COMP2300 Assignment 1

Part 2 - Design Document

Aditya Sharma - u6051965

I. Overview

I.1. *General Overview*

Reviewing the ideas listen in the assignment specification, the simplest waveform to implement was a different base waveform from the initial sawtooth wave. As the listed ideas are in ascending order of difficulty, implementing another base waveform may prove too simple and would not provide enough detail and difficulty. This would mean that the resulting waveform and design document would not be of substantial quality to warrant detailed documentation.

Additive synthesis, being higher on the list and therefore more difficult, may prove too cumbersome at first glance but would provide many challenges and points of conflict to write about and explain. Due to the large dichotomy in difficulty between the two chosen ideas, an effective middle point was chosen and implemented - not a particularly “additive” synthesis, but a result made up of two base waveforms repeating one after another, rather than a weighted result of both.

I.2. *Waveform Overview*

For this idea, two base waveforms were chosen to be used to be combined together to create the result. These were the triangle and the square waveform.
Throughout this document, the waveform in question will be referred to as a triangle-square waveform.

In order to move to implementation, a specification for the waveform had to be defined in terms of visual specification, amplitude specification and frequency specification. As defining the frequency specification depends on the waveforms and combinations used, this will be further discussed in section II.1. in II. Implementation.

As concluded from the general overview, the waveform must visually be a combination of both the triangle wave and the square wave, placed one after the other. Thinking of the solution in literal terms, the final waveform was ideated to be a waveform of the following visual specification:

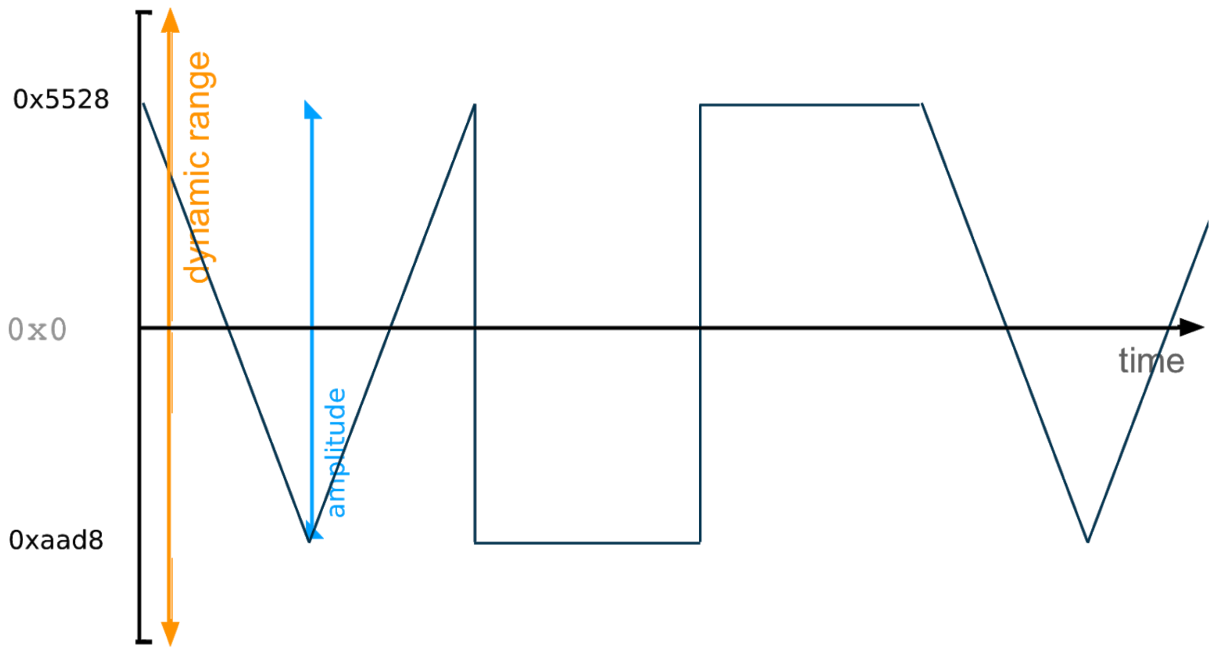


Image 1: triangle-square waveform visual spec. Source image from https://cs.anu.edu.au/courses/comp2300/assignments/01-synth/, by Ben Swift, edited by Aditya Sharma.

