ECSE324 COMPUTER ORGANIZATION

Lab4 Report

Group 29

Yi Zhu

260716006

Shaluo Wu

260713923

2018-11-07

This lab consists of three main parts. The first part is to work with the De1-SoC computer VGA interface, which has been introduced in ECSE222 labs before. The second part is focusing on PS/2 Keyboard controller and the third part is about the Audio controller on our De1-SoC Computer.

1. VGA

In this part of lab, we are asked to write five subroutines, two of which are to clear the screen, three are to write characters, bytes and pixel color respectively. At the main class of this part, we are asked to use the slider switches and pushbuttons which we have implemented in the previous lab to control the output on the screen.

1.1 VGA_clear_charbuff_ASM

In this part of lab, we need to clear the screen for character display. For the character display, we only use x from 0 to 79 and y from 0 to 59. So, we wrote two 'for loops' to go through all characters that clean the right most column from y=0 to y=59 first. Then left shift one column to clear until x reaches 0. For the clear method, we just write both x and y coordinates into the character buffer address and then store zero into this address to clear.

1.2 VGA_clear_pixelbuff_ASM

The clear method for pixel is almost same as for character. The only change is that x coordinates is from 0 to 319 and y coordinates is from 0 to 239. We also wrote two 'for loops' to go through all the coordinates and then took corresponding x y coordinates into pixel buffer address and then store 0 into each address to clear.

1.3 VGA_write_char_ASM

In the writing character method, we first check whether the input x y coordinates are in the range that given. If not, it will directly return to the main class. Otherwise, it will write the corresponding x and y coordinates into the character buffer address and then store the character we want to display in byte format into that address.

1.4 VGA_write_byte_ASM

In this method, we are asked to display two characters that passed into the argument. We first checked the range of x and y coordinates, then we check the first input number. We compare it with 10 that if it is less than 10, we will assign it with corresponding number in ASCII table, otherwise we will assign it with corresponding letter in ASCII table. Then we write the current x and y coordinates into the character buffer address and store the number or letter we assigned into that address in byte format. After that, we will move one more x coordinate and do the same for the second input number. Lastly, we can print the hexadecimal representation of two input numbers on the screen.

1.5 VGA_draw_point_ASM

This method is almost same as the previous character writing method. Firstly, we

will check whether the input x and y coordinates are in the range, which is from 0 to 319 for x and 0 to 239 for y. Then we will write the x and y coordinates into the pixel buffer address and then store the color we want in half-word format into the final address we assigned.

1.6 Application

In the application part, we are asked to combine these write and clear method with slider switches and pushbuttons. I implemented the code wrote in previous lab for these slider switches and pushbuttons. Meanwhile, three testing classes have already been provided. In the main class, I wrote five if statements to implement these tests asked from the lab manual.

The first challenge we met is how to put the x and y coordinates we want into the buffer address. After reviewing the DE1-SoC Computer manual, I solved it. The second challenge is when to write character we need to use STRB for character instead of STR and also STRH for pixel. Since the character is stored in one byte, and color for pixel is stored in half-word. With the small hint in the lab manual, we solved this challenge.

Since these classes are the basic methods to implement such write and clear function, we cannot come up with any further improvement.

2. Keyboard

In this part of we are asked to write a method to check the RVALID bit in the PS/2 Data register. The procedure is quite easy. We first load the PS2_Data_Base and only take the 16^{th} bit of it, which is the RVALID bit. We compared this bit with 0, if it is equal to 0 then we will return 0. Otherwise, we will use ADD instruction to get only the data byte and store it into the return address.

In the main class, since we are asked to display the char in 0,3,5 order, after we got a push on the keyboard, we will display it using writing byte method that wrote in previous section first and then move 3 more space in x coordinates. If the x coordinate reaches the border, we will jump to next line. When the y coordinates reach the border, we will jump back to the first line. The final output will be like ‘make code, break code’

The greatest challenge we met in this section is that once we completed the code, we pushed the button and there was something displayed on the screen, but we had no idea what they were or were they right or not. After we searched online for the scan code table, we finally realized the relationship between characters displayed and corresponding keyboard button.

Since the method is also the basic way to connect the keyboard and the VGA display, we cannot come up with further improvement.

3. Audio

This part of lab is similar to the previous section. We need to check the WSLC and WSRC value in FIFOSPACE_BASE. We first load the content in FIFOSPACE base address in to a register and used the shift method to get specific WSLC and

WSRC bit. We compared it with 0 if either of them is equal to zero we will return zero. Otherwise, we will store the data in both the left and right data base.

In the main class, since it wants us to create a 100Hz square wave and the sampling rate at 48k samples/sec. From the example in lab manual we understood the relationship between the sampling rate and frequency. So, we wrote a loop to check whether the audio space is still available, it will write 240 times 0x00FFFFFF and 240 times 0x00000000 into the memory to play the sound.

The challenge we met here is the 75% full described in the computer manual. We cannot understand how to check it at the first place. However, after deeply study the manual we found that there is no need to calculate the exact 75%. Since this method is quite straightforward, the only improvement we can come up with is in the main class we can write two closed for-loops to play the sound.

4. Conclusion

After this lab, we deeply learned high level I/O capabilities of the DE1-SoC computer such as character VGA display output, the pixel VGA display, the keyboard input and the audio output. We also learned how to control the input and output of these functions in C language class.