Penn Engineering

LAB 3: THEREMIN SUBMISSION

ESE5190 Smart Devices

1 Instructions

In this document, you'll fill out your responses to the questions listed in the Lab 3 manual in Canvas.

Student Name: vedant kelkar

Pennkey: vkelkar

Github Repository: https://github.com/ESE5190-UPenn-Fall2023/ese5190-lab-3-vedantk91

2 Part (B) - Waveform Generation

1. Timer Overflow

1. What frequency is being generated here? Is it what you expected? Show your work.

122Hz

Yes this is the expected frequency. The formula and calculation go as follows:-

16MHz/256/2/256 =122.07Hz.

This is because the maximum count of timer0 is 256 and I have used 256 as the prescalar.

2. Normal Mode

2. Did you have to prescale the system clock and/or timer clock? If so, by how much?

Yes I used a prescalar of 256. I scaled the timer clock and not the system clock.

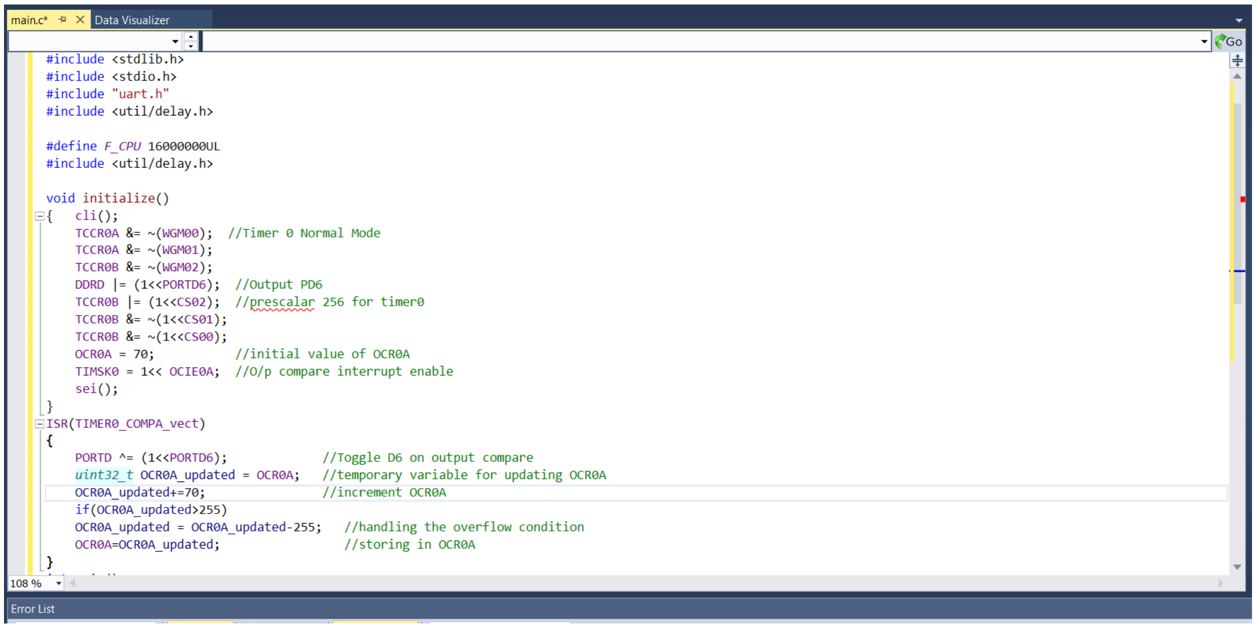
3. What number should OCR0A be in this case?

The OCRA should be 70 at the start.

Formula :- 16000000/256/440/2 -1 =70

The TCNT isn't clearing as it is not a CTC mode. So we need to keep incrementing the OCRA by 71 every time TCNT reaches OCRA. But the limit of Timer0 is 255. So as the OCRA becomes 280 It exceeds 255. So we set the OCRA as 280-255. This manages the overflow condition.

