the list of goals for the project consisted of:

1. Understanding the basic concepts and functions of a timer
2. Handle different events in the interrupt service routine (ISR).
3. Handle the timer overflow and underflow

The main purpose of the lab is to use the onboard timers of the STM32L Discovery board, specifically, TIM1_CH2 initialized to PWM Output of pin PE.11 to trigger a pulse width to an ultrasonic sensor that will receive an Echo signal to PB.6 which will be initialized to Input Capture. The pulse width of the echo signal will be captured by setting up TIM4_CH1 to input capture both the rising and falling edges of the pulse width and establishing a TIM4_IRQHandler to calculate the length of the pulse width. The measured pulse width can then be calculated to a certain distance in centimeters that relates to an object displaced from the ultrasonic sensor. The code to accomplish this lab was thereby designed, implemented, and tested.

Background and Design

The timers on the STM32L Discovery board can be programmed to both PWM Output and Input Capture. Pulse width modulation (PWM) is a simple digital technique to control the value of an analog variable and uses a rectangular waveform to quickly switch a voltage source on and off to produce a desired average voltage output [1]. For this project, we were tasked with using TIM1_CH2, which is the alternative function of GPIO pin PE.11, to output a pulse width of 10µs and amplitude of at least 3V that would trigger the ultrasonic sensor. The sensor itself needs a pulse width of at least 10µs to activate it. The sensor will then emit an ultrasonic burst of 40kHz that will then return an echo signal that looks like a square wave that is proportional to the distance to the nearest object. This square wave's length should range between 150µs to 25ms, or 38ms if nothing is in range. The sensor also requires a 5V supply to its VCC pin and needed to be tied to a common ground by its GND pin. GPIO pin PE.11 was tied to the Trigger pin on the sensor and the Echo pin was then tied to GPIO pin PB.6. To observe the echo signal, TIM4_CH1, the alternative function of GPIO pin PB.6, was initialized to Input Capture the rising and falling edges of the echo signal to find the timespan of the pulse width. A connection and configuration diagram for the lab can be seen in Figure 1 below.

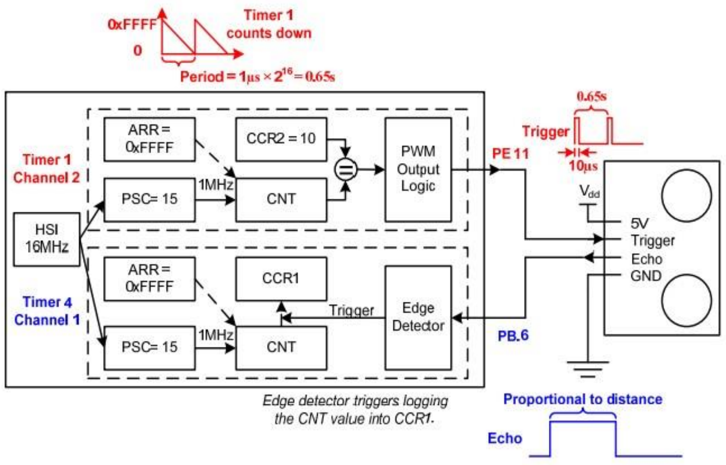

Figure 1. Connection and configuration diagram for interfacing ultrasonic distance sensor

For both timers, we were given a recommended ARR value of 0xFFFF, which is the highest value in the 16-bit counter, and a Prescaler value of 15. The Discovery board was initialized to run its 16MHz high-speed clock, so the Prescaler value would then frequency divide the HSI to run at (16MHz/(1 + PSC)) = 1MHz. TIM1_CH2 was suggested to be configured as Down-counting and to set the CCR2 value as 10. The timer is then configured to count down from 0xFFFF to zero, and every step downward would be at (1/1MHz) so 1µs. Since the CCR2 value was set to 10, that means that when the CNT reaches 10 and counts down to zero, the external signal would then be active for 10µs until it reaches an underflow, and results in a 10µs pulse width output to trigger the ultrasonic sensor.

