CpE5151 Experiment #3
Interrupt Based Ultrasonic Distance Measurement

Goals:
1) Learn about using the ARM Nested Vectored Interrupt Controller
3) Write an interrupt handler for Timer 2
4) Write a simple scheduler to take a distance measurement and output the result.

Grading Rubric:

Name	Description	Points
Functionality	Works as specified in each step of the experiment	35
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		65

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 #3

Procedure:

1) Compile the Experiment #3 example project and verify that it works and prints an output string to Putty. You can modify the number of milliseconds that TIME must be greater than before it resets to zero to control the cycle time. You can add other actions, such as writing the index_g value to the four LED display. The digital output connections are given below:

Port and Pins	Control Function	Oscilloscope Connection
PF15 to PF12	DISPLAY_UPDATE(uint8_t)	D3 to D0
PD8	Up to programmer	D4
PD9	Up to programmer	D5
PD14	Up to programmer	D6
PE9	LED_CONTROL(uint8_t)	D7

2) (Functionality: 4pts) Write an assembly function to enable the TIM2_CC1 interrupt. This function should perform the following tasks:
    a. Save ALL registers altered by the function by pushing them onto the stack as the first instruction in the function.
    b. Set the base address so that TIM2 is being accessed.
    c. Write a 0 to the TIM_SR to clear all of the pending interrupts.
    d. Read the TIM_DIER register and set the CC1IE bit to enable the interrupt. Write the modified value back into the TIM_DIER register.
    e. Restore the registers saved on the stack by popping them back off of the stack.
    f. Return from the function.

3) (Functionality: 3pts) Write an assembly function to disable the TIM2_CC1 interrupt. This function should perform the following tasks:
    a. Save ALL registers altered by the function by pushing them onto the stack as the first instruction in the function.
    b. Set the base address so that TIM2 is being accessed.
    c. Read the TIM_DIER register and clear the CC1IE bit to disable the interrupt. Write the modified value back into the TIM_DIER register.
    d. Restore the registers saved on the stack by popping them back off of the stack.
    e. Return from the function.

4) (Functionality: 3pts) Write an assembly function to read the TIM2_CCR1 value. This function should perform the following tasks:
    a. Save the registers altered by the function except R0 by pushing them onto the stack as the first instruction in the function.
    b. Set the base address so that TIM2 is being accessed.
    c. Read the TIM_CCR1 register so it can be returned by the function.
    d. Restore the registers saved on the stack by popping them back off of the stack.
    e. Return from the function.

5) (Functionality: 4pts) Write an assembly function to reset and then trigger the timers for a distance measurement. The timer initialization functions included with the example code will reset TIM2 and trigger TIM4 and TIM5 when TIM3 is reset. TIM4 and TIM5 will need to be reset manually or the programmer can modify the initialization functions so that these reset and trigger when TIM3 is reset. This function should perform the following tasks:
    a. Save ALL registers altered by the function by pushing them onto the stack as the first instruction in the function.
    b. Set the base address so that TIM4 is being accessed.
    c. Write a 1 to the UG bit in TIM_EGR to reset TIM4.
    d. Set the base address so that TIM5 is being accessed.
    e. Write a 1 to the UG bit in TIM_EGR to reset TIM5.

f. Set the base address so that TIM3 is being accessed.
g. Write a 1 to the UG bit in TIM_EGR to reset TIM3. This should reset TIM2 and trigger TIM4 and TIM5 to start a distance measurement.
h. Restore the registers saved on the stack by popping them back off of the stack.
i. Return from the function.

6) (Functionality: 3pts) Write the interrupt service routine (interrupt handler) for timer 2. It MUST be named "void TIM2_IRQHandler(void)" or you will need to make all of the changes needed to update the interrupt vector table in "startup_stm32l4r5xx.s" with the new interrupt handler name. The interrupt handler should do the following two things:
   a. Read the value from TIM_CCR1 value into a global variable using the assembly function created in step 4.
   b. Disable the TIM_CC1 interrupt using the assembly function created in step 3. This will result in only one value per distance measurement which will make processing easier.

7) (Functionality: 2pts) Add the HAL functions to set the priority of the timer 2 interrupt and enable the timer 2 interrupt in the NVIC to the main.c initialization area. See the lecture notes or the STM32 Programming Manual (page 209) for the HAL function names.

8) (Functionality: 3pts) Add the TIM_CC1 enable function and the start measurement function to the scheduler in the SysTick_Handler. The print statement in the example code can be removed.

   Place a breakpoint in the TIM2_IRQHandler. Compile and debug the code. Verify that code execution stops at the breakpoint. If it does not, check the peripheral registers for the NVIC and TIM2 to make sure the interrupt is enabled. Check the waveforms on the oscilloscope to verify that the distance measurement waveforms look correct. If the code execution does stop at the breakpoint, then use the variable watch window to view the value read from TIM_CCR1 and verify that the value is okay. You may need to take more than one measurement to verify this value.

9) (Functionality: 7pts) Add to the scheduler in the SysTick Handler: a check to see if a measurement has been made after an appropriate delay from when the measurement was started (i.e give enough time for the echo to be received). I used the global value that was read from TIM_CCR1 when the TIM2 interrupt handler executes and set it to some default value (like 0xFFFFFFFF) when the measurement was started. Then I checked to see if it had changed when the echo was received (i.e. if the TIM2 Handler executes, the value was read from TIM_CCR1). Instead, you could create a global variable to be used as a flag to indicate when a measurement was made (clear the flag when the measurement is started and set it in the TIM2_Handler). Verify that the scheduler is working and that an appropriate measurement value is accessible in the SysTick handler. Add a fixed-point calculation to convert the measured value from counts to feet or meters. See experiment #2 for the calculation. Verify that the

measurement is around 6 feet or 1.8 meters. If a measurement was not made, then the calculation should not take place.

10) (Functionality: 6pts) In the scheduler: add a step, a few milliseconds after the measurement value has been checked and calculated, that prints the result using the LPUART. This is similar to the output from experiment #2 except that the "HAL_UART_Transmit_IT" function should be used. Make sure there is enough time to print the output string before another measurement is started (i.e make sure the cycle time in the scheduler is long enough for the entire string to print (at least 1ms per character at 9600 baud)). If no echo was received and the measurement value is still at its default value, then an error message should be printed instead of the measurement. The partial setup stations on rc06aece210 to rc08aece210 are useful for testing this error handling, since they will never receive an echo.

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