4. **Attach a screenshot of your code snippet or copy and paste the snippet into a box in the submission document. It should be quite short.**



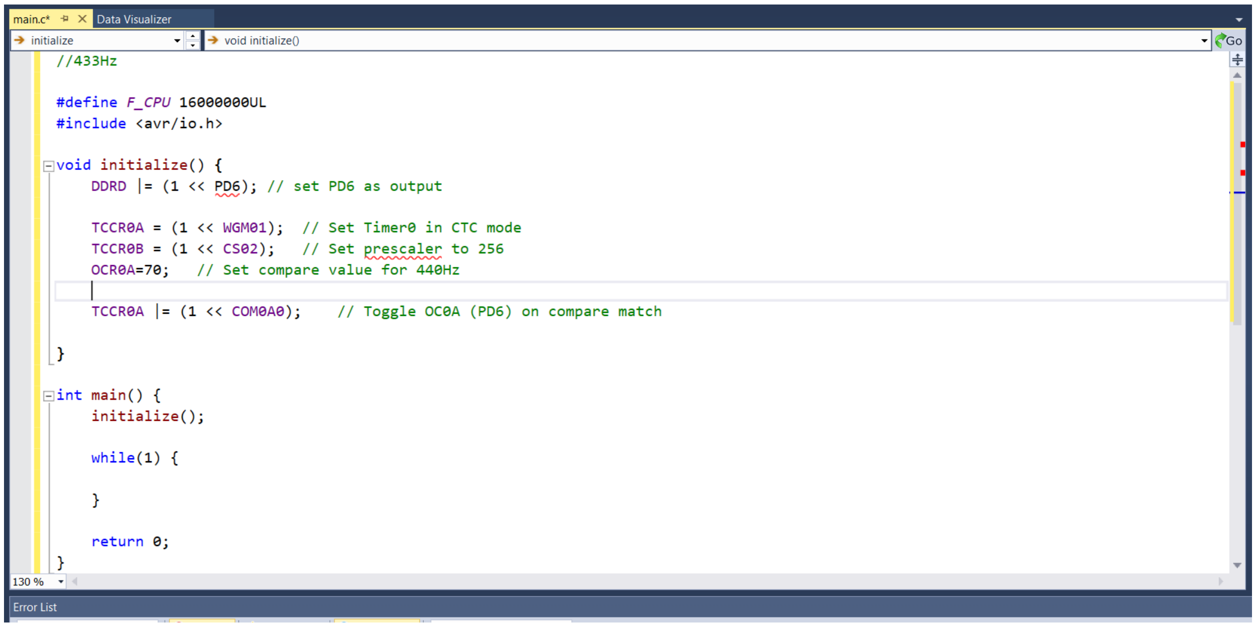
```
#include <stdlib.h>
#include <stdio.h>
#include "uart.h"
#include <util/delay.h>

#define F_CPU 16000000UL
#include <util/delay.h>

void initialize()
{   cli();
    TCCR0A &= ~(WGM00);  //Timer 0 Normal Mode
    TCCR0A &= ~(WGM01);
    TCCR0B &= ~(WGM02);
    DDRD |= (1<<PORTD6);  //Output PD6
    TCCR0B |= (1<<CS02);  //prescalar 256 for timer0
    TCCR0B &= ~(1<<CS01);
    TCCR0B &= ~(1<<CS00);
    OCR0A = 70;           //initial value of OCR0A
    TIMSK0 = 1<< OCIE0A;  //O/p compare interrupt enable
    sei();
}
ISR(TIMER0_COMPA_vect)
{
    PORTD ^= (1<<PORTD6);            //Toggle D6 on output compare
    uint32_t OCR0A_updated = OCR0A;   //temporary variable for updating OCR0A
    OCR0A_updated+=70;               //increment OCR0A
    if(OCR0A_updated>255)
    OCR0A_updated = OCR0A_updated-255;   //handling the overflow condition
    OCR0A=OCR0A_updated;                 //storing in OCR0A
}
```

5. CTC Mode

5. What number should OCR0A be in this case?

The OCRA in this case comes out to be 70. 16000000/256/440/2 -1 =70. Same formula from 5.2. In this case we will clear the TCNT whenever it reaches 70.