The waveform will repeatedly emulate the drop and rise of the triangle wave, followed by the drop and rise of the square wave. When combined, the result may look something like Image 1. Another possible rendition of this combination could be the upwards rise and fall of the triangle wave followed by the rise and fall of the square wave, however due to the way the amplitude works, this would be effectively the same.

An example is a simple triangle waveform - as the triangle is effectively repeating, the visual representation of the inverse of the wave at one point (i.e the wave in the negative amplitude at one point) will be the same as the positive representation of the wave at another point. This notion that the wave is always symmetric at one point when compared to another when repeating the same waveform means that the sound output will be the same as the speaker is outputting the same values at different points in time, as the high amplitude wave points are inverted but the low amplitude wave points are inverted also.

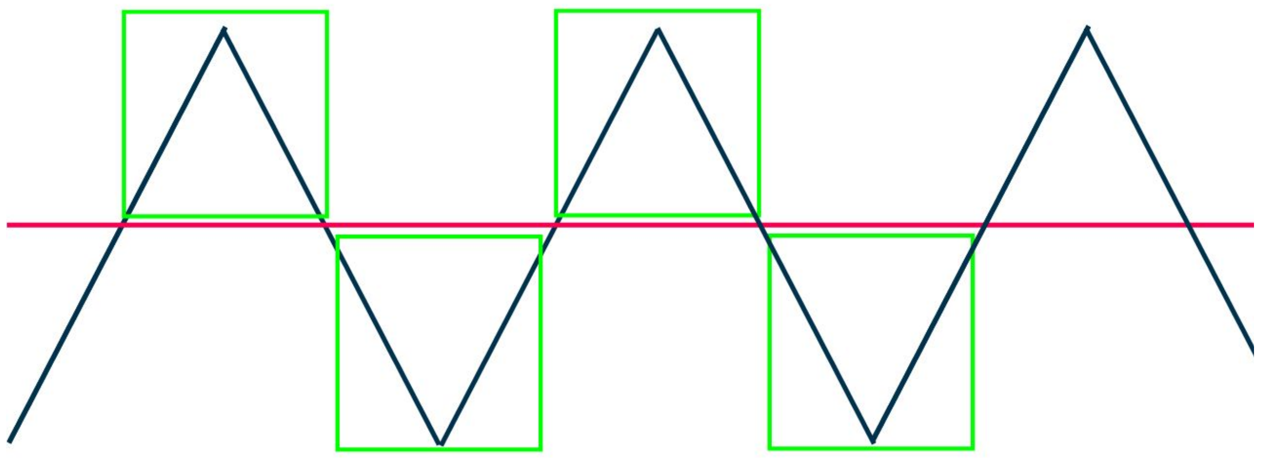


Image 2: The green squares simulate the symmetric qualities of the waveform. Image by Aditya Sharma.

The goal of this design document is to convey the reasoning for the waveform specifications chosen and to explain the implementation of the waveform.

II. Implementation

II.1. *Mathematics*

From the part 1 specifications, we were given three main variables.

48000 - The frequency of the headphone jack.
440 - Cycles per second of the CPU.
Amplitude - An arbitrary range which had to be defined to be at least ~ -16000 to 16000. ((32000 total.)

As the amplitude had to satisfy a certain condition of meeting at least half of the total possible dynamic range (-32767 to 32767), 43600 was a safe bet (-21800 to 21800). The specifics of this specific amplitude will be explained later.

Speaking in terms of a triangle wave, each ‘triangle’ needs to be defined within a specific scope in order to know how many triangles must be output by the code and in which increments the code must implement r0 before playing the audio sample for each triangle.
Given the frequency of the headphone jack, and the frequency of the CPU, we can divide the headphone frequency into the CPU to achieve the *frequency per singular waveform -* or the frequency per *triangle,* as defined prior.

48000/440 = 109

We will define this as the *ticks* or the *upper ceiling*.

For clarity, again, this result is for the amount of increments per shape of the wave, or per *triangle*.

Referring back to the amplitude choice from earlier, if we divide this by our increments, we can see how much we need to increment r0 by the achieve a lower peak to upper peak coverage.

The reason for picking 43600 is because it is higher than our minimum peak coverage and so that we can get a non-decimal value for our increments, as:

43600/109 = 400

This is how much we need to increment r0 by, for every tick in 109 ticks, from the bottom peak amplitude to the upper peak amplitude, to achieve a sawtooth wave.

