COMP2300 ASSIGNMENT 1

Design Document

Rachel Schroder, u6301105

Submitted 9/2/2020

Introduction

Additive Synthesis

For this assignment, I chose to generate a sawtooth wave using additive synthesis. I chose to generate this waveform for two main reasons. Firstly, I wanted to explore how to approximate a sine wave in discrete time using assembly. Secondly, I wanted to add more complexity to just generating one sine wave, and additive synthesis seemed like a good way to do this, as well as helping me practice and learn more about loop structures and functions in assembly.

Additive synthesis adds harm[illegible]ogether to add timbre to the sound. A harmonic is a wave with a frequency that is an integer m[illegible]ve's frequency (referred to as the fundamental frequency) (Sievers, 2006). This can be do[illegible]m, yet is often discussed in terms of adding sine waves. (Wikipedia, 2020)

By changing which harmonics [illegible]heir amplitudes are scaled, many different waves can be generated using this techniqu[illegible]ooth wave, all integer harmonics of a sine wave are summed, with the amplitude of each harmonic scaled by the inverse of the harmonic number. This relationship is shown for a fundamental frequency of 100Hz in the Appendix, Figure 1. The sum of an infinite number of sine waves scaled in this fashion converges to a sawtooth wave with the same frequency as the fundamental. (Sievers, 2006)

My wave

The waveform that I generated is shown in the Appendix, Figure 2. Below is a summary of its characteristics.

Table 1: Summary of wave characteristics

Characteristic	*Value*
Amplitude	0x4578 (17784)
Frequency	133.33 (48000/360)
Number of harmonics summed	17

The wave's amplitude is equal to the sum of the amplitudes of each harmonic. Each harmonic's amplitude is equal to the amplitude of the base wave (10080) divided by the harmonic number. I chose to use 10080 as the amplitude of the base wave because it is evenly divisible by all integer values up to 10. This reduces the inaccuracy caused by rounding when using integer division to calculate the amplitude of the harmonics.

The frequency of my wave was not chosen, but rather was a result of how I generated each sine wave, as there are 360 samples per cycle.

I decided to sum 17 harmonics because this was the highest number that generated a clean wave. When summing more harmonics, the time taken to perform the calculations exceeded that allowed for by the 48kHz sample rate, and thus changed the frequency and sound of the wave. However, summing a higher number of harmonics generated a much clearer sawtooth wave when using the sample plotter (see Appendix, Figure 3).

More information about the implementation of the wave is included in the next section.

Implementation

Overall structure

The generation of each y value passed to `BSP_AUDIO_OUT_Play_Sample` is handled by the main loop, which runs infinitely.

Within this loop there is a for-loop, `additive`, that calculates the y value for each of the harmonics. This loop runs once each timestep for each harmonic being summed.

The y value for each harmonic is calculated using the function `sin_x`.

More details for each of these structures are given below.

Main loop

The main purpose of this loop is to generate a y value for each timestep. To do this, it runs the additive for-loop, and then plays the sample that has been generated.

The other function of this loop is to increment the time counter, stored in r8, wrapping it around 360. The time counter is wrapped around 360 because all integer multiples of the base wave restart their cycle at this point.

Additive loop

The purpose of this loop is to calculate the y value for a harmonic at the current time timestep (t), and add it to the running total for that timestep, which is stored in r10. The running total is reset to 0 at the start of each iteration of the main loop. The current harmonic number is tracked using the loop index, stored in r9.

Once the value of the last harmonic has been calculated, and the loop exited, the value in r10 is moved to r0 and BSP_AUDIO_OUT_Play_S [illegible]

The y value is calculated by ca[illegible]ion, with the parameters adjusted for the current harmonic number (n). These adjustmen[illegible]able 2, below.

