# Laboratory No. 5 - Synthesizer

Gabrielle Doucette-Poirier (260738518) | gabrielle.doucette-poirier@mail.mcgill.ca

Ryan Servera (260741736) |ryan.servera@mail.mcgill.ca

ECSE 324 – Computer Organization

Montréal – Canada

## Part 1: Make Waves

In this section we were tasked with making the wave associated with a given note given a time (t) and a frequency (f) as the inputs of our makeNote() function. The wave was made using the given formula to find the index:

```
float ft = (frequency * time);
int ftFloor = (int) ft;
float ftRemainder = ft - ftFloor;

int index1 = ftFloor % 48000;
```

Then once the index was obtained, it was used to the find signal in the wavetable. s file. In the scenario that the index is not an integer then, signal can be found using linear interpolation, with the floor and the ceiling of the found decimal index. In our implementation the resulting f * t, would be truncated to get the floor of the ft value. That floored value would then be used to get the first index. Then with the first index, the second index could be found by taking the next index. Next, linear interpolation would be performed using the code:

```
int index2 = index1 + 1;

int signal = (int) ((1.0-ftRemainder)*(float)sine[index1] +
ftRemainder*(float)sine[index2]);
```

Then once the signal was calculated it would be returned by the makeNote() function. Finally, with that note, it would be possible to change its amplitude depending on the volume set for the synthesizer as well as the number of the keys pressed. Dividing by the number of the keysPressed is used to normalize the wave since all the signals are the sum of all the notes made (which will be explained in the section 2). Overall, resembling the formula:

```
signal = signal * amplitude / numberOfKeysPressed;
```

Then to ensure that the signal would be loaded using a sampling rate of 48000 Hz, our implementation had an interrupt flag that would be set every 20 microseconds:

$$1/48000 = 20 * 10^{-6} \text{ sec}$$

Then to ensure a cleaner sound, the output signal would only be loaded if the audio port's queues were empty:

```
while(!audio_write_data_ASM(signal, signal));
```

## Part 2: Control Wave

As for this section the notes made would be assigned to a button on the keyboard given the following specification:

| 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 |

As for volume up, it was assigned to the up arrow and volume down was assigned to the down arrow. When one of these is pressed the amplitude would either be multiplied or divided, respectively, by a factor 1.75.

To ensure that the keyboard buttons could be constantly read when it was pressed down and not read when it was released, our implementation set a flag which would be set if the key code "0xF0" would be recorded. If a button is pressed, the program would record it, until the key was released, triggering the flag and making it so that the button would no longer be recorded.

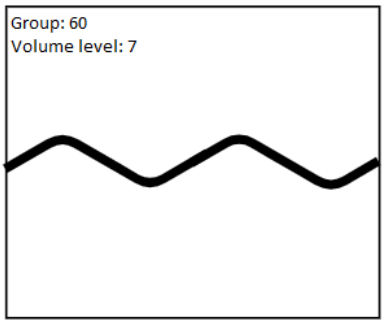
```
case 0xF0:
      isBreakCodeTriggered = 1;
break;

case 0x1C:
      if(isBreakCodeTriggered){
            isBreakCodeTriggered = 0;
            keyMap[0] = 0;
      }
      else {
            keyMap[0] = 1;
      }
break;
```

To make the overall signal, our implementation would make use an array which would map to the buttons pressed down. If the key was pressed down then the mapping array would store a value of 1 to that index, otherwise it would have a value of 0. Then with that array of mappings, the array would be incremented through to make a note with the frequency assigned to that index and then summed up to a total signal:

```
for (i = 0; i < 8; i++){
      if(keyMap[i] == 1){
            numberOfKeysPressed++;
            signal = signal + makeNote (time, frequencies[i]);
      }
}
```

## Part 3: Display Waves

The final section of the lab asked that we display our waveform on the computer monitor as shown below. The team also opted to display group number and volume controls in the top-left corner of the screen.




Our approach involved two variables created to keep track of where on the screen a pixel needed to be modified; the y position where the pixel was last written to a specific pixel column is kept track of such that the program does not have to erase the entire row (which would have led to latency and distorted sound). The rate at which the program writes to the screen was decreased via a counter, only drawing a pixel to the screen every 10 sound samples.

120 is added to the y-value taken from the amplitude to offset the wave to the center of the screen. The number from the resulting computation is the y-position of the pixel to be written.

```
VGA_draw_point_ASM(x, rowColumnPosition[x], 0);
VGA_draw_point_ASM(x, 120 + y, 16000);
rowColumnPosition[x] = 120 + y;
```

The x-position is calculated using x = (x+1) % 320, which wraps them at the width of the screen. The y-position is drawn by dividing the signal by a significant value which we originally set as 900000, but found worked best when we reduced is to 240000. This process was iterative in our reduction of the values until we were satisfied with the display.

We also added the volume control to the screen which we capped at a max of 9 and min of 0 (within the getNote() function). This prevents the signal from ever growing larger than the screen height or lowering below the ‘flatline’ response of zero unnecessarily.

## Part 4: Further Improvements

One of the improvements that the team would have liked to implement with more time would have been more notes to control via more keys, as well as octave control. Whenever a press would be registered, all the frequencies in the table could either be multiplied or divided by two depending on the direction of the change. This could have also added displaying the notes that were being played directly onto the screen.

Further functionalities considered for this synthesizer were the implementation of recording and looping sounds played. To do so, another key would need to be added to be able to record as well as another to play the recorded track back.

Lastly, it would have been ideal to have been able to calculate and optimize the signals such that there was no distortion when more than 2 notes were played at once. This could have meant pre-calculating sample values prior to the start of the program, but was not explored due to time constraints.