CpE5151 Experiment #2
Ultrasonic Distance Measurement

Goals:
1) Learn about using a timer to generate waveforms
2) Learn about using timer/counter capture mode to make a measurement
3) Write assembly functions to calculate a result from a measurement
4) Write assembly functions that interface with C source code (passing and returning values).

Grading Rubric:

Name	Description	Points
Functionality	Works as specified in each step of the experiment	60
Organization	Organized into functions/subroutines where appropriate. Uses decision and repetition structures appropriately. Guidelines for structured programming are followed such as avoiding interconnections in the source code and having one entry and one exit point in a structure.	10
Readability	Source code is well commented. Appropriate condition checks are used. Labels and variable names help explain the source code.	10
Correctness	Appropriate calculations are used and match the source code. Comments match the source code. No patches are used cover errors in the source code.	10
Total		90

Equipment Needed:
1) STM32L4R5 Nucleo-144 development board
2) Micro-B USB Cable
3) Arduino adapter board with expansion connector and cable
4) Ultrasonic Transmitter/Receiver circuit
5) Three Output Power Supply: +10V/-10V/+5V and cables
6) Keil uVision5 project example for Experiment #2

Procedure:

The power supply does not need to be switched on for this part of the experiment.

1) (Functionality: 8pts) Write an initialization function for Timer 3 that creates a 40KHz waveform on CH1 which is output on Port A, Bit 6 (PA6). The initialization function should:
    a. Enable the 4MHz system clock for TIM3 and GPIOA
    b. Set up PA6 as Alternate Function 2 (TIM3_CH1) with a push-pull output and pull-up/pull-down resistors disabled.
    c. Set TIM3, CH1 to PWM2 mode (inactive to CCR1 match and active to ARR match) using the TIM_CCMR1 register.

d. Set the TIM_CCR1 and TIM_ARR registers to generate a 40KHz waveform at 50% duty cycle (TIM_ARR sets the frequency and TIM_CCR1=TIM_ARR/2 sets the duty cycle to 50%).
e. Use the TIM_CCER register to enable the CH1 output with an active high output.
f. Start TIM3 running by setting the counter enable bit (CEN) which is bit 0 in the control register (TIM_CR1).

Call this initialization function from assembly code or C and verify that PA6 has a 40KHz waveform present using an oscilloscope. PA6 is digital pin 12 (where the orange wire is connected) on the Arduino adapter board.

2) (Functionality: 10pts) Write an initialization function for Timer 4 that creates a 106usec high pulse on CH3, which is output on Port D, Bit 14 (PD14). This will be used to control the 40KHz waveform so that a burst of four pulses can be transmitted. The initialization function should:
   a. Enable the 4MHz system clock for TIM4 and GPIOD.
   b. Set up PD14 as Alternate Function 2 (TIM4_CH3) with a push-pull output and pull-up/pull-down resistors disabled.
   c. Set TIM4, CH3 to PWM2 mode (inactive to CCR1 match and active to ARR match) using the TIM_CCMR2 register. This will result in a delay before the pulse that is determined by the value in TIM_CCR3. Keep TIM_CCR3 small so that it will not affect the measurement, but it cannot be zero (I used 10).
   d. Set the TIM_CCR3 and TIM_ARR registers to create a 106usec pulse. The number of counts between TIM_ARR and TIM_CCR3 multiplied by the system clock frequency (assuming no prescaler is used) determines the pulse length.
   e. Use the TIM_CCER register to enable the CH3 output with an active high output.
   f. Set TIM4 to One Pulse Mode by setting the OPM bit, which is bit 3 in the control register (TIM_CR1). The counter enable bit does not need to be set.
   g. A trigger for TIM4 needs to be set to start the timer to generate the pulse. In order to do this a trigger out (TRGO) on a master and a trigger in (TRGI) on the slave (TIM4) must be defined. Also, the action that the slave will take must also be defined. For this part of the experiment, TIM3 will be the master and TRGO defaults to be an update generation, which occurs when the UG bit (bit 0) of the Event Generation register (TIM_EGR) is set. This also causes the count value of that timer to be reset to zero. The Slave Mode Control register (TIM_SMCR) for TIM4 is used to select the TRGI and action. Choose the appropriate internal trigger from Table 280 on page 1353 of the STM32L4 reference manual for TIM4 to be triggered by TIM3. The action should be set to Trigger (start the timer) or Reset and Trigger. Note that I did not test Reset and Trigger, so I cannot guarantee that it will work.

