Patrick Ghazal – 260672468
Kartik Misra – 260663577

ECSE324: Computer Organization
Lab 5 Report
Group 7

## Introduction

As the last lab of the semester, Lab 5 was an opportunity to combine and demonstrate several of the features that have been developed in the past labs. For instance, we learned in Lab 2 to call assembly code from C code, and Lab 4 was especially key in teaching us to use the VGA display, PS/2 keyboard, and audio controller to input or output data from the board.

The specific goal of this lab was to implement a (simplified) synthesizer, with the PS/2 keyboard acting as a piano, the VGA display showing the generated sine waves, and the audio controller being used to transmit the sound to earphones.

## Make Waves

The purpose of this section was to write a *make_wave* method that would receive a frequency *f* and a sampling instant *t* as input, compute the value of the corresponding signal, and write that signal to the audio codec.

We were provided with several drivers (and driver skeletons) allowing us, among other things, to write data to the audio codec. Another key driver was the wavetable.s file that contained one period the data for one period of a 1 Hz sine wave at a sampling frequency of 48 kHz.

The math behind this method is the following:
Given a frequency *f*, sampling instant *t*, amplitude *A*, the signal is:

$$\text{index} = (f * t) \bmod 48000$$
$$\boldsymbol{signal}[\boldsymbol{t}] = A * table[index]$$

A difficulty that arose when writing this method was the fact that modulus operations apply to integers, and the frequency being a double value, its product with the sampling instant is more often than not a double itself. This means we had to cast the product into an int before calculating the mod operation on it, and this resulted in a loss of precision on the data. That said, the errors generated are tiny enough that they are inaudible to the human ear.

In this implementation, the make_wave function had to be called periodically, in order to generate a continuous sound. A signal is generated and written to the audio codec using the *audio_write_data_ASM* subroutine.

## Control Waves

For this section of the lab we made use of the 0, 1 and 2 timers. As long as the flag responsible for timer 0 is set to 1, the input from the keyboard will be read, and then it will be set to 0.

To get the program to work for keyboard presses, we used integers 0 and 1 (which act as booleans) in order to determine whether a specific has been or is being pressed or not. Like in Lab 4, we use a register to check if a key is being pressed (the read_PS2_data_ASM). We then compare the value at that register with the make code of a key. In lab 4 we saw that the make code for "a" is 1C, therefore we know that key "a" is being pressed. In order to tell the program, the key is still being pressed, there is another if statement checking if a break code is registered (again, for "a" the break code would be F0 1C), which means the key is no longer being pressed.

We also implemented a volume up/down feature using the n and m keys. These work just like any other keys, with the register being checked and the break code being read to know the key's status. However, since the m key increases volume, we don't want the volume to increase each time the make code for m is read (for example, if the register is checked 5 times per seconds and we hold the button for 3 seconds, it will increase the volume by a factor of 3*5 = 15), we want it to increase the volume once at each button press. In order to do that, we decided to increase (or decrease) volume only when the break code is read, i.e. when the key is released, that way the increment only happens once.

After input from the keyboard has been determined we make the required signal using the make_wave method described above, we check for audio fifo spaces and write the audio signal that'll be outputted.

The main problem we faced with this section was the poor quality of the signal after more than 2 keys were pressed.

## Display Waves

Finally, the lab required us to display the signal we produced using the *make_wave* method and the keyboard on a display using the VGA output port, this used the timer 2.

We made use of an array that took in a signal value. In order to get the signal to be centered on the display. Since the height of the display is 240 pixels, we had to offset the signal position by 120.

Another problem was the scarcity of points displayed that make up the wake when a specific button was pressed. To get around that, using trial and error, we arbitrarily divided t by 9 and applied a modulus of 320 that was compared to x to be sure that the correct points would be drawn. We also cleared the space at the next pixel where the next pixel would be drawn in order to have a moving sine wave.

Our biggest challenges with this section was to get the wave to display properly on the screen, we would often get half of the wave, or just a single colored pixel displayed, even though the audio worked flawlessly. Moreover, if more than 2 buttons were pressed, the signal became too complex to be displayed of the screen (since using double didn't work we had to cast to int) so there was serious distortion observed.

## Conclusion

In conclusion, despite several issues that we were unable to resolve, this lab gave us a good overview of the DE1-SoC board's capabilities. It allowed us to combine several features at once in order to create a full synthesizer system, with data inputted to the system through a keyboard, and data outputted in audiovisual form on a screen and through earphones. Overall, this course was key to both computer and software engineering students, providing us tools to design programs on a lower-level than what we've seen so far in our classes.