Lab 4 - Ultrasonic Alarm

Due Jul 31 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** txt
Attempts 1 **Allowed Attempts** 2

Introduction

In this lab we will be using the timer peripheral to control the ultrasonic range finder to create an invisible tripwire that sounds an alarm when crossed.

Objective

This lab will exercise your ability to configure the timer peripheral and clock sources to generate precise pulses and read elapsed time. You will also learn how to drive the piezoelectric buzzer to generate an obnoxious buzzing noise.

Required Materials

- MSP-EXP430FR2355
- Seeed Studio Grove Ultrasonic Distance Sensor
- Seeed Studio Grove Buzzer
- Seeed Studio Grove Base Booster Pack
- Grove connection cable
- USB micro cable
- Laptop (Windows preferable)

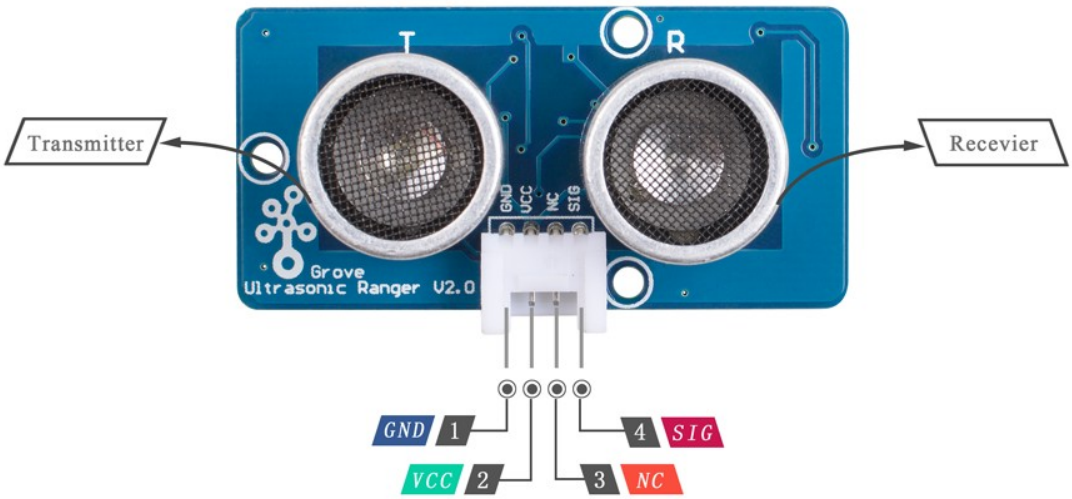
Procedure

Step 1: Understanding the Ultrasonic Range Finder

The ultrasonic range finder emits an ultrasonic sound wave and measures the time it takes to bounce

off something in the environment to determine distance. Since we know the speed of sound, we can determine the distance to an object based on how long it takes the sound wave to return to the receiver.