Part 2 required a different implementation, however. As the sawtooth wave required 109 ticks per shape, and we have two different shapes to implement which both have different dynamics to a sawtooth wave, we must calculate the *ticks* or *increment ceiling* for both shapes of the triangle-square wave separately.

As the sawtooth wave is one continuous increment of ticks, the same increment ceiling cannot be used for square and triangle waves.

Square waves are split into two distinct sections:

- A half section of the shape where the sample is played at the lowest amplitude ceiling, and
- A half section of the shape where the sample is played at the highest amplitude ceiling.

The triangle shape will have similar sections, whereas in this wave, each section will be incrementing and decrementing, respectively.

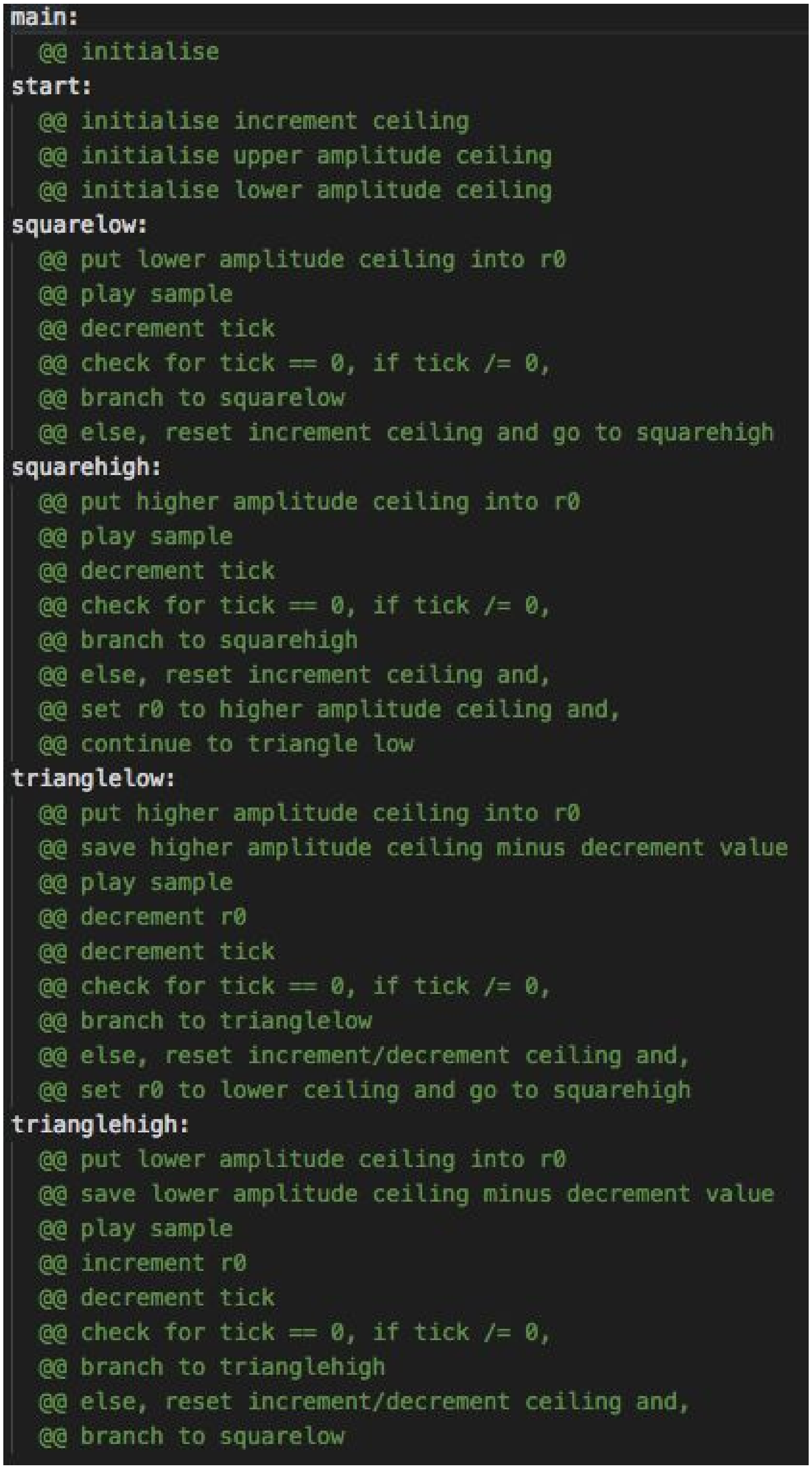
As both shapes follow this convention of modulating at one specific point, effectively splitting the shapes into two halves, we can get an *increment ceiling* for each shape by taking the floor of our sawtooth ceiling divided by 2.