Table 2: Adjustments made to paran[illegible]

Parameter	*Adjustment*	[illegible]
x	x = n*t	Multiply the timestep by the harmonic number, thus also multiplying the frequency by the same amount
	x -> 0 <= x <= 360	After multiplying x by n, it is reduced by 360 until it is within this range, as this is an assumption of sin_x
Amplitude (A)	A = A/n	Scale the amplitude by the inverse of the harmonic number

sin_x function

This function takes the parameters as described in Table 3 and outputs an approximation of A*sin(x) in r0.

Table 3: Parameters of sin_x

Parameter	*Register*
x (in degrees)	r2
Amplitude (A)	r3

The sin_x function utilises Bhaskara I's sine approximation formula (Wikipedia, 2019). This equation is shown below:

Equation 1: Bhaskara I's sine approximation formula (Wikipedia, 2019)

$$\frac{4x(180-x)}{40500-x(180-x)}$$

To generate a wave at the correct amplitude, the result is multiplied by the amplitude parameter.

However, this formula does not work for all values of x. As shown in the Appendix, Figure 4 this formula only works for x values between 0 and 180, i.e. those that produce a positive y value. Therefore, I had to modify my implementation to reduce x values where sin(x) should be negative by 180, then multiply by (-1) to give a negative y value.

Reflection

One of the main problems that I faced was getting the 'additive' part of my additive synthesis to work. When using the additive loop to generate one harmonic at a time, both of the first two harmonics worked as expected, however when using the loop to add the first two together, the plotter was showing a wave that was completely wrong. At first I thought it was because of how I was adding the values of each harmonic. However, after spending a long time trying to debug the addition, I realised that it was due to how I was wrapping the timestep value.

The wrapping was incorrect because it was done after the additive loop, and was based on the most recent harmonic number. This meant that it was wrapping too early for the waves of lower harmonics. For example, if the loop was adding the first two harmonics, the timestep would be wrapped around 180 as this is where the 2nd harmonic restarts its cycle. This does not work for the first harmonic, which does not restart its cycle until 360. Because there is only one harmonic number involved when graphing the waves individually, the problem did not present itself until adding waves together. To fix this, I added another set of wrapping within the additive loop, which is described in the additive loop section above.

If I were to spend more time on my program, I would probably use it to lower how many clock cycles it takes to calculate the y value for each timestep. This would allow me to add more harmonics and generate a closer approximation of a sawtooth, while still producing a clear sound at the specified sample rate.

Reference List

Desmos, 2020. *Desmos Graphing Calculator.* [Online]
Available at: https://www.desmos.com/calculator
[Accessed 9 April 2020].

Sievers, B., 2006. *A Young Person's Guide to the Principles of Music Synthesis.* [Online]
Available at: http://beausievers.com/synth/synthbasics/
[Accessed 5 April 2020].

Szynalski, T. P., n.d.. *Online Tc* [illegible]]
Available at: https://www.szy[illegible]rator/
[Accessed 8 April 2020].

Wikipedia, 2019. *Bhaskara I's* [illegible]*rmula.* [Online]
Available at: https://en.wikipe[illegible]a_I%27s_sine_approximation_formula
[Accessed 5 April 2020].

Wikipedia, 2020. *Additive synthesis.* [Online]
Available at: https://en.wikipedia.org/wiki/Additive_synthesis
[Accessed 5 April 2020].

Wikipedia, 2020. *Sawtooth wave.* [Online]
Available at: https://en.wikipedia.org/wiki/Sawtooth_wave
[Accessed 5 April 2020].