6. **Attach a screenshot of your code snippet or copy and paste the snippet into a box in the submission document. It should be quite short.**



```
//433Hz

#define F_CPU 16000000UL
#include <avr/io.h>

void initialize() {
    DDRD |= (1 << PD6); // set PD6 as output

    TCCR0A = (1 << WGM01);  // Set Timer0 in CTC mode
    TCCR0B = (1 << CS02);   // Set prescaler to 256
    OCR0A=70;   // Set compare value for 440Hz

    TCCR0A |= (1 << COM0A0);    // Toggle OC0A (PD6) on compare match

}

int main() {
    initialize();

    while(1) {

    }

    return 0;
}
```

7. PWM Mode

7. What number should OCR0A be in this case? Specify which Phase Correct mode you use - namely, what is the TOP value. (Refer to Table 14-8 in the datasheet).

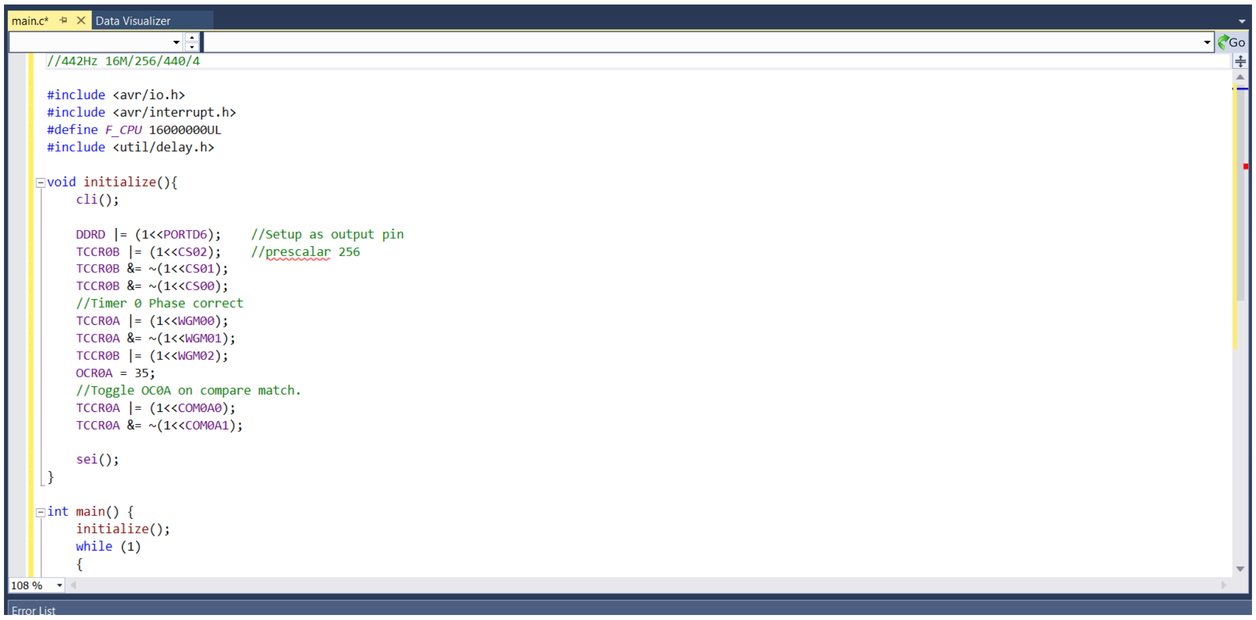
The OCR0A here has to be 35. The calculation goes as:-