(Floor) 109 / 2 = 54. This will be our increment ceiling per half.
As the triangle shape will require incrementation and decrementation, we must also calculate the values to increment/decrement by. As each side of the triangle will go from the lower amplitude ceiling to the upper, we must do similar incrementing to the sawtooth wave, but also leave room to decrement for the other half of the triangle. Using our increment ceiling per half, if we take our dynamic range (amplitude) and divide it by our increment ceiling and find the floor, we can achieve the optimal increment/decrement value to use as much of the dynamic range as possible.

(Floor) 43600 / 54 = 807. This will be our increment/decrement value per tick.

II.2. *Translation to Code and Code Structure*

Amalgamating the results from the previous sections, we can draw several conclusions for the code implementation. Before explaining this, we must break the code down initially.
As pseudocode: (view next page)



```
main:
  @@ initialise
start:
  @@ initialise increment ceiling
  @@ initialise upper amplitude ceiling
  @@ initialise lower amplitude ceiling
squarelow:
  @@ put lower amplitude ceiling into r0
  @@ play sample
  @@ decrement tick
  @@ check for tick == 0, if tick /= 0,
  @@ branch to squarelow
  @@ else, reset increment ceiling and go to squarehigh
squarehigh:
  @@ put higher amplitude ceiling into r0
  @@ play sample
  @@ decrement tick
  @@ check for tick == 0, if tick /= 0,
  @@ branch to squarehigh
  @@ else, reset increment ceiling and,
  @@ set r0 to higher amplitude ceiling and,
  @@ continue to triangle low
trianglelow:
  @@ put higher amplitude ceiling into r0
  @@ save higher amplitude ceiling minus decrement value
  @@ play sample
  @@ decrement r0
  @@ decrement tick
  @@ check for tick == 0, if tick /= 0,
  @@ branch to trianglelow
  @@ else, reset increment/decrement ceiling and,
  @@ set r0 to lower ceiling and go to squarehigh
trianglehigh:
  @@ put lower amplitude ceiling into r0
  @@ save lower amplitude ceiling minus decrement value
  @@ play sample
  @@ increment r0
  @@ decrement tick
  @@ check for tick == 0, if tick /= 0,
  @@ branch to trianglehigh
  @@ else, reset increment/decrement ceiling and,
  @@ branch to squarelow
```

III. Reflection

III.1. *Learning Outcomes*

GIven the stimulating nature of the task at hand, implementing additive synthesis in its purest definition would have proven to be difficult. Learning outcomes for part 2, given the implemented solution were:

- Learning about additive synthesis and how it is used in its weighted form and is not effective as a combination of two base waveforms as the result is only a simple distorted wave.
- The way frequencies are calculated and how they depend on the microcontroller and its architecture. For example, the headphone jack's specific frequency, the cycles per second of the CPU.
- The way sound is created and measured and its components, I.E the amplitude, the period and the effects different waveforms have on sounds - the sine wave is a smoother sounds, the sawtooth wave is rougher, etc.

III.2. *Challenges*

The conclusion for part 2 was that the final result was exactly the proposed result.
Many of the challenged along the implementation were mostly technical, as the only definitive challenge in decision was choosing which idea to implement. The technical challenged were all code based (bugs) and were resolved through debugging and the use of an oscilloscope, they were:

- A bug where the increment value was incorrectly calculated, resulting in a waveform of a different visual representation than expected.
- A bug where the increment ceiling was incorrectly split amongst sections, resulting in a waveform of incorrect frequency.
- A bug where the control flow was incorrectly formatted, and the code would branch to unintended labels.

III.3. *Future Endeavours in ARM Sound Design*

In future endeavours, sound design in ARM seemingly has a common theme which needs to be followed - the under-the-hood mathematics need to be well calculated and implemented. On the surface, this may seem as a general fundamental concept, but it applies even further to sound design as there are many mathematical factors that go into not just implementation, but visual representation and planning as well.