Appendix

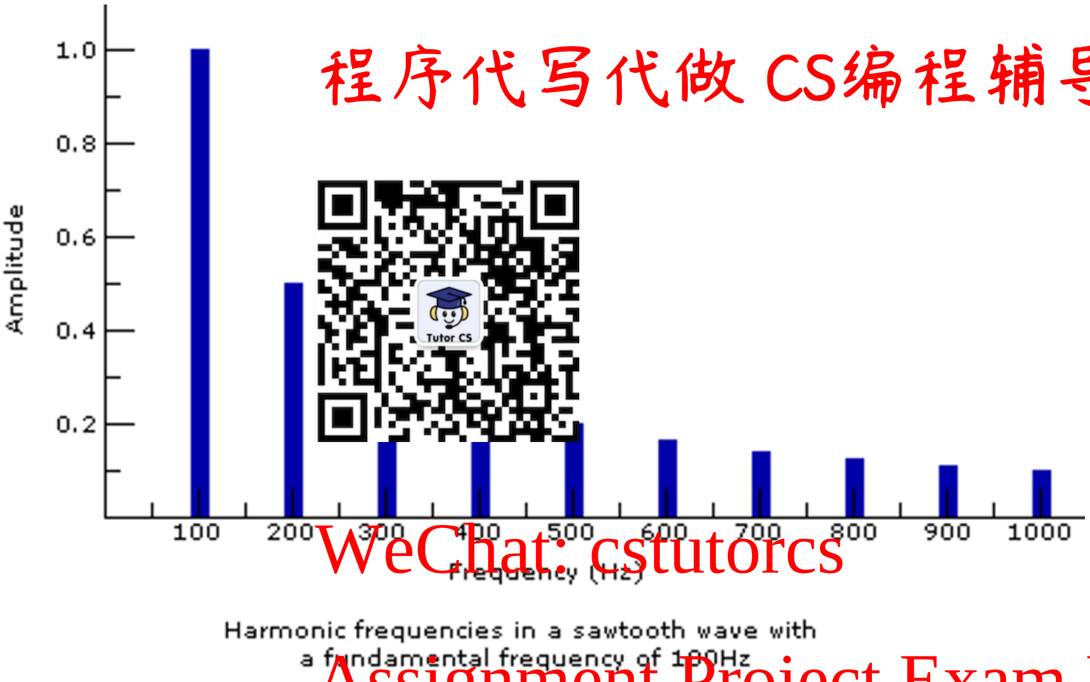

Figure 1: Frequency and amplitude for harmonics of a 100Hz wave (Image source: (Sievers, 2006))