3) Create a program in assembly or C that calls the initialization functions for TIM3 and TIM4. Also call the initialization function for the USER pushbutton switch from experiment #1, so that it can be used to start the timers. Create an endless loop for the

main loop. It should start by reading the USER pushbutton switch repeatedly until it is pressed (Hint: this sounds like a repeat-until, a.k.a do-while loop, would work well here). When the switch is pressed, then reset TIM4 (write to the UG bit in TIM_EGR) and then reset TIM3 (note: when the UG bit is written for TIM3, this will cause TIM4 to start).

Verify that a 106usec high pulse is generated on PD14 (digital pin 10 on the Arduino adapter board) when the switch is pressed. Using NORMAL trigger mode or pressing the SINGLE button will allow you to capture and hold the single pulse. Make sure your trigger source is the single pulse and not the 40KHz waveform which is continuously running. The RUN/STOP button will allow you to capture another pulse, if needed. Measure this pulse and verify that it is about 106usec long.

4) (Functionality: 3pts) Modify the initialization functions for TIM3 and TIM4 so that the 106usec high pulse is used to control the 40KHz waveform so that only four pulses are generated. The following modifications are needed:
    a. Change the default TRGO output for TIM4 to the CH3 output (called OC3REF in the datasheet). This is done using the TIM_CR2 register (page 1350 of the STM32L4 Reference Manual). Do not change the TIM3 TRGO, it will still be the Reset (UG) signal.
    b. Use the Slave Mode Control register (TIM_SMCR) of TIM3 to select a TRGI and action. The TRGI should be set to come from TIM4 (see Table 280 on page 1353 of the STM32L4 reference manual). The action should be set to Gated Mode which will use the 106usec pulse to control when the TIM3 counter is enabled.

       After these changes have been made, verify that only four pulses occur on the TIM3, CH1 output (PA6 or digital pin 12). The signal should be low both before and after the pulses occur.

5) (Functionality: 10pts) Another pulse that is 1ms long and active low is needed. This pulse will be generated by Timer 5. Write an initialization function for Timer 5 that creates a 1msec low pulse on CH4, which is output on Port A, Bit 3 (PA3). This will be used to block the transmitting signal from reaching the capture input. The initialization function should:
    a. Enable the 4MHz system clock for TIM5 and GPIOA.
    b. Set up PA3 as Alternate Function 2 (TIM5_CH4) with a push-pull output and pull-up/pull-down resistors disabled.
    c. Set TIM5, CH4 to PWM2 mode (inactive to CCR1 match and active to ARR match) using the TIM_CCMR2 register. This will result in a delay before the pulse that is determined by the value in TIM_CCR4. Keep TIM_CCR4 small so that it will not affect the measurement, but it cannot be zero (I used 10). It should be equal to or smaller than the value used for the 106usec pulse so that it can be asserted before the four pulses are transmitted.

d. Set the TIM_CCR4 and TIM_ARR registers to create a 1msec pulse. The number of counts between TIM_ARR and TIM_CCR4 multiplied by the system clock frequency (assuming no prescaler is used) determines the pulse length.
e. Use the TIM_CCER register to enable the CH4 output with an active low output.
f. Set TIM5 to One Pulse Mode by setting the OPM bit, which is bit 3 in the control register (TIM_CR1). The counter enable bit does not need to be set.
g. The trigger for TIM5 should be the same as TIM4. Use the same TRGO as was used for TIM4. The TRGI will be set to the same timer as TIM4. Since this is a different timer, Table 280 on page 1353 of the STM32L4 reference manual should be checked to find the appropriate TGRI value. The action should be set to the same action as TIM4.