When the ultrasonic sensor returns an echo signal to GPIO pin PB.6, the signal then runs through an edge detector which is programmed to capture both the rising and falling edges on the signal. It's worth mentioning that TIM4_CH1 was also configured with the same ARR value of 0xFFFF and PSC value of 15, thus TIM4 was also initialized to run a 1MHz clock speed, similar to TIM1. Software was set to update the interrupt enable flag (UIE) and the capture event flag (CC1E) in the DMA/interrupt enable register (DIER), which would allow these interrupts to occur. A TIM4_IRQHandler was then written into the code to check when these flags were updated and to calculate the CCR1 value that was detected from a corresponding rising or falling edge. If these flags were not equal to zero and if the signal polarity was low, then the pulse width was calculated from the following equation:

$$\text{Timespan}(\mu s) = ((CCR_{NEW} - CCR_{LAST}) + (1 + \text{ARR\_Val}) * \text{Underflows})) \qquad \text{(Eq 1-1)}$$

When the edge detector observed a rising or falling edge the value in the CCR was saved to the variable CCR_{NEW}, and when it reached another edge, the code reassigns the previous value to CCR_{LAST} and saves the new current value into CCR_{NEW}. The equation then subtracts the difference from these edges and adds it with the value (1 + 0xFFFF). Another variable called underflows needed to be added to the equation to account for the number of underflows that occurred within one pulse. After calculating the pulse width, the current value of the underflows variable was reset to zero, in preparation of the next observed pulse width. The distance could then be calculated by the following equation:

$$\text{Distance} = (\text{Pulse Width } (\mu s) / 58)\ cm \qquad \text{(Eq 1-2)}$$

So the TIM4_IRQHandler was programmed to check when any of the interrupt flags were enabled, calculate the pulse width (in µs) if the flags were enabled, and clears the flags as to not mistakenly keep running the interrupt request, and the main program can resume. The equation for calculating the distance was used in the Display_Centimeters function which needed to multiple the pulse width by the reciprocal of the counter clock frequency (1/1MHz) and then multiple the value by 1000 to achieve a value in the milliseconds. The distance value is then calculated and then displayed onto the LCD to two decimal places.

EXPERIMENT AND RESULTS

This section of the report will discuss what pins and registers of the Discovery board needed to be initialized to achieve the project goals. Starting with the GPIO pins, for both PE.11 and PB.6 separate functions called TIM1_C2_Init() and TIM4_C1_Int() were defined outside the main function to initialize the pins and timers. Both functions required enabling the GPIO clocks via the AHB2ENR for GPIO port B and E, as well as the APB2ENR clock for timer 1 and the APB1ENR1 for timer 4. Both pins needed to be set to their Alternative Functions in the MODER by have the value 0x2 set to the corresponding bits of PE.11 and PB.6. TIM1_CH2 was enabled by setting PE.11 to its AF1 and TIM4_CH1 was enabled by setting PB.6 to its AF2. Both pins were then set to high speed (16MHz HSI) in the OSPEEDR and then set to no pull-up/no pull-down in the PUPDR for the corresponding bits for each pin. Both TIM1 and TIM4 had the same ARR value of 0xFFFF and PSC value of 15 set for each timer.

To continue initializing TIM1, the direction was set to down-counting in the TIM_CR1_DIR like the suggested timer configuration asked for. The output compare mode bits were then cleared in the TIM1->CCMR1 by clearing TIM_CCMR1_OC2M. In order to initialize PWM Mode 1 output for channel 2 an OR expression was used on TIM_CCMR1_OC2M_1 and TIM_CCMR1_OC2M_2 in the CCMR1. The output 2 preload enable was cleared by TIM_CCMR1_OC2PE and then the output polarity was set as active high in the CCER by clearing TIM_CCER_CC2P. The output of channel 2 was then enabled by initializing TIM_CCER_CC2E in the CCER and enabling the main output enable (MOE) by TIM_BDTR_MOE in the BDTR. Finally, the CCR2 value was set to 10 to achieve the 10µs pulse to the trigger input of the sensor and then counter for TIM1 was enabled by TIM_CR1_CEN.

