Part-2 design document

Overview:

My program has three different parts. The first part of the program, I create a sequencer to play some melodies that can change tempo and pitch continuously over time. The **reason** why I want to do this part is because I recently have learnt the knowledge of stacks and functions and I want to put them into practice.

In the second part, I create harmony and chords. **Reason:** I didn't successfully write the addition synthesizer of the sine wave in the first assignment. In the following study, I have a better understanding of the programming language. I am willing to generate harmony and chords by additive synthesis.

And in the last part, I create percussive sounds to build a drum machine set. **Reason:** After listening to the beautiful music brought by the drums in class, it aroused me a strong interest in making drums with the discoboard.

Percussion Instruments (things you hit)	
Instrument	Fundamental
Drums (Timpani)	90Hz - 180Hz
Bass (Kick) Drum	60Hz - 100Hz
Snare Drum	120 Hz - 250 Hz
Toms	60 Hz - 210 Hz
Cymbal - Hi-hat	3 kHz - 5 kHz
Xylophone	700 Hz - 3.5 kHz

The frequency of drum vibration I set is *100hz*, the vibration frequency of bass I choose is *80hz*, the vibration frequency of snare I choose is *240hz*, the vibration frequency of Toms I set is *200hz*, and the vibration frequency of cymbal I set is *4khz* and also I create a Harmony of drum sounds of *150hz* and *300hz*.

Implementation:

Pitch (The frequency of the node varies from 440hz (note A) to 587.33hz (note D))

As it is told to change one of the melodies, it is impossible to change the pitch directly considering the pitch of other melodies will be affected, hence a new register is set to control the frequency of the note. At first, initialize the value to #41 and it increments by one and move it to r6, and when it is incremented to value #55, it resets an original initial value, which continually loops.

Tempo (the melody interval reduced from 0.5s to 0s)

In this part, I was considering moving values to a register but without adding an additional register because there are many registers needed to be used when making harmony and drum music. I move r7 to r0 and store the r0 to the stack in the main function. R0 is decremented by 1000 from 12000 (48000*0.5). When decrementing to 0, r0 is reset an original value and store the r0 to the stack memory, and it loops continuously.

Harmony/Chord

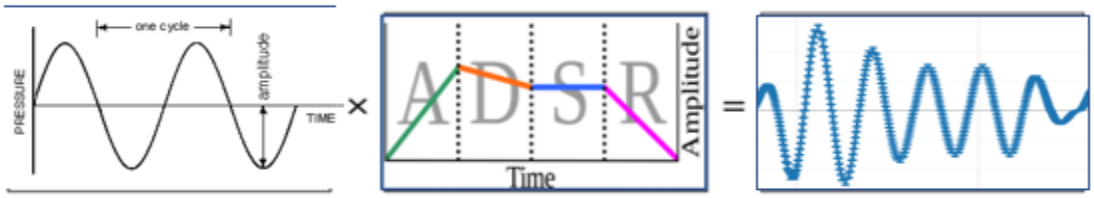
I was inspired by the first assignment. The sine waveform in the first assignment can be generated by the hexadecimal value of the sine wave table stored in the SRAM and the hexadecimal value can be loaded to the register r0 every 2 memory cells. In that way, it is only required to take the data out of the RAM and store it in r0 to make the desired sine wave of a specific frequency, but it cannot make the sine wave with generalized frequency, which is its limitation.

```
int T = 240;
int A = 16;
for(int i=0;i<=T;i++) {
    double div = 2*Math.PI*i/T;
    int result = (int) ((int) A*Math.sin(div));
    String hex = Integer.toHexString(result);
    while(hex.length()<8) {
        hex = "0"+hex;
        }
    System.out.print("0x"+ hex.substring(4,8)+",");
}
}
```

Figure 1

I utilize java programming to calculate and generate the half-word 16-bit data table of the sine wave, and then load the data into the register every two memory cells (Figure 2). T is period of sine wave and A is the amplitude of sine wave. I load the data separately from the stack and add each of them for each time before branch to **BSP_AUDIO_OUT_Play_Sample**. I search the Internet and figure out when harmonic relationships are found in the natural overtone series, the interval is called "perfect" (ie, the unison 1:1, octave 2:1, fifth 3:2, and fourth 4:3). Hence, for harmony I choose two waves 390hz and 261hz respectively. As for Chord, I chose the most common octave (C:261hz, E:329hz, G:390hz).