16000000/256/440/4 -1 =35

Here we divide by 4 because we are using phase correct PWM. In this mode 4xOCR0A is one period. To set the phase correct mode for Timer 0 we set the following registers.

```
TCCR0A |= (1<<WGM00);
TCCR0A &= ~(1<<WGM01);
TCCR0B |= (1<<WGM02);
```

8. **Attach a screenshot of your code snippet or copy and paste the snippet into a box in the submission document. It should be quite short.**



```
//442Hz 16M/256/440/4

#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 16000000UL
#include <util/delay.h>

void initialize(){
    cli();

    DDRD |= (1<<PORTD6);    //Setup as output pin
    TCCR0B |= (1<<CS02);    //prescalar 256
    TCCR0B &= ~(1<<CS01);
    TCCR0B &= ~(1<<CS00);
    //Timer 0 Phase correct
    TCCR0A |= (1<<WGM00);
    TCCR0A &= ~(1<<WGM01);
    TCCR0B |= (1<<WGM02);
    OCR0A = 35;
    //Toggle OC0A on compare match.
    TCCR0A |= (1<<COM0A0);
    TCCR0A &= ~(1<<COM0A1);

    sei();
}

int main() {
    initialize();
    while (1)
    {
```

3 Part © – Measuring Distance

9. **Reading through the datasheet, what is the length or duration of the pulse that needs to be supplied in order to start the ranging?**

The duration of the pulse is 10us.

10. What are the Trig and Echo pins used for?

Trig Pin: This pin serves as the trigger, responsible for initiating a pulse. When a 10µs pulse is sent to this pin, the ultrasonic module emits 8 cycles of ultrasonic sound at a frequency of 40kHz and subsequently raises its echo pin.

Echo Pin: The echo pin generates a pulse width signal. The duration for which this signal remains high corresponds to the time it took for the ultrasonic burst to travel to an object and return. By measuring the length of this pulse in microseconds or using a timer count in a microcontroller, one can determine the distance to the object.

11. What is the largest distance (in cm) that you observed printed out in the terminal?

99cm

12. What is the smallest distance (in cm) that you observed printed out in the terminal?

2cm

4 Part (D) - Generating Different Tones

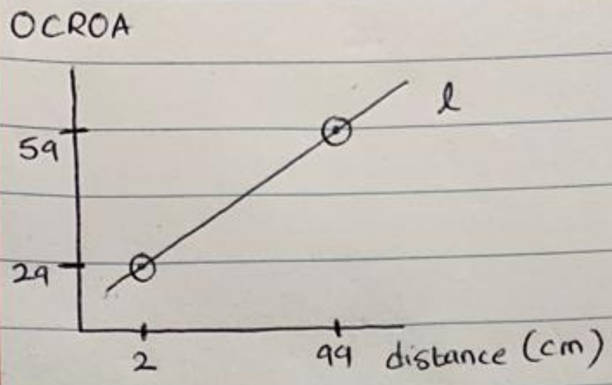
13. Fill in the table below with the correct OCR0A values that will yield the required frequency. You will have to choose a prescaler that will allow for the entire range to be generated with just one timer. Rounding errors will be expected.

OCR0A= ((clock speed)/(4xPrescalar frequency))-1 as we have used phase correct PWM in this case.

Note	**C6**	**D6**	**E6**	**F6**	**G6**	**A6**	**B6**	**C7**
Freq (Hz)	1046	1174	1318	1397	1568	1760	1975	2093
OCR0A	59	52	46	44	38	34	30	29

1. Continuous Frequency

14. Write your linear formula here. It should look something like: FREQ (or OCR0A) = SENSOR_VALUE * SOME_RATIO + SOME_NUMBER



29 is the minimum OCROA (for frequency 2093)

59 is the maximum OCROA (for Frequency 1046)