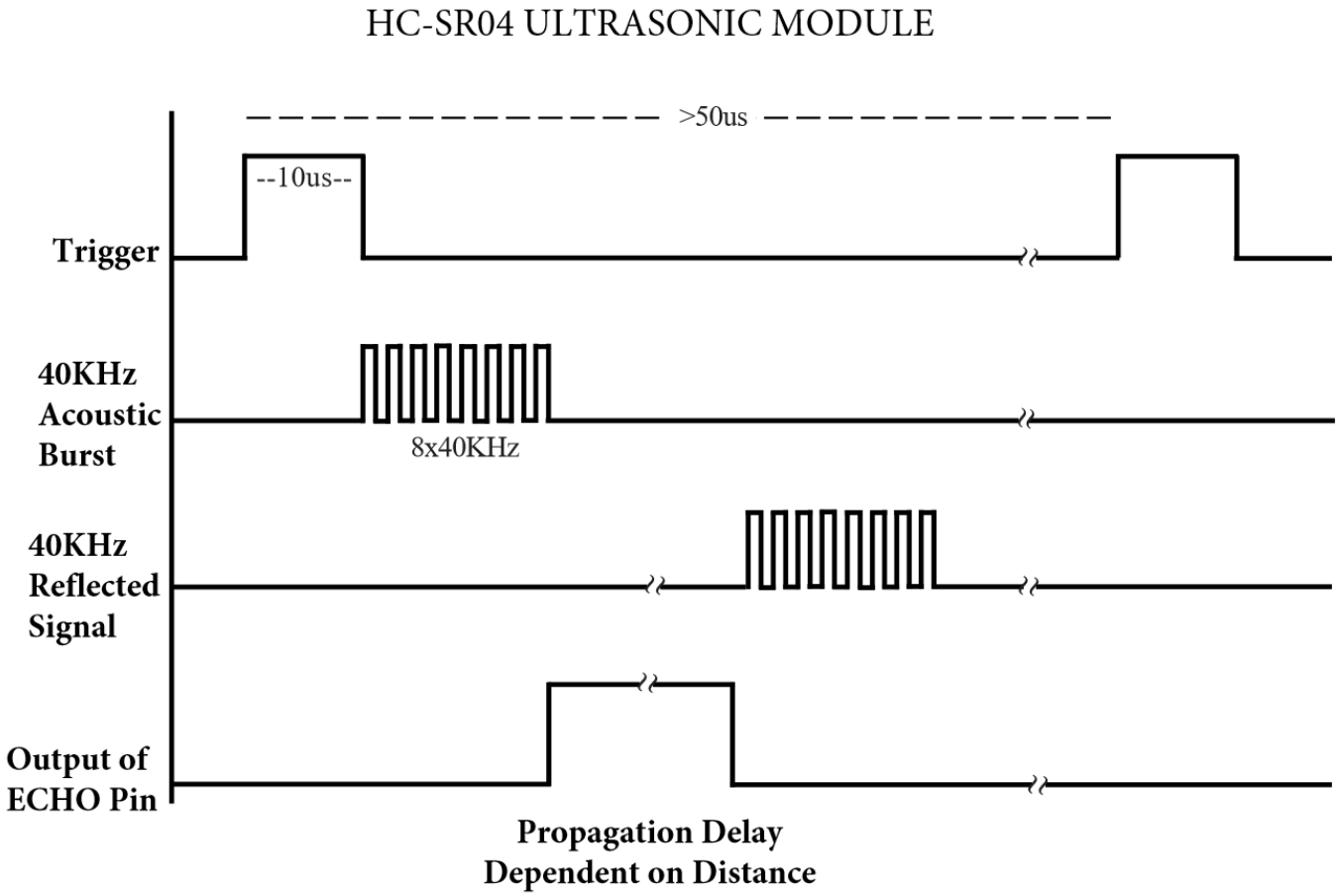
The grove ultrasonic ranger has 4 pins on the connecter (like all grove connections). The pins are labeled as follows:



1 : connected to system GND
2 : support 3.3v and 5v
3 : not connected
4 : signal for recevier and transmitter

We can see that pins 1 and 2 are connected to power and ground. Pin 3 is not connected, so it is unused; leaving only pin 4 left to control the sensor. The ultrasonic sensor will emit an ultrasonic sound wave when it receives a 10 microsecond pulse on pin 4. Then, the ultrasonic sensor will raise pin 4 high and will drive it high until the sound pulse returns. The duration of the high sound pulse