Build a drum machine set



The drum wave is generated by multiplying the base sine function $(A * \sin\left(\frac{2\pi}{T} * x\right)$ and the ADSR envelope function. (T is the period of the sine function) After looking up the Internet, the time ratio I set for A: D: S: R is 1:2:4:1. Firstly, I use Java function to generate a sine function with **A** amplitude, multiply with ADSR function and finally divide by amplitude **A** so that it becomes the multiplication of ASDR function and the sine wave with amplitude **1**. However, as the assembly language cannot accurately represent decimal number, the graph I generated is defective (figure 2). Therefore, I have to avoid using division or decimal number and I try another method to calculate.

Calculate for ADSR function:

The optimal case in the attack part, each cycle r8 plus 4; In the decay part, each cycle r8 minus 1; In the sustain part, each cycle r8 remains unchanged; In the release part, each cycle r8 minus 2. Then loading the data into the register r0 every two memory cells, r0 is multiplied by r8 and pass the parameter r0 to **BSP_AUDIO_OUT_Play_Sample**. In that way, if draw the graph for the number stored in the r8 register, it matches ADSR envelope.

Calculate for Base sine function:

For drum:

Base sine function: $16 * \sin\left(\frac{2\pi}{480} * x\right)$, A = 480, D=960, S=1920, R=480, amplitude = #30720 (A*16*4)

For snare:

Base sine function: $20 * \sin\left(\frac{2\pi}{200} * x\right)$, A = 400, D= 800, S=1600, R = 400, amplitude = #32000(A*20*4)

For bass:

Base sine function: $13 * \sin\left(\frac{2\pi}{600} * x\right)$, A = 600, D=1200, S=2400, R=600, amplitude = #31200(A*13*4)

For toms:

Base sine function: $16 * \sin\left(\frac{2\pi}{240} * x\right)$, A = 480, D= 960, S=1920, R = 480, amplitude = #30720(A*16*4)

For cymbal:

Base sine function: $27 * \sin\left(\frac{2\pi}{12} * x\right)$, A = 300, D=600, S=1200, R=300, amplitude = #32400(A*27*4)

For harmony:

Base sine function: $24 * \sin\left(\frac{2\pi}{320} * x\right)$, A = 320, D= 640, S=1280, R = 320, amplitude = #30720(A*24*4)

Reflection

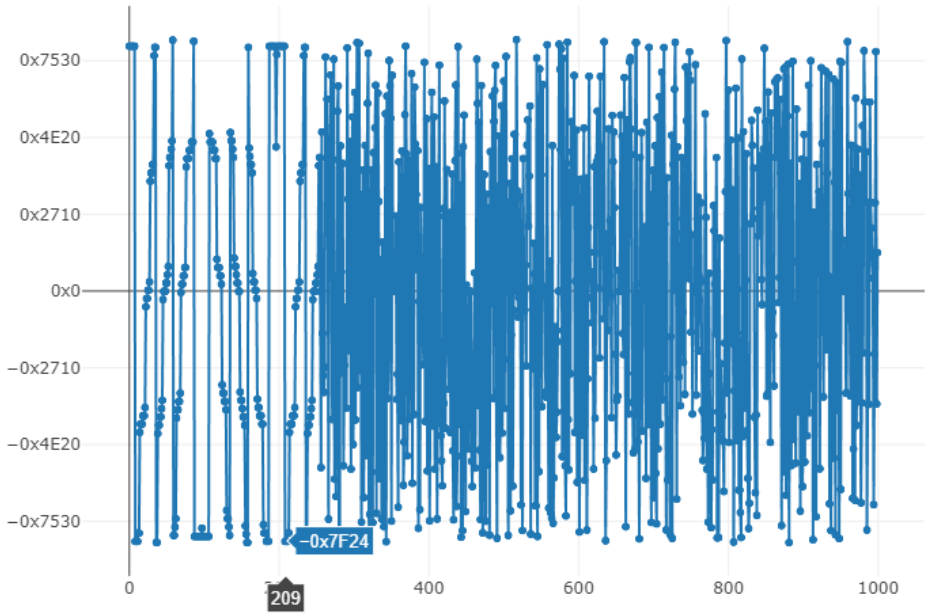
Is there anything you'd do differently if you knew more about assembly language & programming your discoboard?