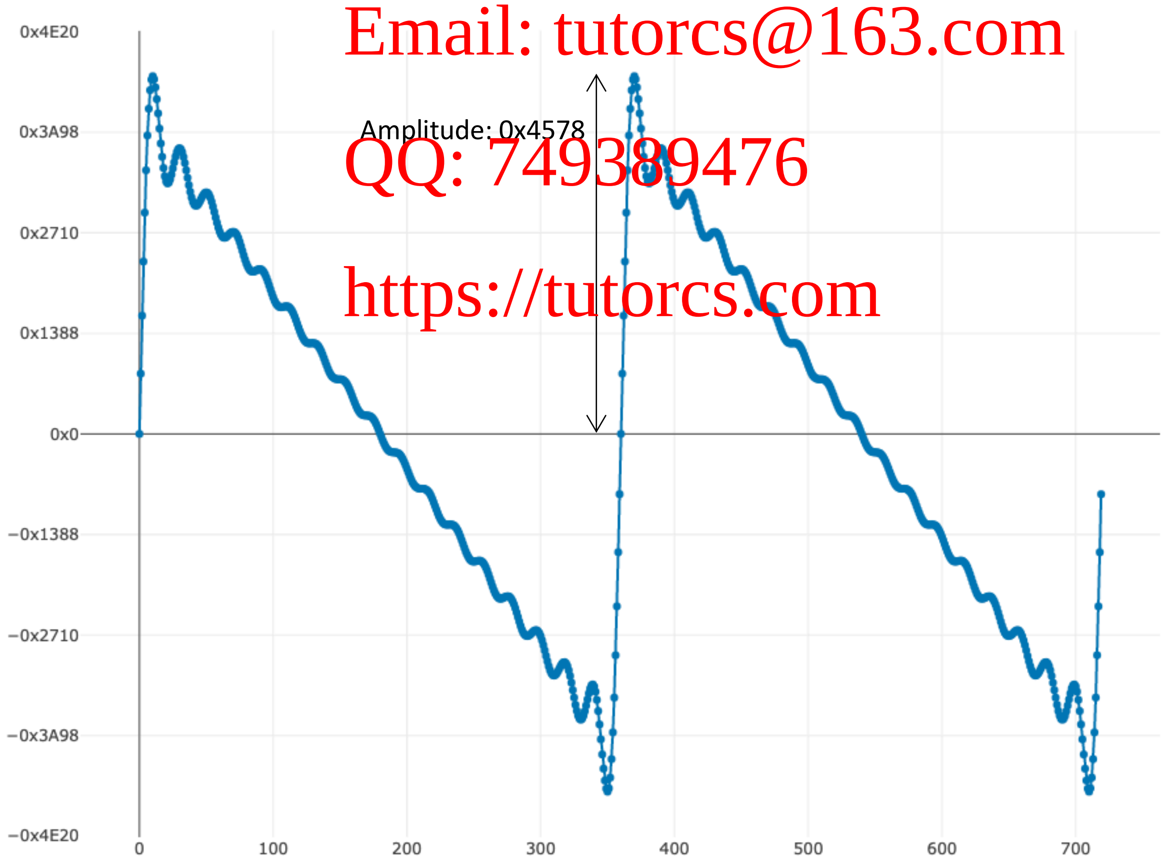

Figure 2: Sawtooth generated using 17 harmonics (submitted waveform)

