ECSE324 COMPUTER ORGANIZATION

Lab5 Report

Group 29

Yi Zhu

260716006

Shaluo Wu

260713923

2018-12-02

1. Introduction

This lab is an integration lab which consists of all materials we learned in this semester. The main objective is to use the PS/2 Keyboard and Audio function on the De1-SoC Computer to create a musical synthesizer. The whole design process is divided into three main sections. The first part is to generate the signal of the piano by using the formula provided in the lab manual. The second part is focusing on PS/2 Keyboard controller and the third part is about displaying the wave that generated by the keyboard on the screen. All the drivers for writing to the audio codec and displaying to VGA are provided. The only section needs to be filled in is the main C language class. The main class is the section to implement the logic of transferring key that has been pressed on the keyboard to related sound that to be played and waveform to be displayed on the screen.

2. Methodology

2.1 Make Waves

In this part of lab, we are asked to generate the piano key signal for playing. We learned from previous labs that the sampling frequency of the De1-SoC Computer is 48000 Hz. Both the wavetable containing one period of a 1 Hz sine wave of the piano and the formula of playing a note with frequency ‘f’ using the wavetable are provided. The formula is shown in the figure below:

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

Figure 1 Formula for Signal

We then create a method named ‘generateSignal’ based on this formula. In this method, we have two inputs: one float type variable frequency ‘f’ and one integer variable time ‘t’. The return value of this method is a float variable ‘signal’. In the body of this method, we generally followed the formula provided to calculate the signal. However, since the index we calculated using ‘mod 48000’ will not be an integer in most cases, we need to calculate the signal[t] by linear interpolation using two nearest samples. The sample calculation process is also provided in the lab manual, shown in the figure below:

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

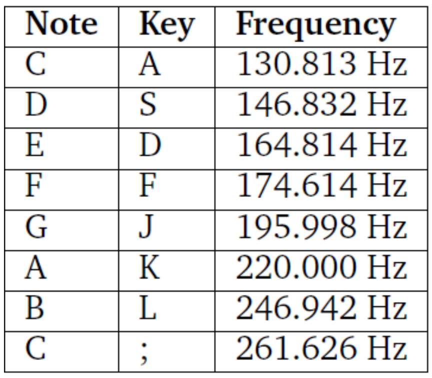
Figure 2 Sample Calculation

We noticed that two variables between the ‘mod’ method have to be both in integer type. So, we first convert the result of multiplication of ‘f’ and ‘t’ to be an integer value and then do the mod process. After getting the index value, we store it in a float variable ‘index’. To use the method of linear interpolation, we need to separate the index into integer part and decimal part. We first create a new variable ‘intIndex’ to store the integer part of the index. This ‘intIndex’ variable is the result from converting the float ‘index’ variable into an integer. After that, we subtract the ‘index’ variable by the ‘intIndex’ and then we get the decimal part of the index. We create another float variable named ‘decimalIndex’ to store it. Finally, we followed the formula provided in the lab manual to generate the signal we want to play. The final result is stored in a float variable named ‘signal’.

Since the basic design idea and formula of generating the signal are already provided in this part of the Lab, we did not meet any challenge. The methodology is straightforward, we do not think there will be further improvement.

2.2 Control Waves

In this part of the lab, we are asked to achieve the control functionality of the music synthesizer using the PS/2 keyboard. The goal of this section is to use the key on keyboard as a trigger to trig and decide which sound that will be played. The relationship between keys on keyboard and note is shown in the figure below:



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

Figure 3 Table of Note, Keys and Frequency

The general idea of this section is to use the break code to check whether a key is pressed or released and to use the PS/2 keyboard code to check which key is pressed. There is an integer variable named ‘keyReleased’ is used as a Boolean variable to check whether a key has been released. We use a switch method with 11 cases and 1 default setting to achieve key-press-check. We also use two arrays, one named ‘keystate’ is used to store the status of 8 keys for notes and another one named ‘freqs’ is used to store the 8 frequencies for sound of 8 notes. The working mechanism is if there is a press on the specific keys on the keyboard, first the read_ps2_data_ASM(&data) method will get the PS/2 keyboard code of the input, and then it will go into the switch method to check which key is pressed. After the computer acknowledged which key is pressed, it will check whether the key has been released. If the read_ps2_data_ASM(&data) method read a break code, it means the key has been released. However, if the key is not released, it will set the key sate of corresponding PS/2 keyboard code in the ‘keystate’ array to be high. There are eight cases for eight notes, two cases for volume adjustment and one case for break code check. The default setting of ‘keyrelease’ is low. After all key-press-state check, a for loop will go through the ‘keystate’ array to check all high state keys, grab their corresponding frequencies for signal generating. If multiples keys are pressed at the same time, we will add the signals together. For signal generating, since we have the sampling time and current frequency for note, we will use formula duration =