I was informed that it is possible to use Taylor Series to write a sine function. The first two terms of Taylor Series are $x - \frac{x^3}{3!}$. However, the frequency of the sine wave is relatively small, and the amplitude of the sine wave is relatively large ($A * (\frac{x}{w} - \frac{\frac{x^3}{w}}{3!})$ A is the amplitude of sine function and w is the semi-periods of the function). Without using FPU stack, the sine function will be inaccurate because of all float numbers are rounded. If I can know how to use FPU stack, I can probably do sine function by using FPU stack registers instead of storing data to the register.

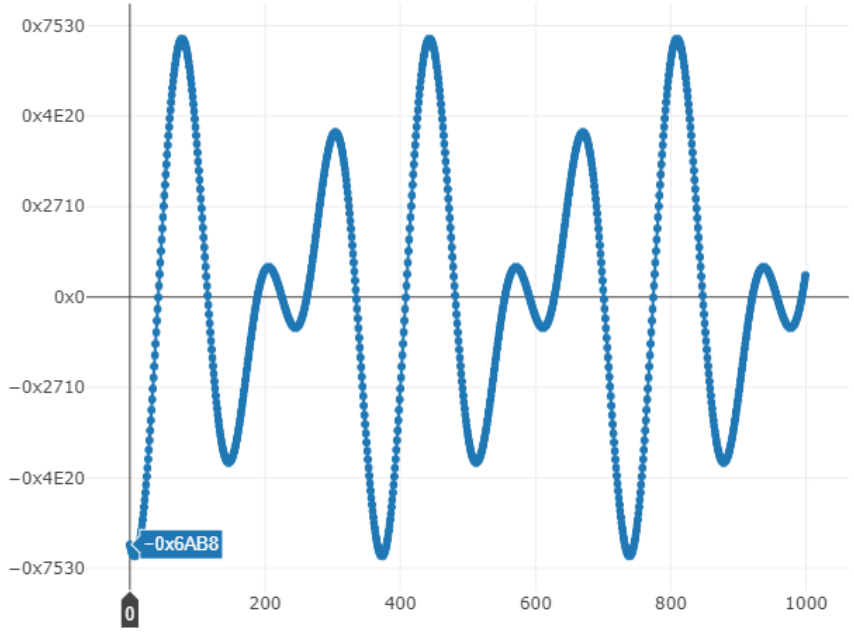
What did you learn?

In this assignment, I know how to generate percussive sounds according to ADSR rules by storing the data to the ram and loading data from memory to the register. The assignment makes me understand the course concepts better and how to debug and control the execution of the program.

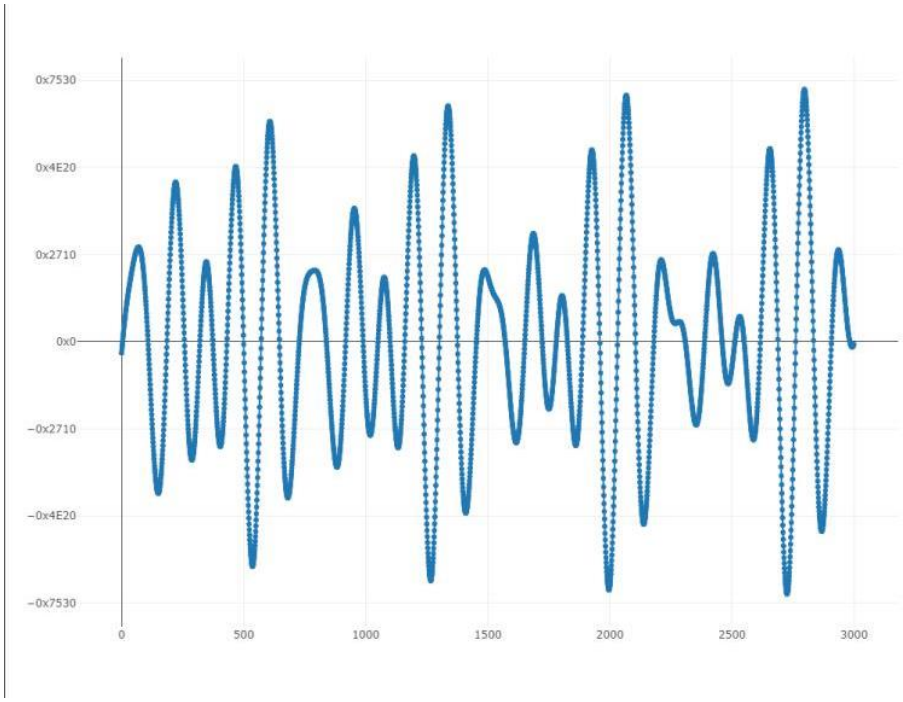
Appendix:



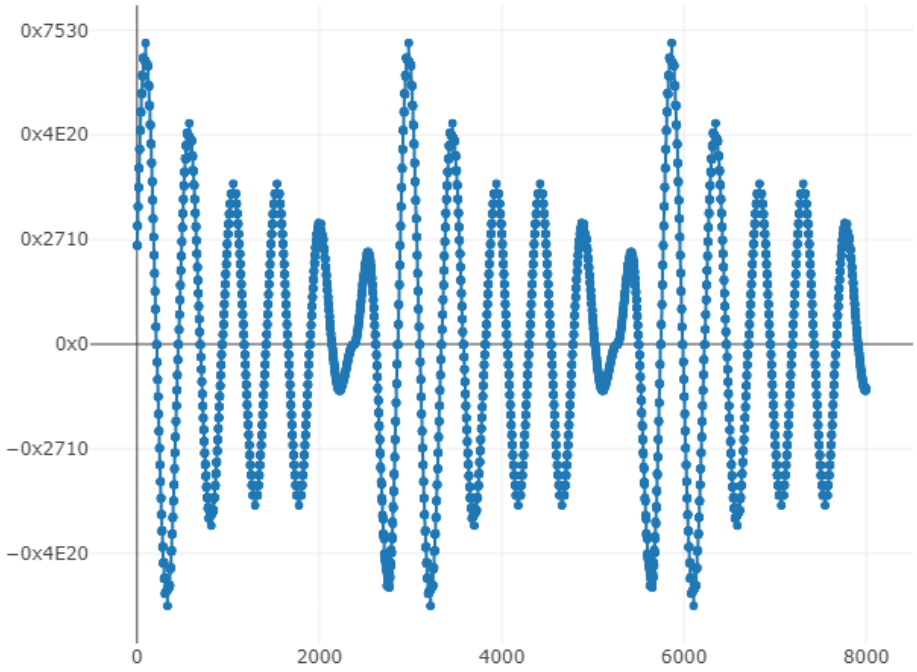
defective wave (figure 2)



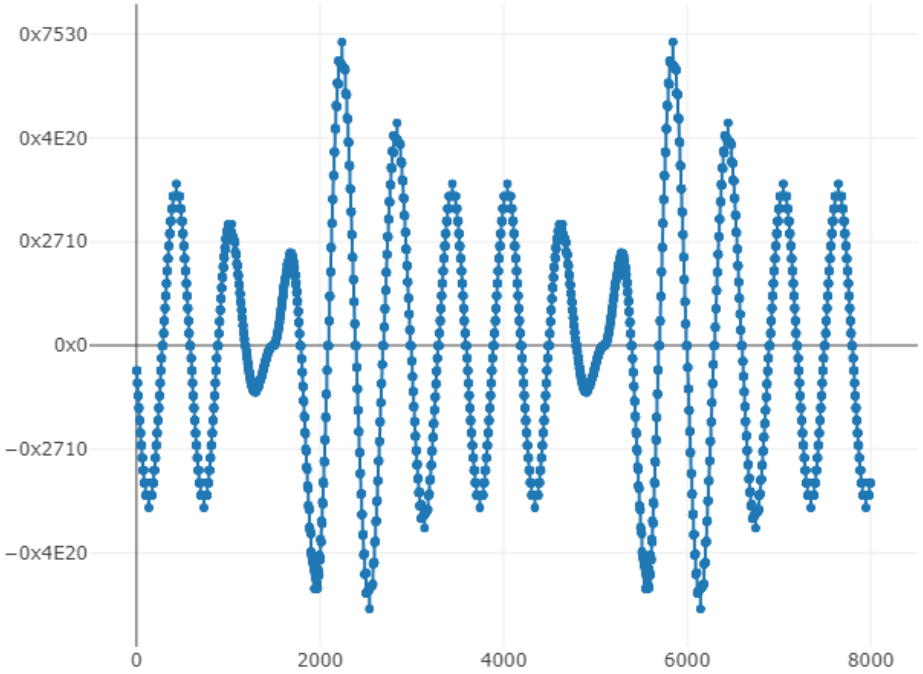
Harmony Waveform (two wave frequency ratio 2:3)