To finish up initializing TIM4, the direction for the counter was also set to down-counting by enabling TIM_CR1_DIR. The direction was set as an active input and CC1 was mapped on Timer Input 1 by clearing the TIM_CCMR1_CC1S and setting the value to 0x1 by enabling the TIM_CCMR1_CC1S_0 bit. Digital filtering was then disabled by clearing the TIM_CCMR1_IC1F bit and to ensure that the edge detector captured every event the TIM_CCRM1_IC1PSC bit was cleared as well. To select the edge of active transition as both rising and falling edges, both the TIM_CCER_CC1P and TIM_CCER_CC1NP bits were executed with an OR expression in the CCER. Capture/compare for CC1 was then enabled by the TIM_CCER_CC1E bit set as 0x1. The following related interrupts were then enabled: TIM_DIER_CC1IE to enable capture/compare interrupts for channel 1, TIM_DIER_UIE to enable update interrupts for underflows during the down-counting, and TIM_CR1_Cen to enable the counter for TIM4. The NVIC was then called upon to set the priority for TIM4 by NVIC_SetPriority(TIM4_IRQn, 1), which set the priority to 1 and NVIC_EnableIRQ(TIM4_IRQn) to enable the TIM4 interrupt in the NVIC.

The TIM4_IRQHandler was initialized by checking if the interrupt flag in the SR was not equal to zero. If this statement of the if loop were true then the variable current_captured would read the value from the CCR1, which also clears the CC1IF, and then toggles the signal polarity flag. Another if loop was used to check whether the signal polarity flag was equal to zero, which meant it was active high in this moment, and calculated the timespan of the pulse by using (Eq 1-1). The underflow_counter was then reset to zero before exiting the nested if loop, and then the last captured variable was updated from the previous current_captured variable value. The TIM4_IRQHandler then checks in another if loop whether the update interrupt flag was set. Again, this would occur when an underflow has taken place, and thus the code clears the TIM_SR_UIF bit to prevent re-entering the handler and increments the value for the underflow by 1.

The main function of the program has then been initialized with almost all the appropriate functions setup accordingly, all of which are called in the header of the C program. The main program runs through the System_Clock_Int() function which sets the internal clock to the 16MHz HSI as mentioned previously. The TIM1_C2_Int() and TIM4_C1_Int() functions then established the PWM Mode 1 output of PE.11 and Input Capture of PB.6. The LCD is then initialized by the functions: LCD_Initialization() and LCD_Clear(). An infinite while loop is then coded into the program to run the Display_Centimeters() and Delay() function as the ultrasonic sensor is constantly detecting whether there is an object at some distance in front of it. The Display_Centimeters() function then calculates the pulse width obtained from the TIM4_IRQHandler and then uses (Eq 1-2) to calculate the distance. The float value of the distance is then converted to a string which is then displayed to the LCD via the LCD_DisplayString() function. Finally, the Delay function is set to 100, which updates the value of the LCD to a time increment of every 100ms. This concludes the code needed to initialize the program to meet the project goals.

