Computer Organisation and Program Execution
Assignment 1 Part 2 Design Document

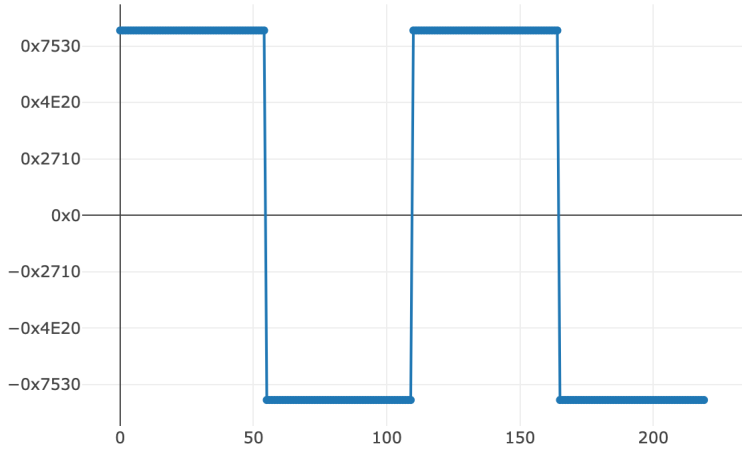
Kathia Anyosa - u6414938

9 April 2020

Overview

For part two of the assignment, I decided to modify the code in part one so as to generate a square wave as I figured that a similar approach would result in the desired waveform. However, the change in waveform from triangle to square makes a more "buzzy" sound while maintaining the same pitch. This wave has an amplitude of `0xFFFF` and a frequency of 440Hz. The algorithm is divided into 7 functions, where the first one, `main`, initialises and turns on the headphone jack, while the rest produce the square wave.

Figure 1: Sample plotter output for the generated square wave



Implementation

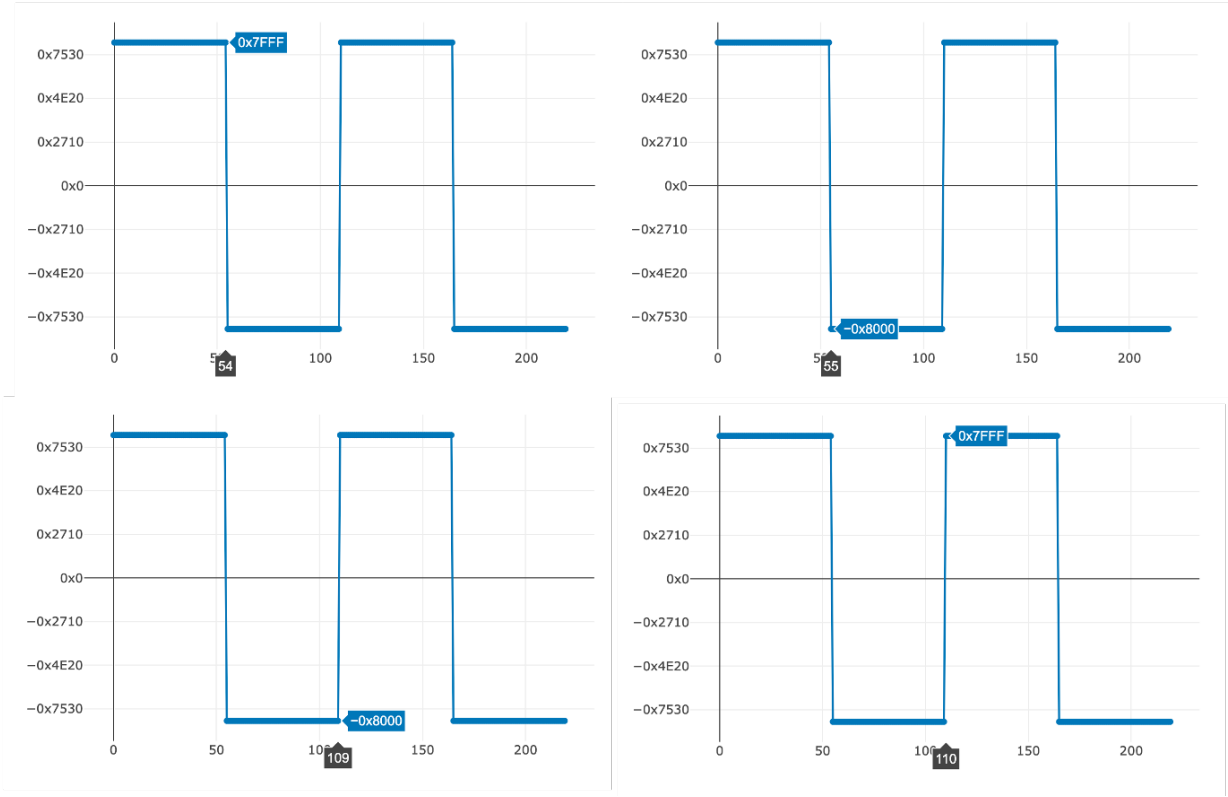
Dividing the sample rate of 48kHz by the desired frequency gave the period of the oscillation, which is

$$\frac{48kHz}{440Hz} \approx 109$$

As half the period corresponds to half an oscillation, I figured that a counter should be set so that the wave moves from its maximum value to its minimum value after 54 steps.
From this I wrote two functions: `begin_max` and `begin_min`, which set the maximum and minimum values of the square wave in `r4`, then move this value to `r0` in order to call `BSP_AUDIO_OUT_Play_Sample` to produce sound and then branch to either `loop_max` or `loop_min` so as to continuously move these values to `r0`. I chose to use `r4` to store the values as it is convention that registers `r4` to `r11` are used to hold local variables. Both loops branch to a counter, whose value is stored in `r5`, again following convention as this is a local variable. The counters are set to 54, and decrease by one every time a loop runs. Once the count reaches 0, the function branches to either `begin_max` or `begin_min`, where the counter is reset to its initial value, and the next half of the oscillation is generated.

The maximum and minimum values of `0x7fff` and `0x8000`, respectively, are the same as those in part 1, as I wanted to obtain the loudest sound possible by using the full dynamic range. This was so that I could check that the square wave had the right frequency by using an app that measures sound frequency on my phone. I also used the sample plotter to visualise the square wave, and it can be seen that the wave jumps to the next peak after 54 steps and so a period of 109 is achieved.

Figure 2: Visualising the change in peak of the oscillation



Reflection

At the start I struggled with generating a sound, but then I went through the FAQ section and used the given pseudo-code to generate a saw-tooth wave, as this only required a loop and a counter. After that I had to modify this code to generate the triangle wave, from where I was able to generate the square wave. I found it hard to calculate the values for each half of the oscillation at first, as I was initially using 54 steps for the first half and 55 for the second, so as to generate 109 different values in order to obtain a full oscillation. This, however, made the wave stop halfway through the second half and suddenly jump to the maximum value and continue down. I realized that this was possibly due to the value at `r4` increasing to the point where there would be an overflow. I fixed this by setting both counters to the same value of 54. After completing the assignment, I learnt how to use loops to continuously move values, and to change them after each iteration, as well as how to generate three different waveforms with the same frequency.

References

[1] Calling convention
`https://en.wikipedia.org/wiki/Calling_convention#ARM_(A32)`