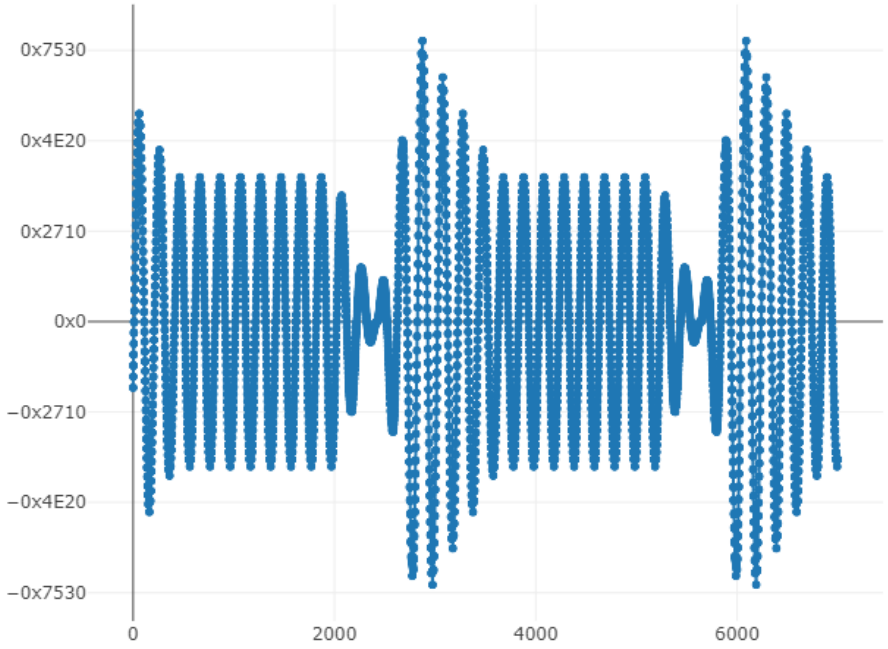
Chord Waveform (CEG)