Call the TIM5 initialization function along with the initialization functions for TIM3 and TIM4. In the endless loop, after the switch is pressed, reset TIM5 along with TIM4. When TIM3 is reset, it will start TIM4 and TIM5. Verify that the 1ms low pulse is generated on PA3 (Analog pin 0). Also verify that it is low when the four 40KHz pulses are generated.

The next section will set up a timer to measure the time from when the four pulses are sent to when the echo is received. A timer will be set up for input capture. It is started (or reset) at the same time that the pulses are sent. The capture flag is polled until it is set. Then the captured value is read from the capture/compare register and then used to determine the distance. Note that the power supply must be switched on for the echo to be received properly. The power supply should be set as follows: supply1=+10V (white), supply2=-10V (yellow), supply3=+5V (red). Make sure that the power supply outputs are switched on.

6) (Functionality: 7pts) Write an initialization function for Timer 2 to set it up for input capture. The input capture channel is CH1, which is on Port A, Bit 5 (PA5). The initialization function should:
   a. Enable the 4MHz system clock for TIM2 and GPIOA.
   b. Set up PA2 as Alternate Function 1 (TIM2_CH1) with pull-up/pull-down resistors disabled.
   c. Set TIM2, CH1 to Input TI1 mode (no filter or prescale will be used) using the TIM_CCMR1 register.
   d. Use the TIM_CCER register to enable the CH1 input with either rising or falling edge causing a capture.
   e. Enable TIM2 by setting the counter enable bit (CEN), which is bit 0 in the control register (TIM_CR1).
   f. The trigger for TIM2 should be the same as TIM4 and TIM5. Use the same TRGO as was used for TIM4 and TIM5. The TRGI will be set to the same timer as TIM4 and TIM5. Since this is a different timer, Table 280 on page 1353 of the STM32L4 reference manual should be checked to find the appropriate TGRI value. The action should be to reset the count value so that it starts at zero for

the measurement.

Another option is to use the reset of TIM2, instead of the reset of TIM3, to start the measurement. The value for the TIM_SMCR in each timer initialization function would need to be rewritten to use TIM2 as the TRGI instead of TIM3. Make sure to add the reset of TIM2 after resetting TIM4, TIM5 and TIM3.

7) (Functionality: 5pts) Call the Timer 2 initialization function along with the other timer initialization functions. In the endless loop, when the USER pushbutton switch is pressed, the reset of TIM3 (or TIM2 if that option is used) will cause the measurement to start. When an echo is received, the capture/compare flag (CC1IF, bit 1) in TIM_SR will be set (see page 1355 of the STM32L4 reference manual for a description of this register). A loop should be created to repeatedly read this register and check if the CC1IF bit is set. When the bit is set, exit the loop and read the captured value from TIM_CCR1. This value is the number of counts from when the measurement started to when the echo was received.

   When a loop is created, the programmer must consider the possibility of an endless occurring because of an error condition. This could happen in this project if an echo is not received. This could occur if the distance being measured is too far (estimate that 50 feet would be too far) or if the object is at an angle and the echo is not directed back to the receiver. A timeout out should be placed in this loop to exit in the event an echo is not received. An easy method is to place a value in a register and decrement the register each time through the loop. The timeout occurs if the register reaches zero. The value in the register should be large enough that the farthest echo will be received, but not so large that the user has to wait for the error timeout. Another possible timeout is to read the timer 2 count value and if it reaches a value that is too big, then exit with an error. Another possibility is to check the UIF bit, which is set when an overflow or update occurs. The problem with this method is that it takes over 16 seconds for an overflow to occur when using a 4MHz clock, which is too long of a wait for an error to occur.