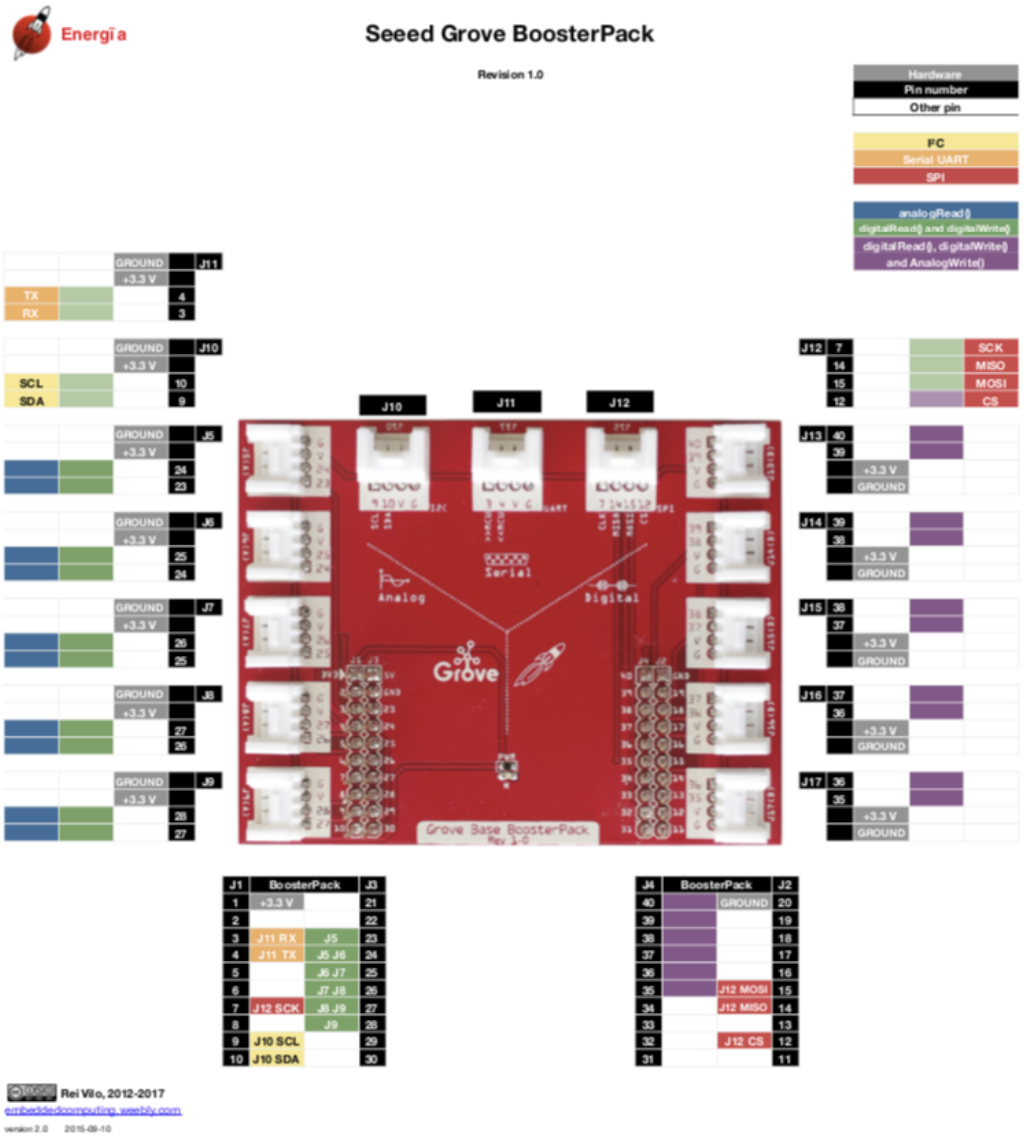
corresponds to the amount of time it took the ultrasonic sound to travel to the object and back to the receiver. This is illustrated in the timing diagram below:



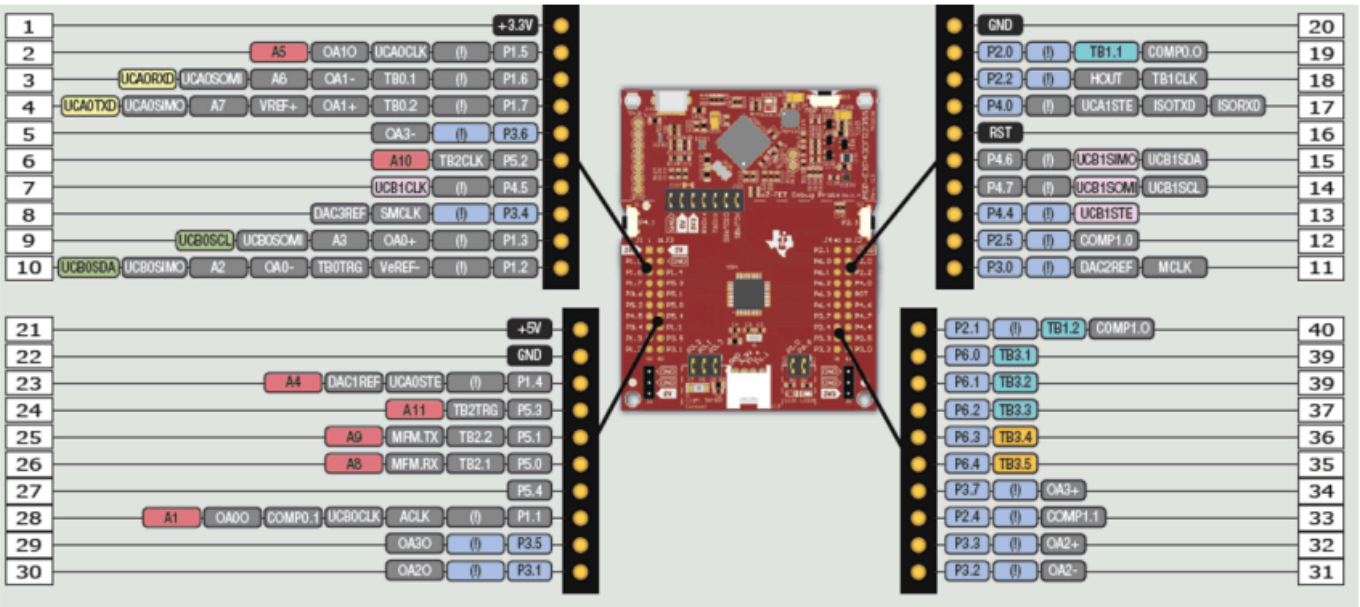
If we measure the response high time on the "Output of ECHO Pin" we would find the total flight time of the ultrasonic wave. We need to divide this value by two and multiply it by the speed of sound to get the distance. We have to divide by two, because the time accounts for travel TO and FROM the object, and we want just the distance one way!

Step 2: Connecting Components

For this lab, we are using two grove devices, which means it's time to use the grove base boosterpack to get more connections. Choose two of the connectors on the boosterpack to connect and connect your buzzer and ultrasonic ranger. When you plug the connectors in, take note of the numbers printed on the white portion of the silk screen next to the connectors you selected. These indicate which pins on the grove booster pack they are connected to. For this lab, any of these connections will do, as they all are connected to GPIO on our MSP430 board.



For instance, if I plugged my ultrasonic onto the bottom left connector, I would note that my grove connector's yellow and white wires are connected to "36" and "35". I would then note that those two numbers appear on the inner column of the right jumper pins as well.



When I plug this board onto the top or bottom of the MSP-EXP430FR2355, pins 36 and 35 on the booster pack would then be connected to pins P6.3 and P6.4. Now I know which GPIO to manipulate to interact with that particular grove connector on the booster pack.

Plug in both your buzzer and ultrasonic to any two of these and note their associated GPIO pin connections.

Step 3: Driving the Buzzer

We'll start with the easy part, getting the buzzer to make a sound. The buzzer essentially works just like an LED, but is way more obnoxious. Like the Ultrasonic sensor, the buzzer has four connections, but one of them is unused. Look at the bottom of your buzzer to find out which pin is unused. The pin labeled "sig" is the connection we care about and writing a digital high to this pin will cause the buzzer to make sound, and a digital low will silence it.

Create a new project (https://uncc.instructure.com/courses/197484/pages/creating-a-new-ccs-project) in CCS to write this code. We will begin by creating an initialization function for our GPIO peripheral, which as of right now we'll only worry about the initializing the buzzer's pin. Set the buzzer's pin as an output in this function. After that, create a function that will allow you to turn the buzzer on or off.