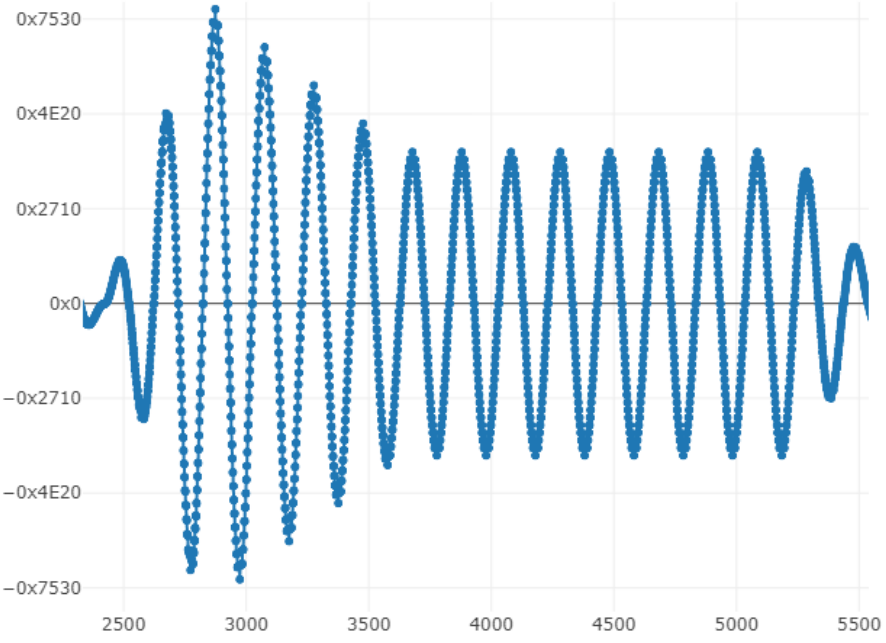
Regular drum waveform



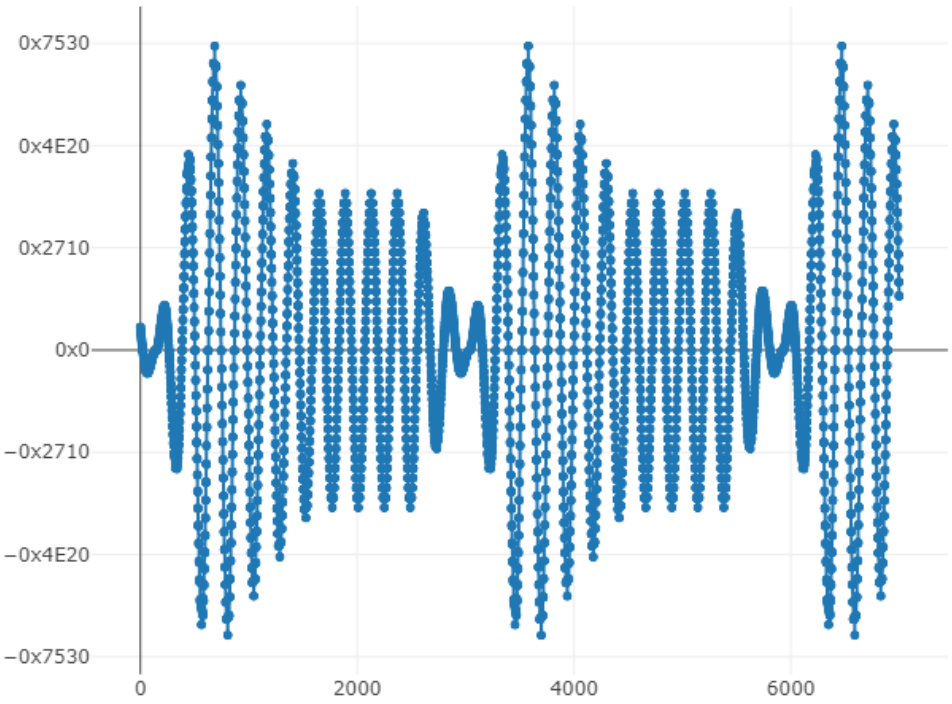
Bass waveform



