Part-2 design document

Overview:

My program has two 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. Another part, I create percussive sounds to build a drum machine set.

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 *210hz*, and the vibration frequency of cymbal I set is *4khz*.

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

```
/*
  r4 is a counter related to r6
  r5 is a counter related to r7
  r6 is a number which stands for how many times to run in the semiperiods
  r7 is a number which controls the tempo
  r8 is a number which controls the frequency of the note fa
  r10 is the label memory address of the instruction
*/
```

Implementation:

Pitch

As it is told to change one of the melodies, it is impossible to change R6 directly so that the pitch of other melodies will be affected, hence another register should be introduced. R8 is set to control the frequency of the note Fa. At first, initialize the r8 value to #40 and R8 increments by one and move r8 to r6, and when it is incremented to value 57, it resets an original initial value, which continually loops.

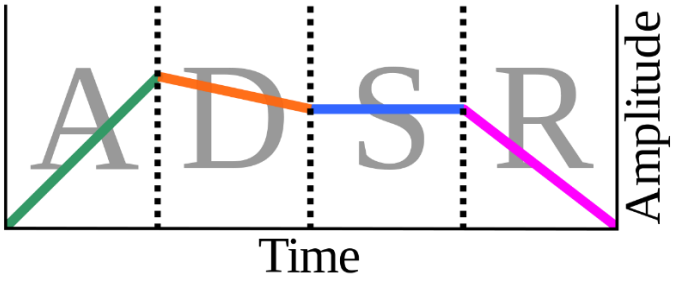
Tempo

In this part, I was considering moving values to a register but without adding an additional register. Therefore, I, at first, move r7 to r0 and store the r0 register to the stack.
R7 presents how many times of the musical notes play. If r7 is changed, then the tempo of the music is also varied. My intention is to let r0 decrement 1000 by one. When decrementing to #0, r0 is re-given back to the original initial value and store r0 back to the stack memory, and it loops continuously.

Build a drum machine set

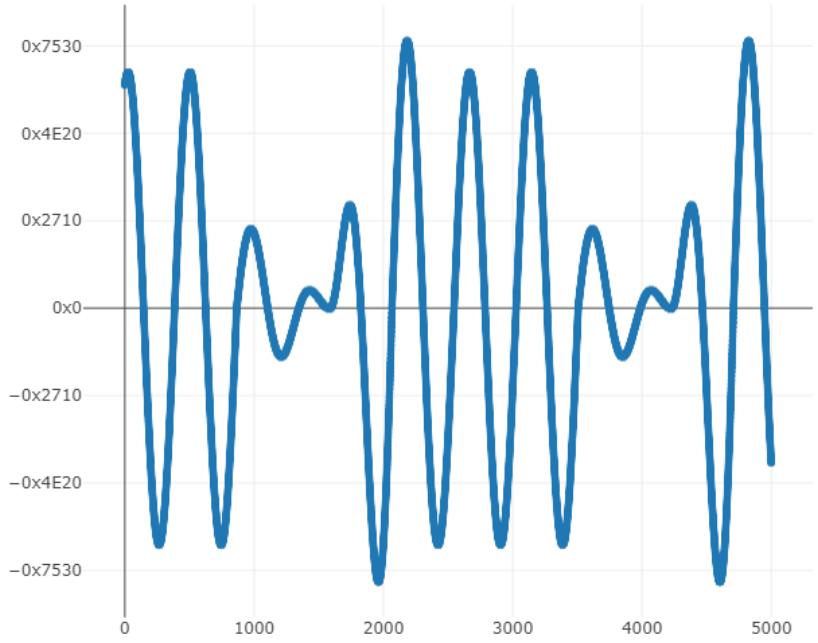
```
for(int i=0;i<N;i++) {
    float p = C*i;
    double div = Math.PI/wf;
    int result = (int) (p*Math.sin(i*div));
    String hex = Integer.toHexString(result);
    while(hex.length()<8) {
      hex = "0"+hex;
    }
    System.out.print("0x"+hex.substring(4,8)+",");
}

for(int i=N;i<M;i++) {
    float p = (-C1)*(i-C2);
    double div = Math.PI/wf;
    int result = (int)(p*Math.sin(i*div));
    String hex = Integer.toHexString(result);
    while(hex.length()<8) {
      hex = "0"+hex;
    }
    System.out.print("0x"+hex.substring(4,8)+",");
}

for(int i=M;i<O;i++) {
    float p = C;
    double div = Math.PI/wf;
    int result = (int)(p*Math.sin(i*div));
    String hex = Integer.toHexString(result);
    while(hex.length()<8) {
      hex = "0"+hex;
    }
    System.out.print("0x"+hex.substring(4,8)+",");
}

for(int i=O;i<=P;i++) {
    float p = (-C1)*(i-C2);
    double div = Math.PI/wf;
    int result = (int)(p*Math.sin(i*div));
    String hex = Integer.toHexString(result);
    while(hex.length()<8) {
      hex = "0"+hex;
    }
    System.out.print("0x"+hex.substring(4,8)+",");
}
```