Before moving on, test this code by turning the buzzer on and off in a similar fasion to our very first

blinking LED example. Ensure that you have your connections correct and are able to successfully control the buzzer.

Step 4: Driving the Ultrasonic Sensor

To drive the ultrasonic sensor, we'll need to create the following sequence on the GPIO connected to the signal pin:

set pin as output -> set pin high for 10 us -> set pin to input -> wait for pin to go high -> start timer -> wait for pin to go low -> stop timer -> record count -> compute distance

This process requires careful attention. We'll begin by creating a function to contain this entire process:

```
float getDistance()
```

Inside this function we'll do the following:

- Begin by setting the pin connected to the signal wire as an output.
- Then, set the signal wire as a digital high.
- Wait for 10 microseconds. Because this delay is so short, I would recommend simply using the __delay_cycles() function for this. How many cycles will you need to wait? Keep in mind default clock speed is 1 MHz.
- After the 10 microseconds set the pin low and then configure it as an input.
- Now, we wait until the signal goes back high.
- Once it goes high, we initialize and start the timer.
    - The idea here is we will let the timer count until the signal goes back low
    - Once it's low, we'll read the count value stored in the timer to determine the ultrasonic flight time
    - We're operating on a microsecond scale here, so we need to select a fast clock source for the timer
        - If the clock is too slow, we won't have the resolution to compute the range accurately
        - The default clock speed for the SMCLK is 1 MHz, which is one of the clock options for our timer
    - We also will need to select a mode for the timer
        - Since we're not counting up to a specific value or generating an interrupt, the continuous mode makes the most sense
    - TLDR: Continuous mode, SMCLK as clock source, and no interrupts
- Once the signal goes back low, we stop the timer
    - Change the mode from continuous to stop.
- Record the count value of the timer
    - There is a specific register that holds the count value

- Compute the distance
  - Convert the count value to seconds (use the 1 MHz clock source to get this value)
  - Multiply the time in seconds by the speed of sound (~343 meters per second)
  - Divide the value by 2 to get the distance one way
  - Return the value in meters (or convert it to another unit if you want)

Hint: To understand how to control the timer, look for these settings in the **MSP430FR2XX user's guide (https://uncc.instructure.com/courses/197484/files/20876987?wrap=1)** **(https://uncc.instructure.com/courses/197484/files/20876987/download?download_frd=1)** . Section 14.3 on page 408 contains is where the register descriptions begin. For this assignment, you will only need to interact with 2 of the timer's registers (a control register and the count register).

Hint: Remember the relationship between clock frequency, count values, and delay time: *counts* = *delay / clock period*

Set up a test scenario to verify this function works before moving on. This stuff can get complicated, so be sure to use the debugger to inspect variables and register contents if you aren't getting the results you expect!

Step 5: Creating the Tripwire Logic

Now it's time to set up our tripwire. We'll use a button press to "arm" the trip wire. Once armed, the ultrasonic will continuously take measurements. If it reads a measurement that is significantly different from what it read once armed, it will sound the alarm by turning the buzzer on. Pressing the arm button again will reset it and repeat the process.

If we arm the ultrasonic tripwire while it's facing a wall, we'll expect to keep reading a similar distance. Remember, sensors are subject to white noise which will result in a slightly different reading each time, even if the sensor didn't move at all. You'll need to incorporate some kind of tolerance value to keep your tripwire from being too sensitive.

You can designate either or both of the buttons to act as the "arming" button for this project.