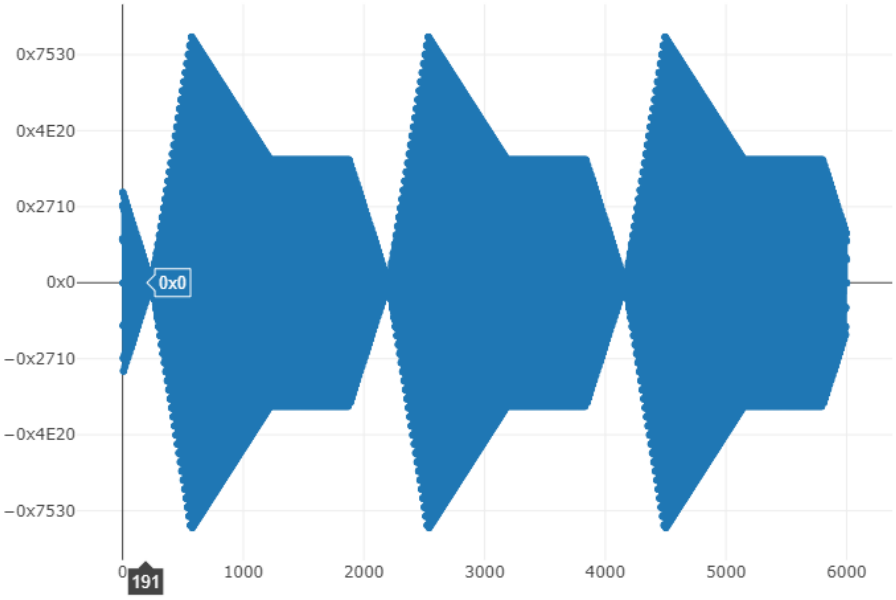
Snare waveform



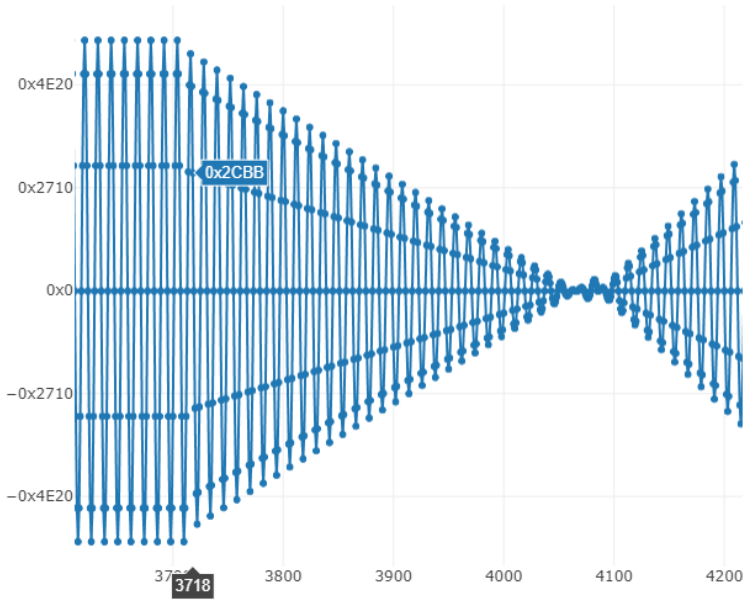
Snare waveform (Zoom)



