Design Document of Assignment 1

COMP2300 – Danny Feng (u6611178)

4 Apr. 19

Overview

For this assignment of generating sound wave by assembly programme on disco board, I have successfully produced sawtooth and triangle waves with different frequency and amplitude. I also implemented FM synthesis based on those two waves. However, I tried but failed to generate a sine wave.

Implementation & Reflections

Sine wave

Sine is a trigonometric function from math. This is my first attempted after finished part 1 because I was eager to hear the most human ear friendly wave from the board. The general idea is to use trapezoidal approximate. I tried to draw a sine wave liked plot and treated it as lines that have a changing slope. In other words, it's like plotting based on the tangent. Started from 0x8000, the value in r0 adds a number in r4. And the number in r4 will add itself in the loop. R4 is just how far that r0 will increase or decrease. By doing that, I can control how fast r0 grow because it should grow faster and faster in the first stage, which is like a curve that goes up smoothly at the beginning. After a particular time of loop, the number in r4 will increase slower and change to negative, which make the curve goes up slower. Then the wave reaches the top and starts going down. Finally, the value in r4, which will always add to r0 will begin to increase again. I implemented the ideas above. But the sample plot only looks correct for the first climbing stage. Then it disappears. No sound output either. I concluded that the idea above was not enough since the r0 might be out of control and goes above or lower than limited value. I should think more about how quick r4 increase based on the sine function formula and its lookup table.

Triangle wave

Targeting 240 Hz triangle wave since it has a similar frequency to music note B3 and easy to calculate. I made it efficiently and precisely in a few lines of code with all basic arithmetic instructions just manipulating registers. There are two labels, triangle_up and triangle_down, which maps the characteristic of the triangle wave. The first mission is to calculate how many samples are there per period. Given the sample rate 48 kHz and the details of the audio output, I knew that each function call to BSP_AUDIO_OUT_Play_Sample would move the speaker physically by the amount in r0 and wait for 1/48,000 second. Thus, to calculate how many samples per period, I divided 48,000 by 240 and the result is 200 samples. Based on the characteristic of a triangle wave, 200 samples a period will be divided into 100 samples going up and 100 samples going down. I set counters that decrease one each time. When the counter equals 0, the code will branch to another label which does the opposite operation to r7, which is the value that will be put to r0. R7 started from the bottom, 0x8000 minus step value and it ought to increase to 0x7FFF in 100 steps. So, the step value is the amplitude range divided by 100 which is 655.

There was an issue in my triangle wave that the last sample of each period rounded up to the top. This was caused by the incorrect initial value, so the subtraction made it went below the minimum value. This issue is fixed in the final submission.

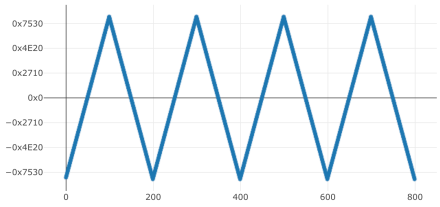

Fig.1 Triangle Wave with frequency 240Hz

Sawtooth wave

Many music uses sawtooth waves as “bass” sounds. I choose to make a 100 Hz one because it has an exciting rock feeling. A sawtooth wave is a special form of the triangle wave. The only difference is that the value reset to the lowest when it reaches the highest. In my last commit, I added a global counter to make my sawtooth wave and triangle wave play alternately.

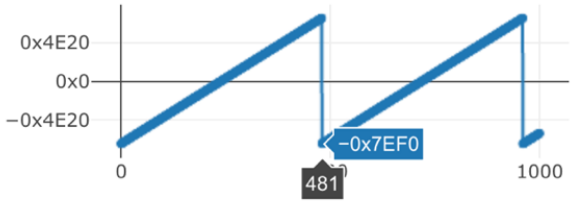

Fig.2 Sawtooth wave with frequency 100Hz

FM synthesis

Code in commit “Done FM synthesis but not perfect” (SHA: cec6047d582cec05d9ca0a9150ced4472368b4bf)

I studied additive synthesis, FM synthesis, and wavetable synthesis and decided to implement the second one because the first one requires a working sine wave. The wavetable synthesis is to sweep between waves which are not easy to implement. FM synthesis adds up waves with different types, amplitude, frequency, and produce a completely different but interesting sound. My first attempt is in the experiment branch and it’s not working. I tried to calculate the slope and intercept for each part dynamically in code. There is a regular pattern in the highest point and the lowest point so that the expression can be calculated in real time. However, I ended up with a problem, that the number grows infinitely, and the basic assembly instructions like multiply and add won’t work because the number will go above maximum integer. The purpose of calculating the expression is to plot the average point based on two or more waves. Why not just calculate the value manually? When I draw triangle wave, I add 655 for the next point in the first 100 samples and subtract 655 for the sample 100-200. For the sawtooth wave, I add 136. Therefore, in the FM synthesis, I take the average that adds (655+136)/2 = 395 for the first half and add (-655+136)/2 = -259 for the reset. When it reaches the sample 480 (which is the reset point for sawtooth wave), the whole FM synthesis also got reset to the lowest point. However, the here is not very correct because it should take the average there, not simply reset to the lowest point.

To sum up, I think I did well in this assignment and learned a lot in music, especially in electroacoustic music (That's one of my favourite music types in Spotify). I also got more familiar with assembly programming. But I should think more about sine wave and how to handle big numbers in assembly. Last but not least, I should consider using functions to make my code reusable and cleaner.

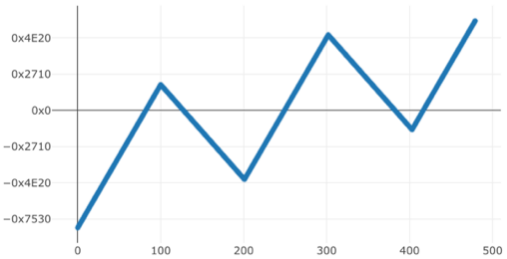

Fig.3 FM synthesis with 240Hz triangle and 100Hz sawtooth waves