(sampling time)/f to get the time duration we want to play the correct tone. An integer variable ‘t’ is used as a counter here and it is initialized as 0 at first. After generating the signal each time, ‘t’ will be increased by one until it reaches the time duration we calculated by the formula above. After we have the frequency ‘f’ and time ‘t’, we will use the method ‘generateSignal’ we created in the first section to generate the signal. There is another variable named ‘amp’ that used to control the amplitude of the signal, which is same as the volume of the sound. We will multiply the signal we generated by the ‘amp’ to control the volume. The ‘amp’ can be accessed in the switch case by pressing two keys ‘<’ and ‘>’on the keyboard. Finally, we will write the signal we generated into the sound output using method ‘audio_write_data_ASM(s,s)’. Then the control wave section is finished.

Since the drivers for playing sound have already been provided, the most challenging part of this section is to connect keys to corresponding note and play the correct sound. While working on this section we were worried about whether using the timer or not. Since the timer was simply used as a counter in previous labs, we decided to directly use a variable ‘t’ as a counter to make life easier. Thus, we do not need to change ant code in the ISR driver.

However, we find it difficult to draw the wave at the same time play the sound without using a timer. Therefore, the best improvement we can come up with is using the timer for both playing sound and drawing wave.

2.3 Display Waves

In this section of lab, we are asked to display the waveform of the sound generated by pressing the key on the computer monitor. The ideal waveform is shown in the figure below:

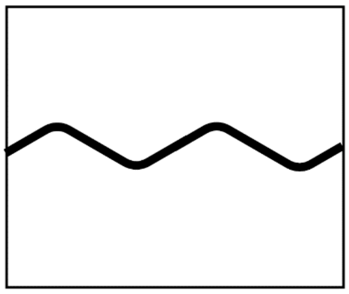


Figure 4 Waveform Display

To complete this part of the lab, we created a new method named ‘drawWave’. This method has two inputs frequency ‘f’ and amplitude ‘amplitude’, and it will only be called when a new key has been pressed. In the method body, we first clear the VGA and then initialize variables we will use for drawing the waveform. Since we have been provided with one period of 1 Hz sine wave at a sampling frequency of 48000 Hz, the only step is to sample it with current frequency and display the sampled waveform on the VGA. The width of the monitor is only 320 pixels, but we have 48000 samples in the provided sample waveform. Therefore, we need to make a gap in the sample waveform to choose which 320 points should be used to display. We choose the gap by dividing the 48000 by 320 and the current sound period ‘1/f’. Then we multiply the result by 50 since the result gap is still a small number compared to

48000 and it cannot display even one complete period on the screen. After choosing the gap, we use a for loop to display the sampled waveform on the screen. A new variable used as the counter in the loop is named ‘xPosition’. It starts at 0 and will be increased by one gap value each loop. Then we go through the ‘sine’ array provided by each ‘xPosition’ and multiply it by the current amplitude to get the result. Since we are also asked to make the volume adjustable, we will increase or decrease the amplitude of the waveform each time we increase or decrease the volume. We accomplish the amplitude-adjustable-waveform by multiplying 10 times the scale of current amplitude and the peak-to-peak value of the sample waveform each time we change the volume. Finally, we draw the waveform on the screen using a for loop that make x go through 0 to 319 and for each value of x, y is the waveform we calculated by sampling method above. We use the VGA_draw_point_ASM(x, y, colour) method to draw each pixel on the screen and the colour is value is ‘0xFFFFFF’ which stands for white.

The most challenging part in this section is sampling the sample wavetable provided to the waveform of current frequency. The improvement we can come up with is using a timer as a synchronized counter, thus each time we generated the signal for sound playing in the previous section we can directly display the waveform of it on the screen.

3. Conclusion

Since this is the last lab of this semester, it concludes all the stuff we learned in this course and previous courses such as ECSE222 or ECSE206. We found the logic of generating the signal for sound playing is the most challenging part of the whole lab. Besides on minimum functionality required on the lab manual, we also implemented welcome words and volume adjustment instruction display function on the screen. If we have more time for the lab period, we will also be able to implement a visual piano keyboard function.