8) (Functionality: 5pts) Write a function in ARM assembly that has an input parameter of the number of counts measured by timer 2 and returns the distance as a fixed point integer in q8 format. The calculation can be done with one multiplication, but this function should have at least one multiplication and one division for the experience of using these instructions. The calculation is given below:

$$distance = (counts * speed\ of\ sound)/(2\ *\ system\ clock)$$

   Dividing the number of counts by the system clock frequency gives the time in seconds that it took for the 40KHz burst to travel to an object and return. Dividing that number by 2 would give the time to travel to the object. Multiplying this value by the speed of sound would then give the distance. However, if the division is performed first using integer

division, then the result will be zero because the time is milliseconds. Therefore, the multiplication by the speed of sound should be first. Use: 1125 feet/second or 343 meters/second, for the speed of sound.

Since we are using integer division, if this value is divided by 2xsystem clock, the result will give feet or meters with no fractional portion (i.e 5.9 feet would give a result of 5 feet or 6.1 feet would give a result of 6 feet). The specification given above is to return a result in q8 format (i.e the result should have an 8-bit fractional portion). This amount of precision should allow the programmer to give a measurement to at least 0.1 foot or 1 inch. There are two ways to get the q8 result. The first is too multiply (scale) the numerator to q8 format by multiplying 256 before the division. The problem with this approach is that it is possible that the resulting numerator would exceed 32-bits, resulting in a more complex solution. The second method is to divide the denominator (2*system clock) by 256. This value is a constant, so the programmer just needs to multiply by the speed of sound and then divide by this constant.

One of the problems with integer arithmetic is truncation or round-off error. As mentioned earlier, a result of 5.9 is truncated to 5. An integer division can be rounded by adding 0.5*denominator to the numerator before the division. For example: if a value is to be divided by 128, then add 64 to the numerator before dividing. The programmer can add rounding to the calculation for distance if they prefer.

9) (Functionality: 4pts) This part of the experiment can be written in C. The number in q8 format consists of an integer portion (upper 24-bits) and a fractional portion (lowest 8-bits). Right shifting by eight to remove the fractional portion can separate the integer portion. This can be printed as any other integer would be printed. The fractional portion can be separated by ANDing with 0xFF. The fraction portion represents a value/256. This value needs to be rescaled to tenths or hundredths of feet or to inches to be meaningful to a user. It can be rescaled by solving the following equation for Y:

$$\frac{value}{256} = \frac{Y}{resolution}$$

Where resolution is: 10 for tenths of feet, 100 for hundredths of feet or 12 for inches. This can be printed as an integer, such as “integer_portion.Y feet” (sprintf(“%u.%2.2u feet \n”, integer_portion, Y);) or “integer_portion feet, Y inches.”

10) (Functionality: 3pts) The final step is to output the measurement. This will be done by printing to Putty or some other terminal emulator. The ST-Link debugger has a virtual serial port connected to the low-power UART (LPUART1) of the STM32L4R5ZI processor. The example code for this project has the LPUART1 initialization code and a function for printing a character string.
    a. The first step for using LPUART1 to print the measurement is to determine which COM port on the PC is connected to the ST-Link virtual com port. Click on the

USB device icon that is used to eject a USB device to bring up a menu. Click on the “Open Devices and Printers” option. There should be at least one unspecified device. Right click on this device and select properties. Then click on the “Hardware” tab. In the name window, locate the ST-Link Virtual COM port (COM??) and determine what number is listed. You may need to resize this window to see the number.

b. Open Putty (or your preferred terminal emulator) and click the serial connection type. Change the COM port to the number found above and click on “OPEN” If you get an error saying it cannot open the port, then go back and look or the COM port number again.
c. There is a “Hello World!” test string in the example code. Verify that it prints to Putty when the code executes.
d. Then ese sprintf(print_buffer_name, ”string to print”); to create a formatted output of the measurement. Then use the HAL_UART_Transmit function to print the string to putty using the LPUART.

   Verify that you can press the button and print measurements to Putty. The ceiling is about 6 feet from the table. You can use a book held over the transmitter/receiver circuit to change the distance. If you are not getting accurate measurements, verify that the power supply is on and the outputs are switched on.

   (5pts) Demonstrate the experiment.

The experiment source code files (as a Keil uVision5 project) should placed in a compressed folder (.zip file) and uploaded to Canvas. Make sure each member of the team has their name in the comments.