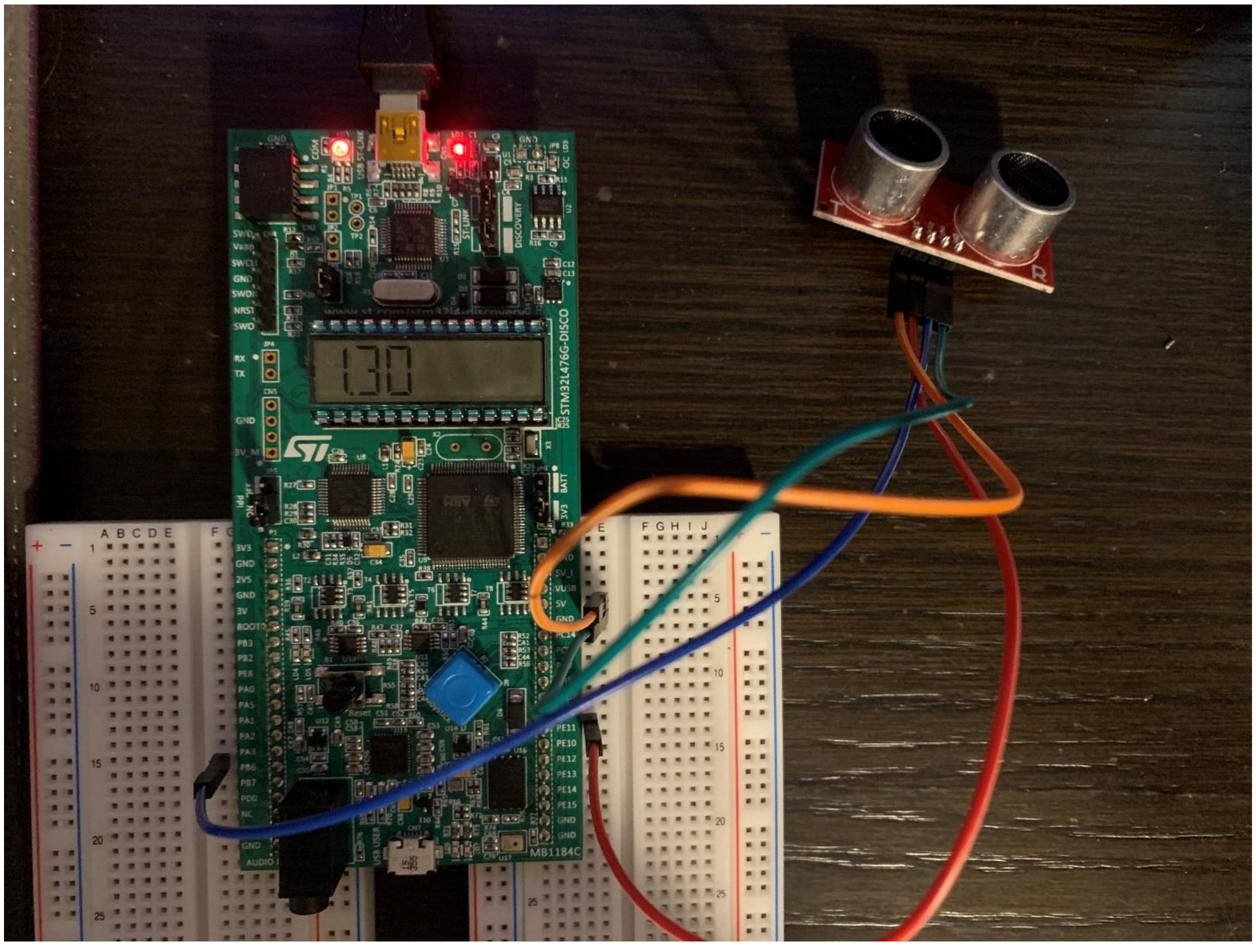

Figure 2 - Discovery Board setup for Project 4

Observations and Conclusions

To aid in the observation process of this project, I used the Analog Discover 2 oscilloscope to capture the PWM output that was being used to trigger the ultrasonic sensor from PE.11. Screenshots from the oscilloscope can be seen below.

