ECSE 324 - Lab 5 Report

ARCHIT GUPTA (260720757)
KAI LUO (260479742)

Part 1 - MAKE WAVES

For this part of the lab, we had to write C code that takes an input frequency and time and returns a signal[t], according to the equation provided to us through mycourses. The code was also supposed to implement addition of waves if we simultaneously press multiple keys. In addition, if the index was not an integer, we had to interpolate the two nearest samples and add them together. We wrote a method called generateSignal that returns the sample of the wave. The method takes in as arguments the frequency and time and returns the signal sample.

A 1Hz sine wave was provided, which contained 48000 samples. Initially, we implemented a for loop that looped through all 48000 points and calculated the sample at each point. But this method delayed things quite a bit. So instead, we used timers where the samples could be precomputed and were only written when enough time passed. The timer flag was raised every 20 microseconds, which was chosen because the FPGA samples at a rate of 48k/sec. The timer flag, hps_tim0_int_flag, is continuously polled and goes high every 20μs. The sine wave index, t, is only incremented when the audio sample is successfully written so as not to miss any values.

We faced some difficulties in properly implementing the timers. There were a lot of errors in the audio. But slowly, through a bit of trial and error, we fixed them.

PART 2 – CONTROL WAVES

For this part of the lab we had to control the waves using keyboard buttons A, S, D, F, J, K, L, and ;. Each of these buttons represented a different note and had a corresponding frequency.

Firstly, we declared an array of size 8 called keyPressed (corresponding to each of the buttons). We used a variable called keyReleased that was used to determine a key release. We then defined 11 switch cases, 8 for the buttons, 2 for the volume buttons and the last one for the break code. The switch cases consisted of an if-else block. If keyReleased equals 1, set the KeyPressed array corresponding to the specific button to 0 and then reset keyReleased to 0. Otherwise, the key has not been released but pressed.

For the break code case all the characters started with the same break code ‘F0’. Therefore we would simply set keyReleased to 1 to indicate that the next character has been released. We chose the volume keys U and I because U stands for ‘UP’. The max amplitude we could reach was 10 and the minimum was 0.

To generate the actual wave we used a method called generateSignal(char* keys, int t) to continuously generate the sample of the signal. The method takes in as arguments an array of pressed keys. There is another array, frequencies[], that holds the values for the frequencies of each note. This was provided to us in mycourses. generateSignal loops through the array of keys pressed and sums up all the samples from getSample corresponding to each frequency. Pressing more than 4 keys results in interference.

The timer was used to determine if it was time to write the audio to the output. The ‘time’ variable, t, was only incremented if the audio signal was written so that no samples were missed.

<u>Part 3 – Display Waves</u>

The last component of the lab was to display the waves generated onto the screen. To do this, we mapped the value of time on the x value of the display and the sample magnitude on the y value of the display. The screen size was 320*240 pixels. Instead of clearing the entire wave and displaying the next wave, we used another array that stored the y values. Using this array helped us save a lot of time while displaying the waves.

While displaying the signal, we had to center it. To do this, we add 120 to the x value of the signal. We also made some other small mathematical computations to resize the wave so that it displays correctly.

IMPROVEMENTS

Coming up with improvements is always tough. On top of the screen we displayed our names. We could also display any other message. This was not technically an improvement but rather a separate challenge that we took on. We implemented a volume display that showed the current volume level of signal from 0-10. When four or more keys are pressed at the same time on the keyboard the sound becomes distorted. This is since each additional frequency sample is added to the rest. Eventually the number becomes too large and starts to overflow which causes the interference as also explained above. We tried fixing this using normalization but were not able to achieve much success when 4 or more keys were pressed.

For the timer, we calculate the signal again on every pass. One way of saving time and hence improving the code could have been to only calculate the signal again if t has been incremented.