2cm is the minimum Distance

99cm is the maximum distance

We need the equation for line l

Using 2 point formula $\frac{y-y_1}{x-x_1} = \frac{y_2-y_1}{x_2-x_1}$

$$\frac{y-29}{x-2} = \frac{59-29}{99-2}$$

$$\therefore \frac{y-29}{x-2} = \frac{30}{97}$$

$$97y = 30x + 2753$$

$$\boxed{y = 0.309x \ 28.38}$$

where x is the distance

y is the OCROA

15. Take a video of the buzzer changing frequency as your hand moves back and forth in front of the ultrasonic sensor.

https://drive.google.com/file/d/1ISpUEdE1wXqXnjVyeiAi2MmBTWnjaaJ6/view?usp=sharing

2. Discrete Frequency

16. Take a video of the buzzer changing frequency as your hand moves back and forth in front of the ultrasonic sensor.

https://drive.google.com/file/d/1ISWChujajx3t1hoFR9UY59m20B-7qsiK/view?usp=sharing

5 Part (E) - Adjusting Volume

17. What is the maximum and minimum ADC values read?

The Minimum value I recorded was 3 and the maximum value was 860.

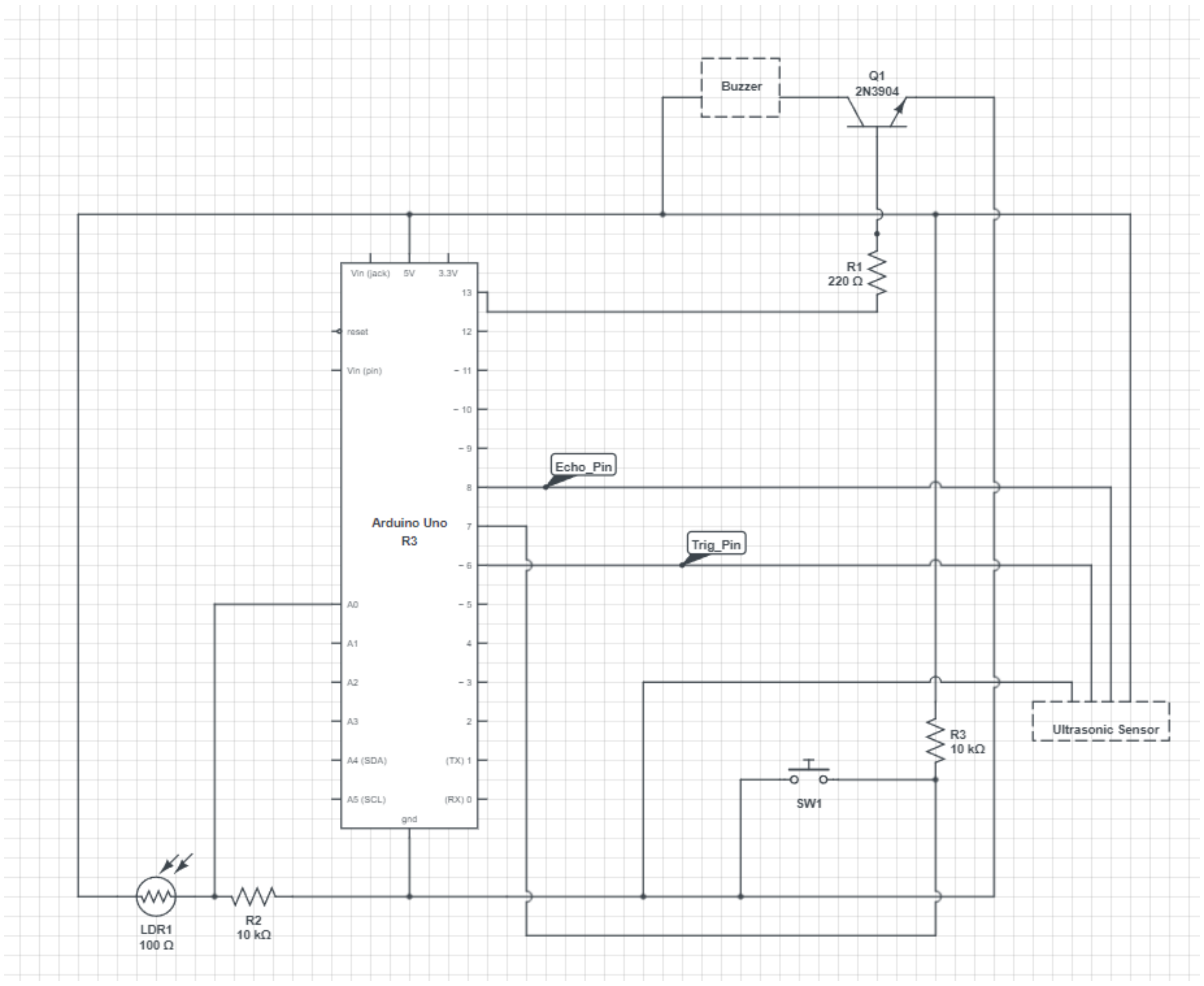
18. Fill in the table below with your ADC ranges.

