Karl Doumar 260733160 April 16th, 2018

He Qian Wang 260688073

Group 32

ECSE 324 Lab 5 Report

**Introduction**

The objective of this lab was to take the subroutines and code that we have written in the previous labs and build a musical synthesizer with them. We were provided with a drivers API in case that we were missing code. We should be able to generate sounds of varying frequencies when keyboard keys are pressed. The corresponding waves should be generated on the monitor with the use of a VGA cord.

**Part 1 – Make Waves**

The first part of the lab consists of creating the sine signal waveforms that will be used later in the lab. We decided not to compute the amplitude in order make\_sine function. The amplitude would be implemented later with the audio component of the lab. This part of the code simply follows the formulas and methodology given by the lab document. We would approximate the sine using the wavetable assembly code that was provided.

We decided to make a 2D array of 48000 by 8 table of every possible value of sine waves of each of the 8 different frequencies. We would generate each sine value and place them into the array. Later in the program, when we will need to use these values, we would simply fetch it from this array. We know that an array is instant access and we would not slow down the rest of the program by doing this.

**Part 2 – Control Waves and Audio**

The way we decided to feed our signals to the audio and to keep track of the keyboard is with the usage of two different timers. One timer has a timeout of 20 and it is used to periodically feed our signals to the audio FIFO. The other timer has a timeout of 10000 and it is used for the keyboard interrupt; whenever a key is pressed, we would stop and go in our subroutine.

The way we decided to monitor our signals is with the use of 8 different flags for each of the 8 notes. Whenever any of the 8 musical keys are pressed, we would set the flag to active. When we receive a break code F0 from the FPGA, we would use this flag to set the note flags to 0. Our system makes use of the F0 keys to make sure that we stop the sound whenever the keys are released. We have a variable declared for amplitude outside the infinite while (1) and whenever any of the two amplitude control keys are pressed which are V and B, we would update the value of that variable. We decided to place a lower end limit for the value of the amplitude. Sometimes, when the amplitudes gets too low, it would be set to 0 and we would be unable to further increase the amplitude back to normal again.

After each iteration, we check each of the 8 flags for keys pressed and we would add each sine signal to our variable sumSignal. This way, we would account for the cases when multiple keys are pressed simultaneously. At this stage, we would simply fetch our values from the signals table that was generated at the beginning of the program. This way, we save a lot of time and our system becomes more efficient.

If the timer for the audio is active, we would write our signal to the audio FIFO and it would be output be the system. We would simply need to use headphones to able to hear the melody.

**Part 3 Display Waves**

Our display when no key is pressed is a simple line in that spans from 0 to 320 on the x axis at a height of 120. Whenever a note is pressed, we would simply output the shape of the waveform on the monitor. We decided not to implement scalability of the waves with the amplitude as it simply bugged our code too much. Our waves are of the same shape whether the amplitude is at maximum or it is at minimum.

The way we display the waveforms is with whenever a key is pressed, our interrupt service flag becomes active and we perform our subroutine before going back to the main code. We have two arrays, one of the arrays is to remember the previous values of our sine waves, the second array contains the height of the 320 points that were displayed. The first thing we do is to manually go to each of the previous 320 heights and set them to 0x000000, which effectively clears the display at that point. Then, we would see if the sum of the signals is 0 or if it is not trivial. If the sum of signals is 0, then we simply draw a line in the middle of the screen.

If the value of the sum of signals is non-zero, then we would simply draw our points on the monitor depending on each value that was obtained at the sine wave. Those values were stored in the array that contains the present sum of signals. We do not scale the wave based on the amplitude, so the waves should occupy the whole screen by default. We implemented alternating colour by using a variable colour that starts at -32767 and will increment each cycle until it reaches 32767 where it will be reset to -32767.

The challenges we faced during this lab is the optimization of the system. We needed to find ways to reduce the time spent in each subroutine because if we spent too much time in certain places of the code, the audio would lag the display or vice versa. Initially, we did not compute a 48k by 8 table of every value for each frequency and its sine values. However, we realized that computing the value of the sine signal took way too long if we were to interrupt the main code and go in a subroutine every time. Having the table ready helped us tremendously in optimizing the code.