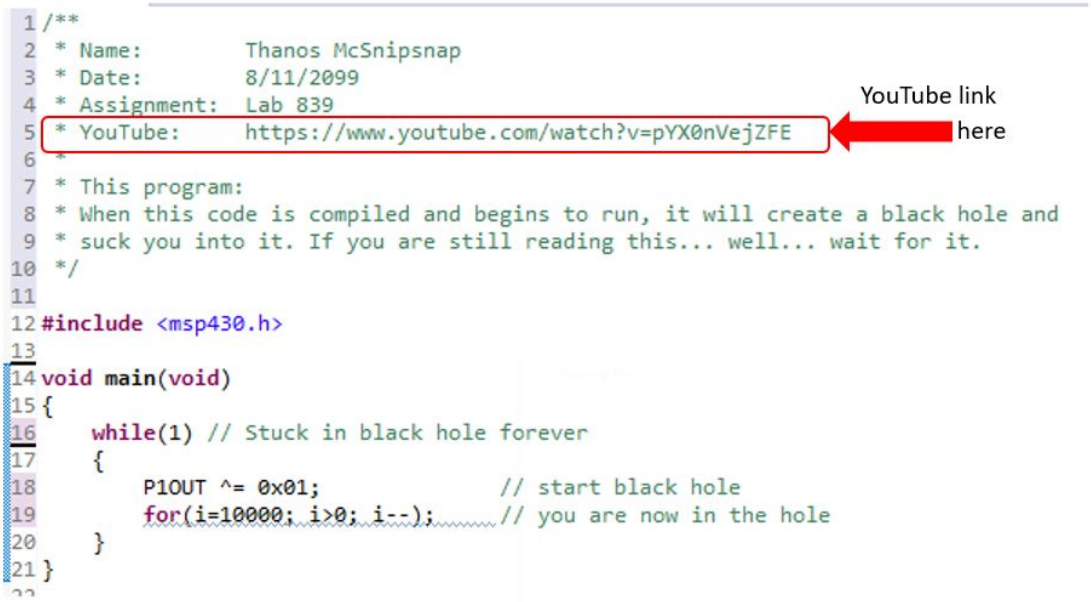
Hint: while this logic seems simple, coding it may prove more challenging than expected. I would recommend using a state machine to manage this. For example, you might have an initialization state, armed state, and alarm state.

How to Submit:

Take a video of your code working on your board. In your video, be sure to clearly demonstrate and explain all requirements `(which are listed in the grading rubric below)` of your working code.

Upload your video to **YouTube only** (do not use google drive or other platforms) and then provide a link within your code submission as a comment like the picture shows below. Remember to include

your name, program details, etc., within the program header at the top. It is common coding practice to provide a program header, so make sure you do this. If you used an example template, remember to delete the previous program header and replace it with your own.




```
/**
 * Name:         Thanos McSnipsnap
 * Date:         8/11/2099
 * Assignment:   Lab 839
 * YouTube:      https://www.youtube.com/watch?v=pYX0nVejZFE
 *
 * This program:
 * When this code is compiled and begins to run, it will create a black hole and
 * suck you into it. If you are still reading this... well... wait for it.
 */

#include <msp430.h>

void main(void)
{
    while(1) // Stuck in black hole forever
    {
        P1OUT ^= 0x01;              // start black hole
        for(i=10000; i>0; i--);     // you are now in the hole
    }
}
```

Copy your lab code from CCS into a `text file (.txt)` using notepad, Word, etc. Save your text file as `lastname_firstname_lab#.txt` with `#` being the current lab you are submitting.

Submit this text file (.txt) to Canvas.

No other submission files will be accepted. You will only have two attempts to submit during the submission deadline. If a second attempt is needed, make sure all the necessary changes are made before submitting, as this would be your final attempt.

Some Rubric

Criteria	Ratings		Pts
Code is neatly formatted	**5 pts** **Full Marks**	**0 pts** **No Marks**	5 pts
Buzzer function is created and works	**20 pts** **Full Marks**	**0 pts** **No Marks**	20 pts
Ultrasonic range function is created and works	**50 pts** **Full Marks**	**0 pts** **No Marks**	50 pts
Tripwire behavior works as described in Step 5 of the Procedure	**25 pts** **Full Marks**	**0 pts** **No Marks**	25 pts
			Total Points: 100