ADC Ranges	Duty Cycle
3 - 86	5%
86 – 172	10%
173- 258	15%
259 – 344	20%
345 – 430	25%
431 – 516	30%
517 – 602	35%
603 – 688	40%
689 – 774	45%

775 - 860	50%

6 Part (F) - Putting it All Together

19. Draw your final circuit in Circuit Lab and attach an image.



20. Take a video of your final product working, clearly showing the four requirements above and attach the link.

https://drive.google.com/file/d/1IUAraiz5ozrrI0416gXr_Fm5j26uH-AM/view?usp=sharing

Link to **`main.c`**

https://github.com/ESE5190-UPenn-Fall2023/ese5190-lab-3-vedantk91/blob/main/main.c

7 Part (G) - Extra Credit

21. Why is there a resistor required at the base of the transistor in Figures 1?

Its purpose is to control and restrict the current flowing from the microcontroller pin to the BJT transistor's base. In the absence of this resistor, there would be a direct connection made between the base of the BJT and the microcontroller output, which might allow for the passage of excessive current. The microcontroller or the BJT may sustain harm as a result of this circumstance. By incorporating this resistor, we establish a controlled limit on the amount of current that enters the base, regardless of the voltage output of the microcontroller.

22. Why is a BJT used instead of a MOSFET in Figure 1 and can a MOSFET be used instead?

MOSFETs are voltage-driven devices, whereas BJTs are current-driven. A BJT allows a microcontroller to regulate a higher load current with a lower base current than a microcontroller GPIO pin, which can normally source or sink only a certain amount of current. This is why BJTs work well in these kinds of applications. In many situations, a MOSFET can replace a BJT if the necessary modifications are made to the circuitry taking into account the differences between the two devices. When utilizing a MOSFET, we have to make sure that the output voltage of the microcontroller is high enough to activate the MOSFET completely. In addition, a gate resistor can be utilized to shield the microcontroller pin from high-frequency oscillations.

23. What additional feature would you want to add to this "instrument"? Outline how you would implement it.

Pulsating Mode: In this mode, the buzzer emits pulsating signals, with the pulsation rate determined by the object's distance.

Harmonious Mode: This mode overlays multiple frequencies under specific conditions, creating a harmonic sound effect.

Melodic Mode: This mode plays distinct melodies or patterns based on object distance or light intensity, offering varied audio feedback.

Implementation:

Toggling : A button toggles between audio feedback modes.

Audio: Melodies, pulsations, and harmonics are generated based on proximity or light intensity.

Integration: Button presses are detected using interrupts or polling, and visual feedback through LEDs shows the active mode. We can use double presses or long presses for more functionalities.

24. Generate a 440Hz sine wave. Take a video or image of the result, include the image or link to the video, attach a diagram of the circuit, and upload the code into your repo with the name sinewave_ec.c