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.
I utilize java programming to calculate the half-word 16-bit data table of the envelope wave, and then load the data into the register every two memory cells.
The data is the output in java programming according to the ADSR principle of the envelope wave.

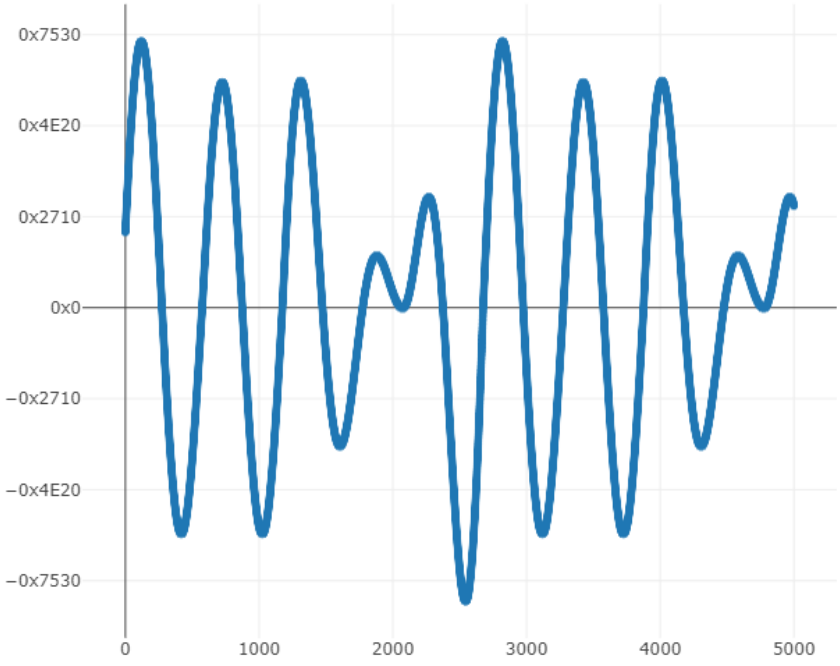


the first part is C*x*sin(pi*x/w) w is the semi-periods of sin wave
the second part is (C1-C2*x) * sin(pi*x/w) w is the semi-periods of sin wave
the third part is C*sin(pi*x/w) w is the semi-periods of sin wave
the fourth part is (C1-C2*x) * sin(pi*x/w) w is the semi-periods of sin wave

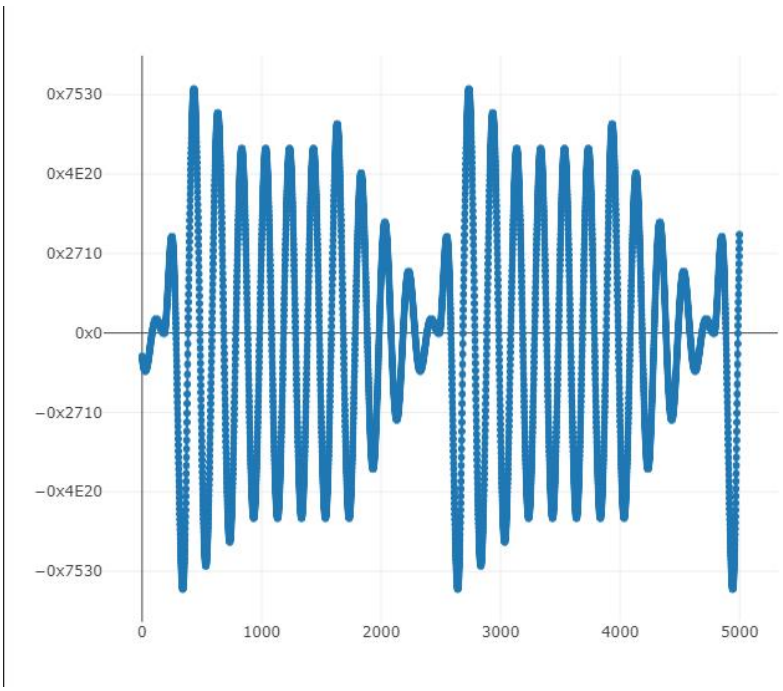
Appendix:



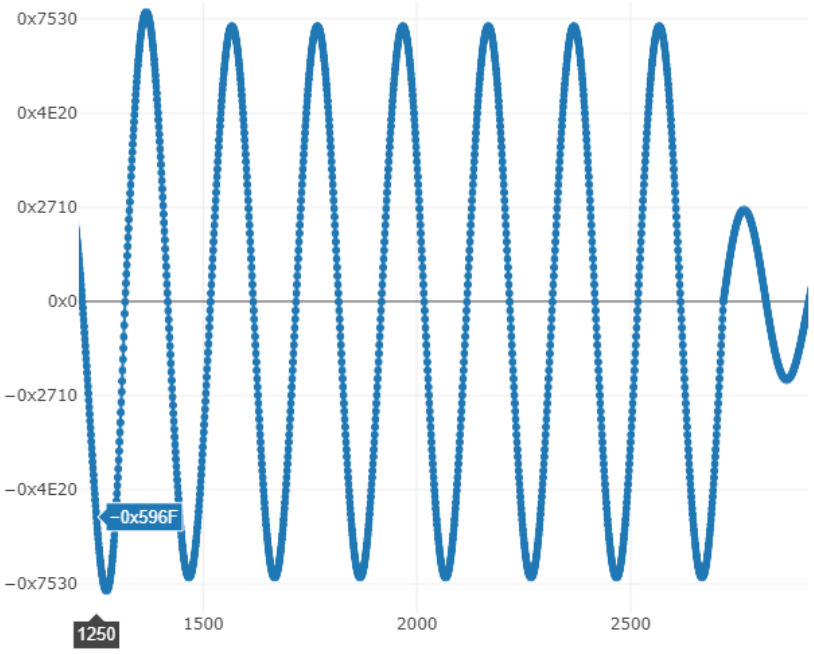
Regular drum waveform



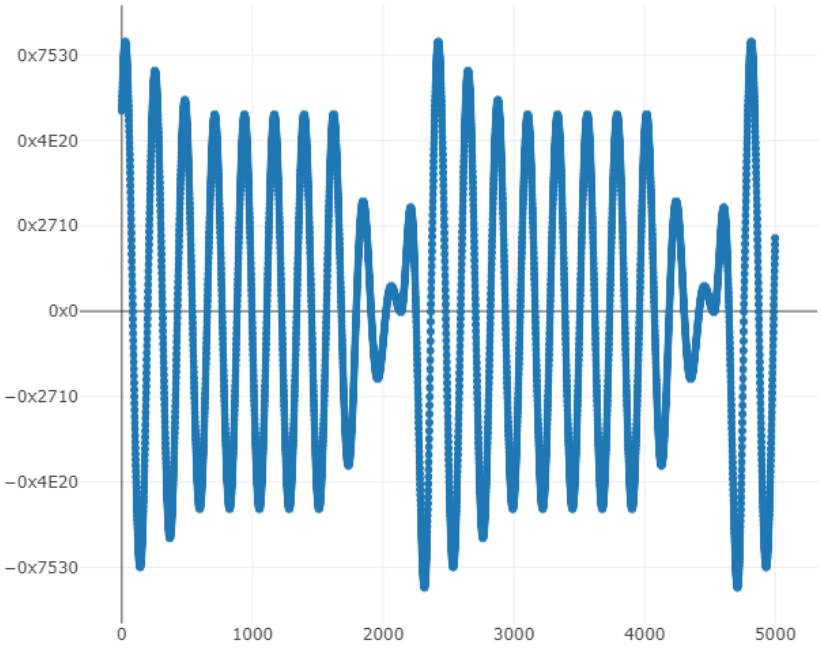
Bass waveform



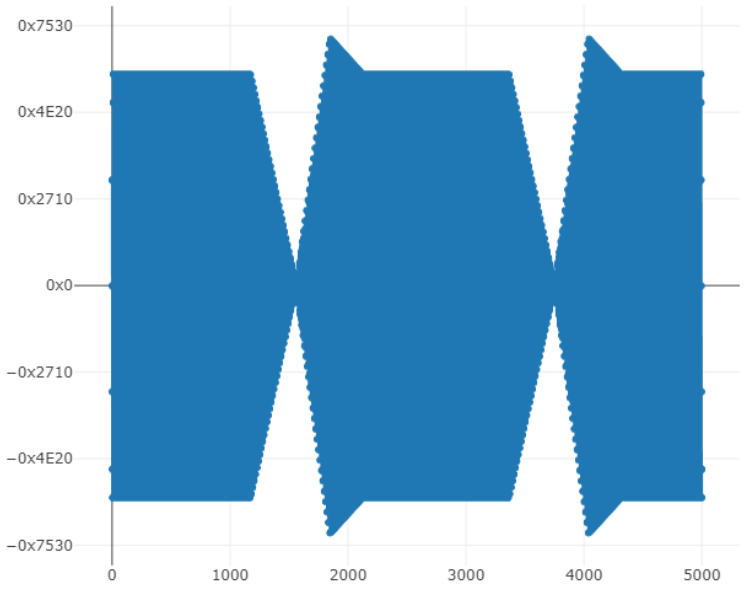
Snare waveform



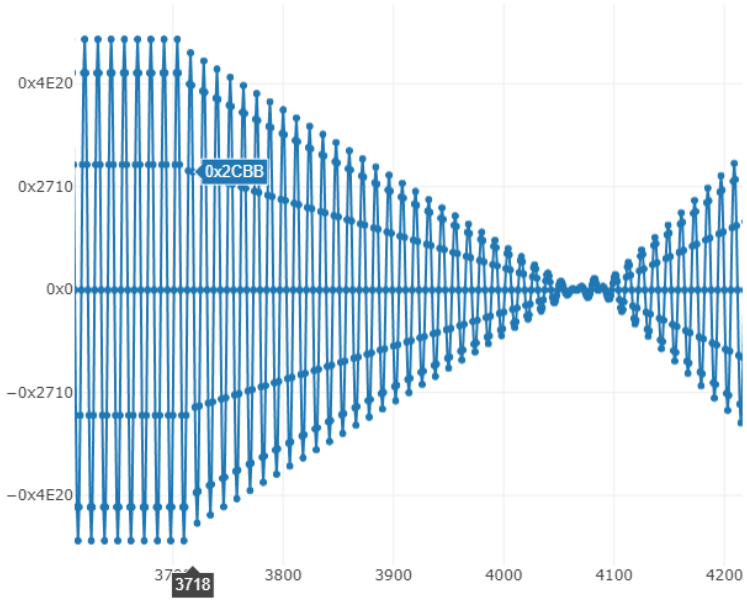
