ECSE 324 Lab 5 Synthesizer Group 47

Nihal Pakis- 260733837
Abdallah Basha- 260719132
Fouad El Bitar- 260719196

December 4th 2018

1. Make Waves

- ***Description:***
  - For this lab we were asked to write a code in C that takes an input of frequency, time and amplitude returning signal [t] using the two equations shown below:

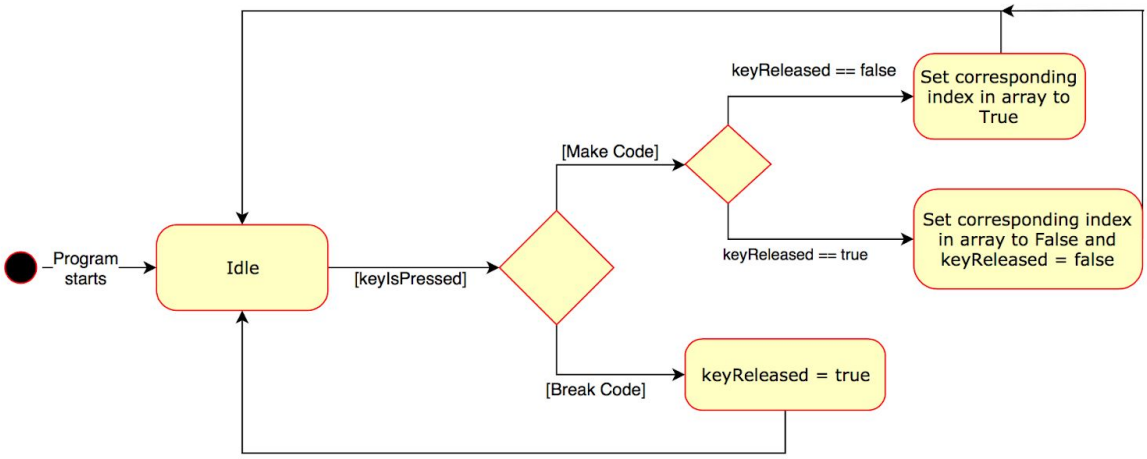
$$\text{index} = (f * t) \bmod 48000,$$
$$\text{signal}[t] = \text{amplitude} * \text{table}[\text{index}].$$

  - The wavetable provided to us has one period of 1Hz sine wave with a sampling frequency of 48000 Hz.
  - The program first stores all the keys that are currently being pressed inside a global array that other methods will need to generate samples where multiple keys are pressed. The program polls for a key press and when one occurs either a make code or a break code is sent, if a make code is sent then we want to see if the key that was sent before this was a break code and we do this using a boolean variable. We then reset the boolean variable after clearing the array specific index.
  - If more than one note is played at once it is expected to sum both samples for each given frequency in order to produce one output.
  - If the index turns out to not be an integer the code interpolates the two nearest integers, samples and adds them together as shown as exemplified below:

$$\text{table}[10.73] := (1\text{-}0.73)*\text{table}[10] + 0.73*\text{table}[11]$$

  - This method is called linear interpolation.
  - A timer is used to systematically input the samples made to the audio codec for which the drivers have been provided.
- ***Problems and Improvements:***
  - One of the issues for this lab was figuring out how to play multiple keys at one time. The issue was that if one key was held and the make code was repeatedly sent, once another key was pressed that make code would no longer be sent so we had to design the software in a way that allowed the signal to incorporate all keys

pressed until the break code was sent. The figure below describes the solution we came to:



2. Control Waves

- ***Description:***
  - For this section of the lab we wrote code that was able to control the waves using the keyboard buttons A, S, D, F, J, K, L and the semicolon (;), where each button represents a note from the octave note C-C with a corresponding frequency in Hz mapped to it as shown in the table below:

Table 1: Note Mapping

Note	Key	Frequency
C	A	130.813 Hz
D	S	146.832 Hz
E	D	164.814 Hz
F	F	174.614 Hz
G	J	195.998 Hz
A	K	220.000 Hz
B	L	246.942 Hz
C	;	261.626 Hz

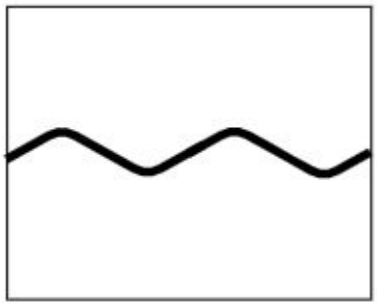
  - Volume control was also implemented with the keyboard by altering the waves amplitude in order to correspondingly raise or lower the volume. We chose two keys to use for incrementing and decrementing the amplitude value and optimized it so that only when the key was pressed we would change the screen pixels to incorporate the change.

- An 8-index array named ‘keyPressed[]’ is used in order to store which buttons have been pressed and which have not. This is initially declared with zeros stored in each of its indexes.
- The variable keyRelease checks if a key has been pressed then a switch statement checks specifically which button has been pressed.
- The next step is to actually generate the wave corresponding to the button pressed in which the ‘generateSignal’ method was used. As long as the button is continuously pressed the the signal will be continuously sampled.

- ***Problems and Improvements:***
  - It’s important to notice that this program was specifically designed specifically for the frequencies and corresponding keys we associated them with. If we wanted to expand this program we would want to add an option where we could edit the keys to correspond to different frequencies, this way we might run out of keys to press but we could easily change the frequencies to obtain different sounds.
  - We could have also improved this program by replacing the poll for the key press with an interrupt.

3. Display Waves

- ***Description:***
  - For this section we were asked to display the waveforms produced on the screen in the form of a table and graph. An example of the table is shown in section 2 where as the graph is shown below:



    - To implement the graph we mapped the value of time onto the x-axis value of the screen and the magnitude onto the y-axis of the screen. Only less than 1% of the values are used due to the screen only containing 320x240 pixels giving a limit of 320.
    - A value of 120 was added to the signal to shift it across the screen to the center.
    - The signal also did not fit the screen initially so its value was divided by 1,000,000 in order to get it to a reasonable size that fits.
- ***Problems and Improvements:***
    - The only challenge here was to filter the signal well in order to have it fit correctly within a reasonable runtime through trial and error. The code itself was fairly simple to write.
    - An improvement would having the signal be continuous i.e. displaying more so that it is a line rather than a trail of dots.