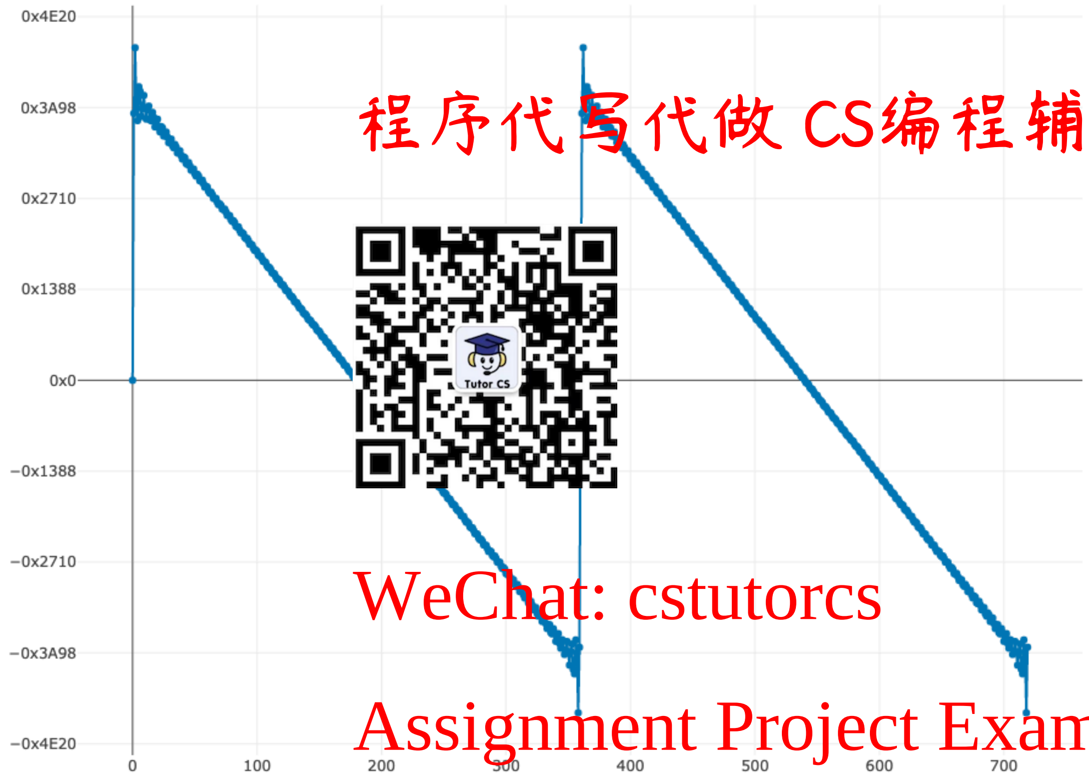

Figure 3: Sawtooth generated using 100 harmonics

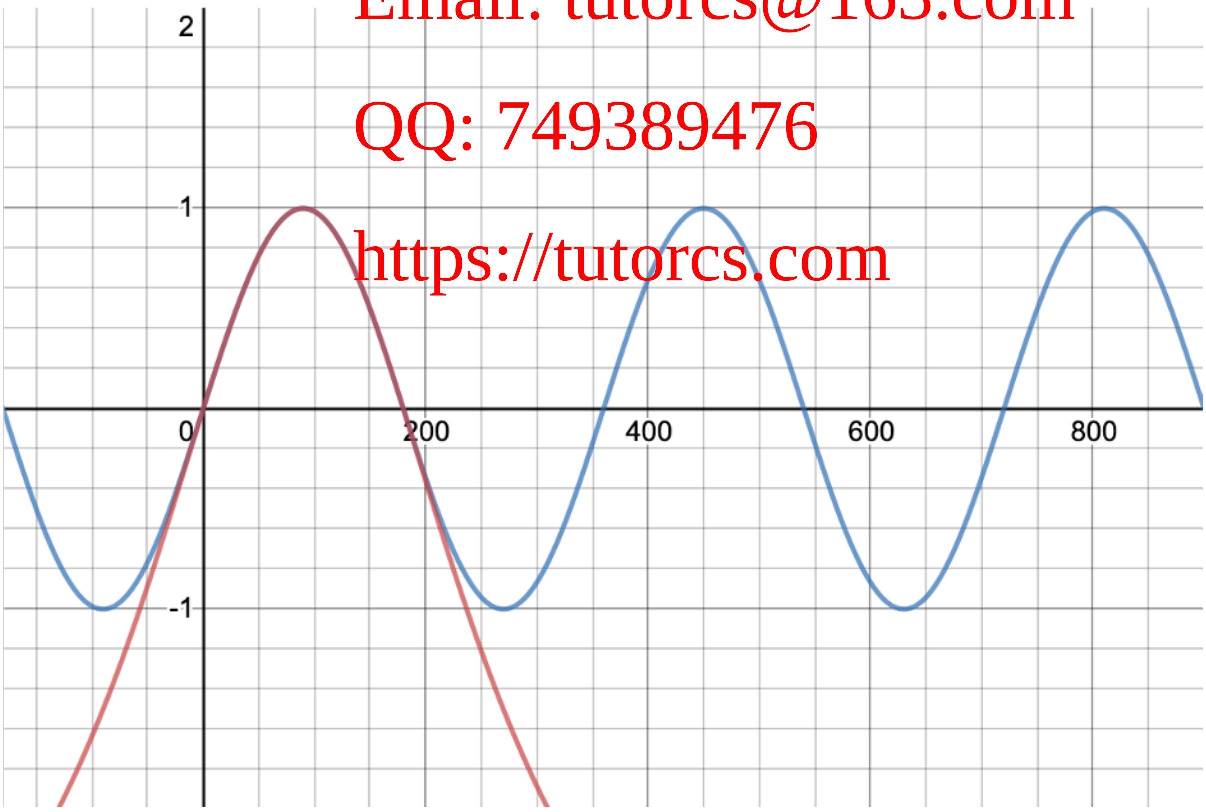

Figure 4: sin(x) (blue) and Bhaskara's sine approximation formula (red) (Image created using the Desmos Graphing Calculator)