Snare waveform (Zoom)



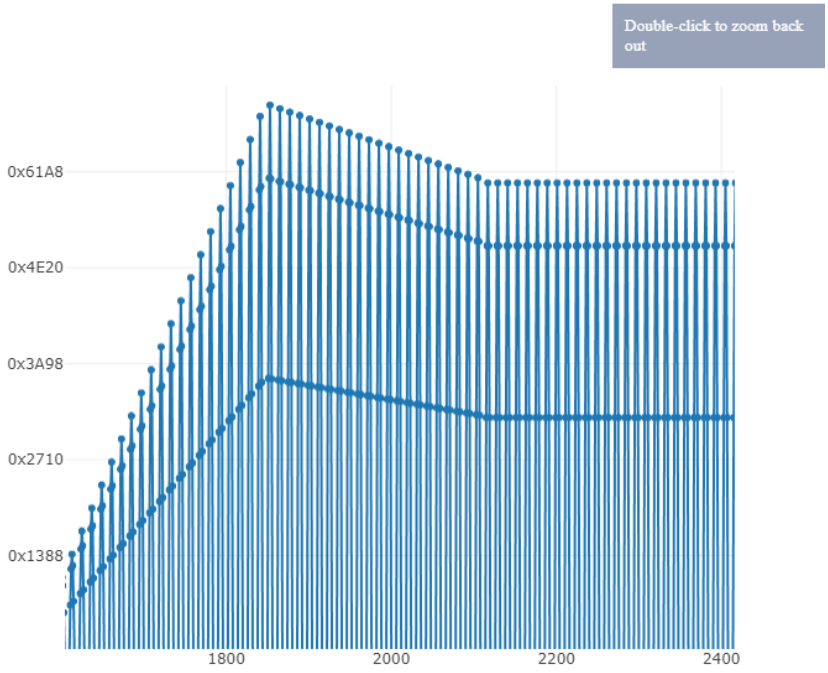
Toms waveform



Cymbal waveform



Cymbal waveform (Zoom)



Cymbal waveform (Zoom)

Xuecheng Zhang
u6284513

Due date:

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