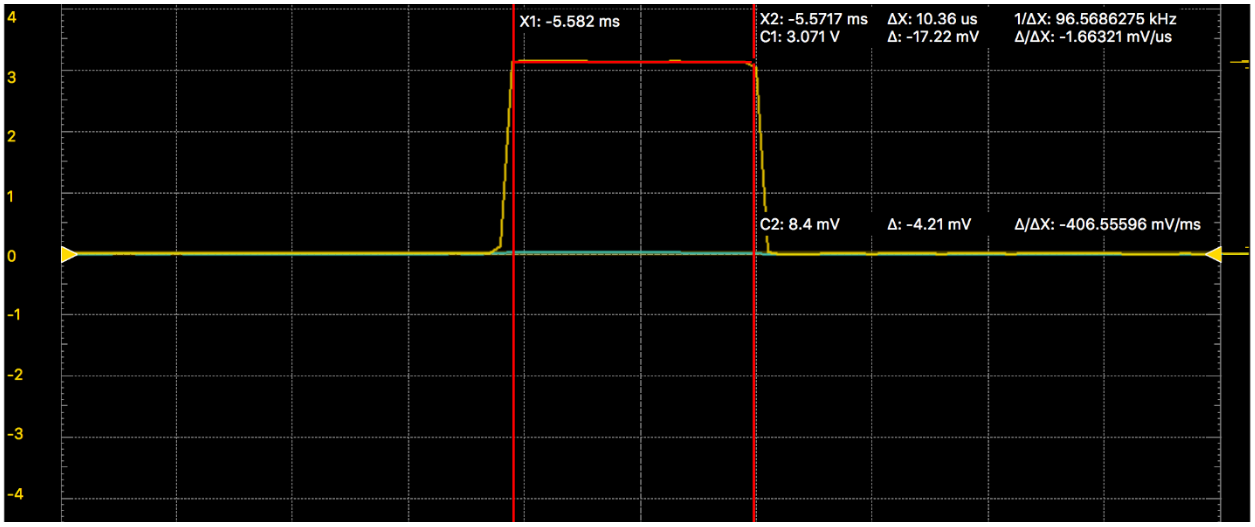

Figure 3 - 10µs pulse coming from pin PE.11's TIM1_CH2 PWM Mode 1 output

The ΔX measurement in the top right of the imagine reads 10.36µs, which shows that PE.11 was outputting a 10µs pulse.

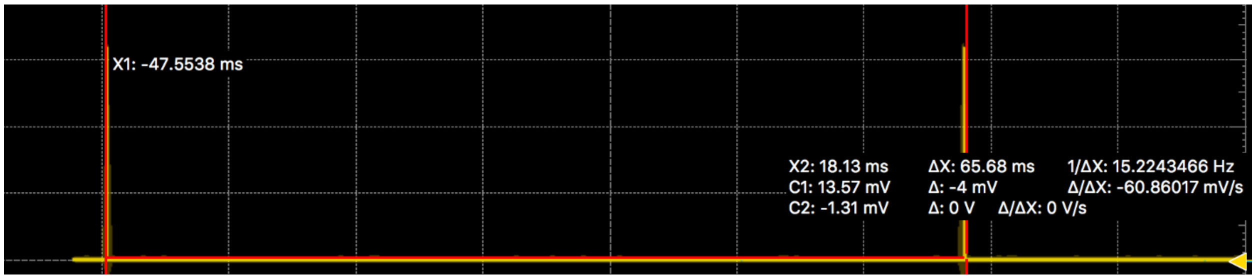

Figure 4 - Oscilloscope measurement of the Period of Pe.11's TIM1_CH2 PWM output

Here the oscilloscope shows that the period between 10µs pulses was around 65.7ms, which due to the counter clock set to a frequency of 1MHz, then the calculated period of should be ($1\mu s * 2^{16}$) = 65.54ms, which is close to the measurement taken from the AD2.

A testing video for the ultrasonic sensor is included in the zipped Project 4 folder

First, pressing the reset button on the Discovery board will reset and wipe away the values from the LCD. When testing the ultrasonic sensor, I noticed the closest value that the LCD reads was about 1.13cm with my hand completely covering the sensor. When I slowly lift my hand away from the sensor, the reading increments in an inaccurate step size. The increments seem to be around 2-2.5cms away when the LCD will increment from 1.13 to 1.14, then 1.15 and so on. I assume that the inaccuracy of the step size is coming from the TIM4_IRQHandler not be able to accurately read both the rising and falling edges of the echo signal pulse width. This project asks us to only use TIM4_CH1 to observe the incoming echo pulse width, but I think if we could utilize channel 1 of TIM4 to only read the rising edges and used channel 2 to read the falling edges, the Handler would be able to capture a more accurate pulse width reading. This would be done by setting the current_captured variable to read the value from CCR1 and then last_captured variable to read the value from CCR2. Doing so may would present an alternate solution to this project's inaccurate distance LCD readings, however it was not part of the suggested configuration of the project.

BIBLIOGRAPHY

[1] Zhu, Y. (2018). *Embedded systems with ARM Cortex-M microcontrollers in assembly language and C* (pp. 384-412). United States of America: E-Man Press LLC.