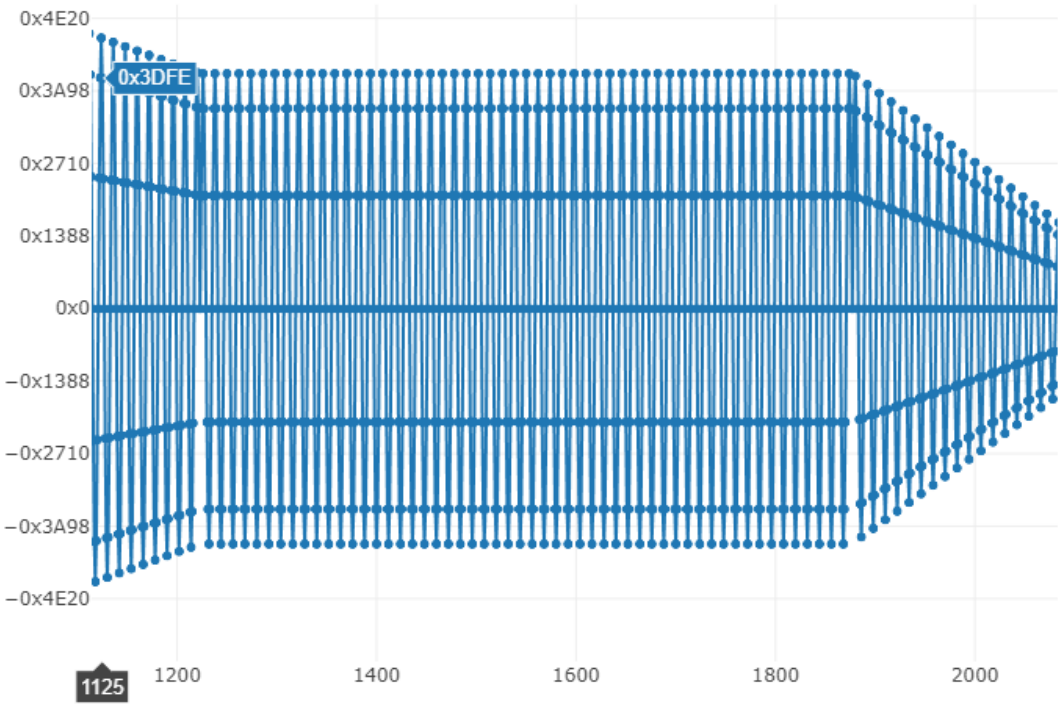
Toms waveform



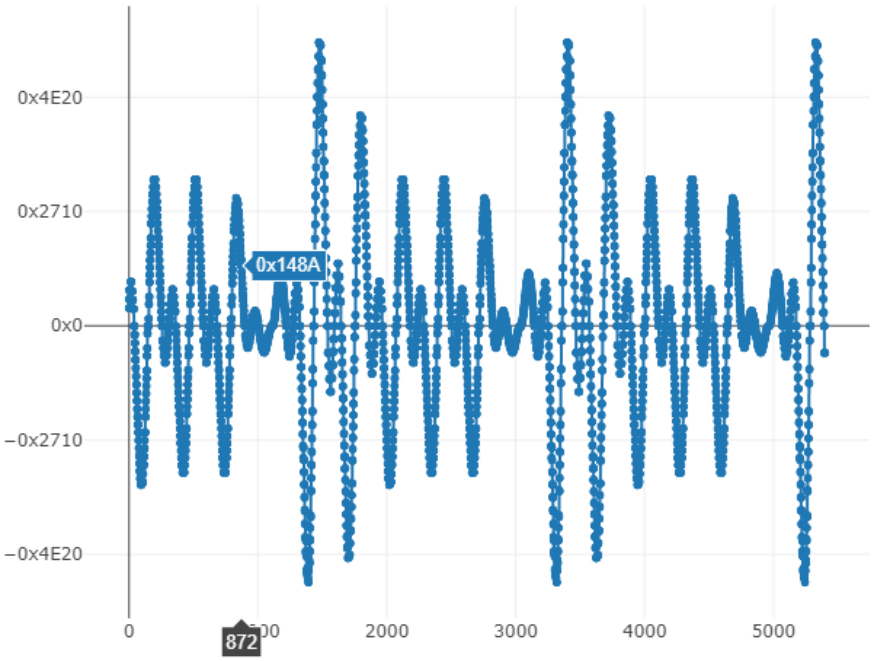
Cymbal waveform



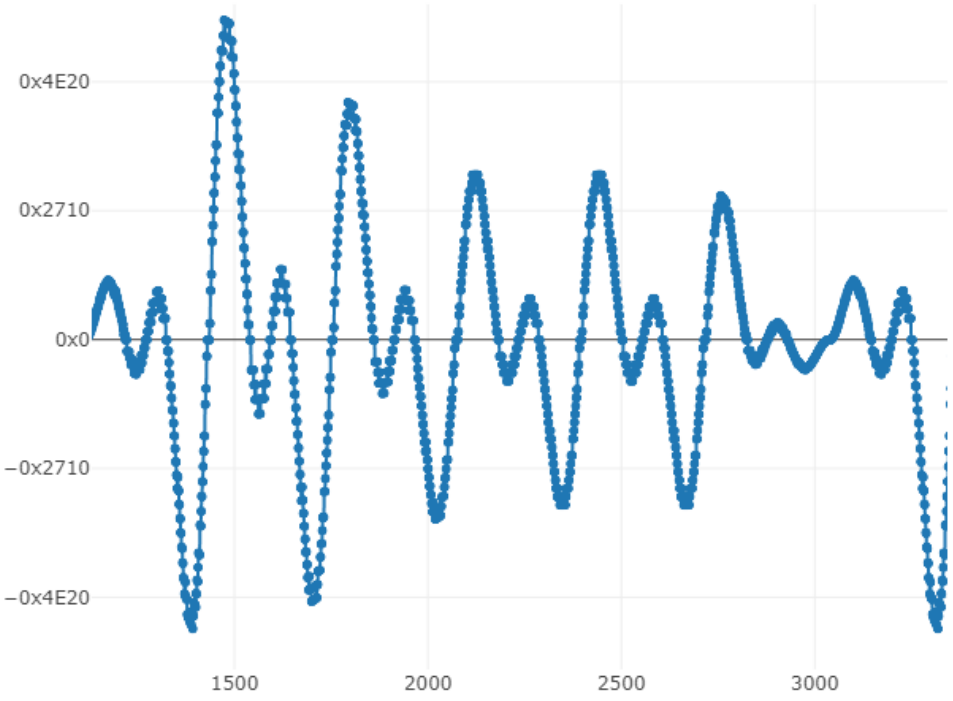
Cymbal waveform (Zoom)



Cymbal waveform (Zoom)



Harmony drum waveform (300hz + 150hz)



Harmony drum waveform (Zoom)

Xuecheng Zhang
u6284513

Due date:

11:59 PM, Friday 3rd of May 2019 (Friday of week 8)