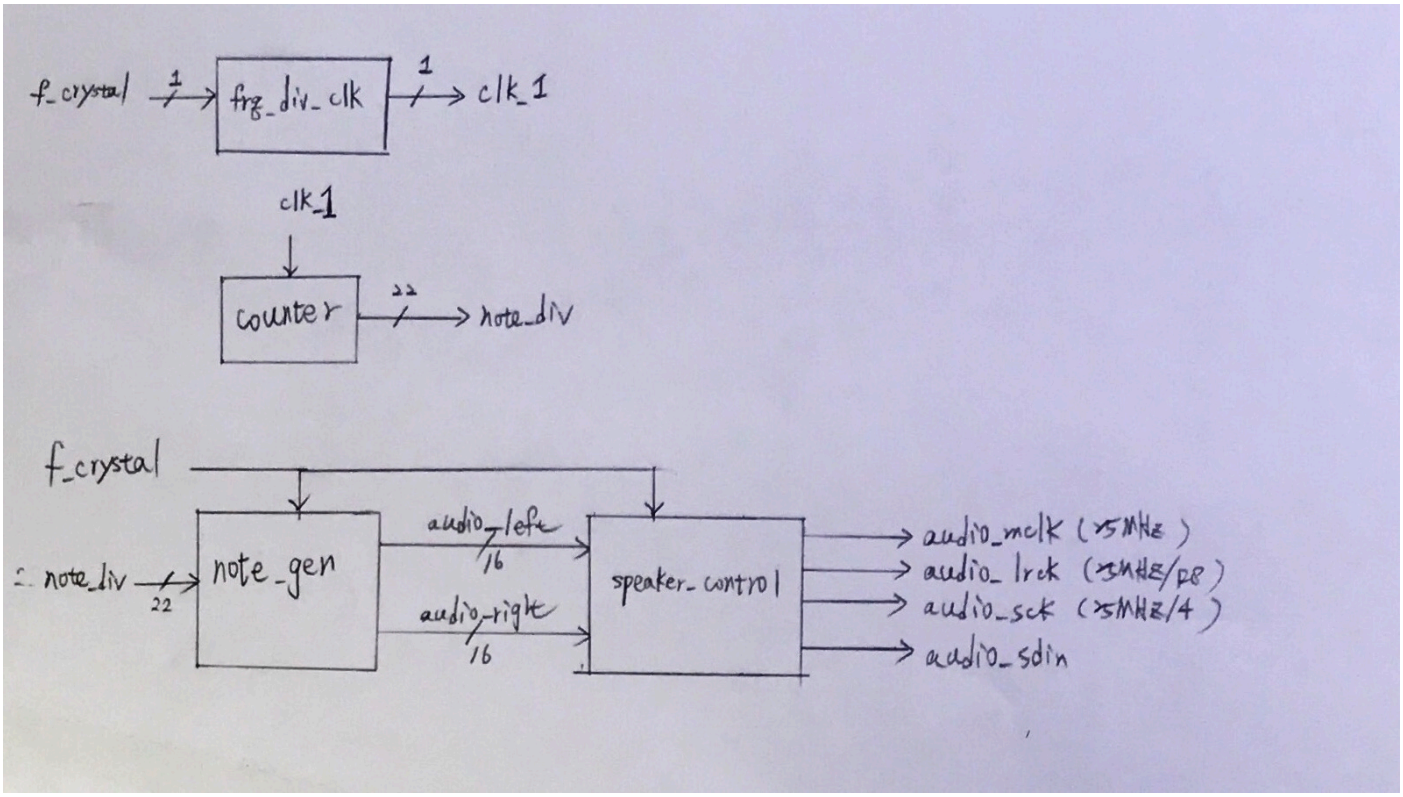
1. Play the 14 sounds repeatedly based on the sound table. Every sound is played for one second.

Design Specification:

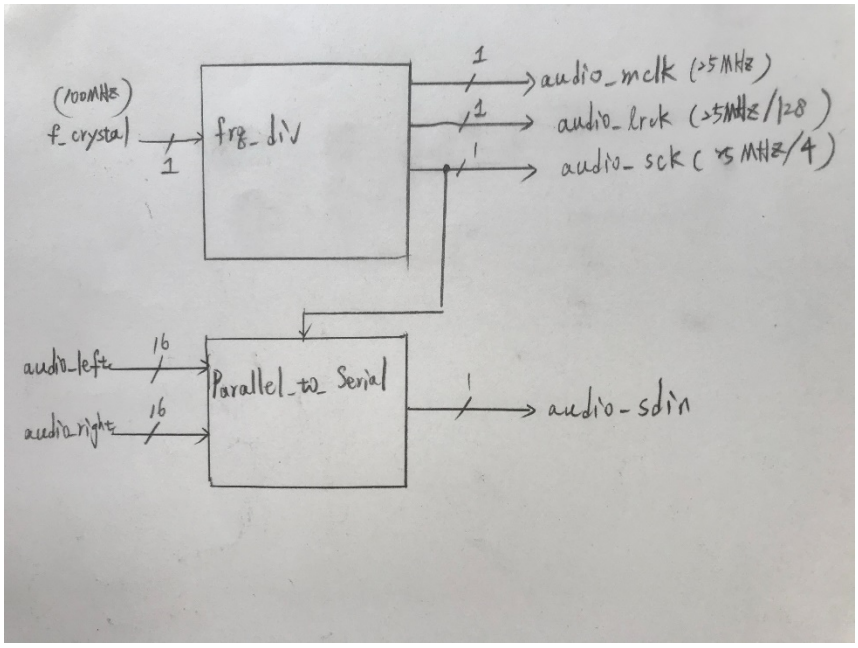
Input: f_crystal, rst

Output: audio_mclk, audio_lrck, audio_sck, audio_sdin

Block Diagram:



Inside speaker_control module:



Design Implementation:

I/O pin assignment:

f_crystal—W5

rst—V17

audio_mclk—A14

audio_lrck—A16

audio_sck—B15

audio_sdin—B16

In this experiment, we are required to play 14 sounds repeatedly and each sound is played for one second. So, in my counter module, I design a counter (using 1Hz clock frequency derived from frq_div_clk) and base on different counter value, it will output different note_div values. And by using the module note_gen and speaker_control we have done in Lab8, we can hear 14 different sounds.

Discussion:

By using note_gen and speaker_control from lab8 and adding a counter to choose note_div, this experiment can be completed.

2. Electronic Organ

2.1 Integrate the keypad as the keyboard of the electronic organ. Keys c, d, e, f, g, a, b, C, D, E, F, G, A, B (two octaves from mid-Do) represent the sounds from low to high frequencies.

2.2 Display your playing sound (Do, Re, Mi, Fa, So, La, Si) in the 7-segment LED.

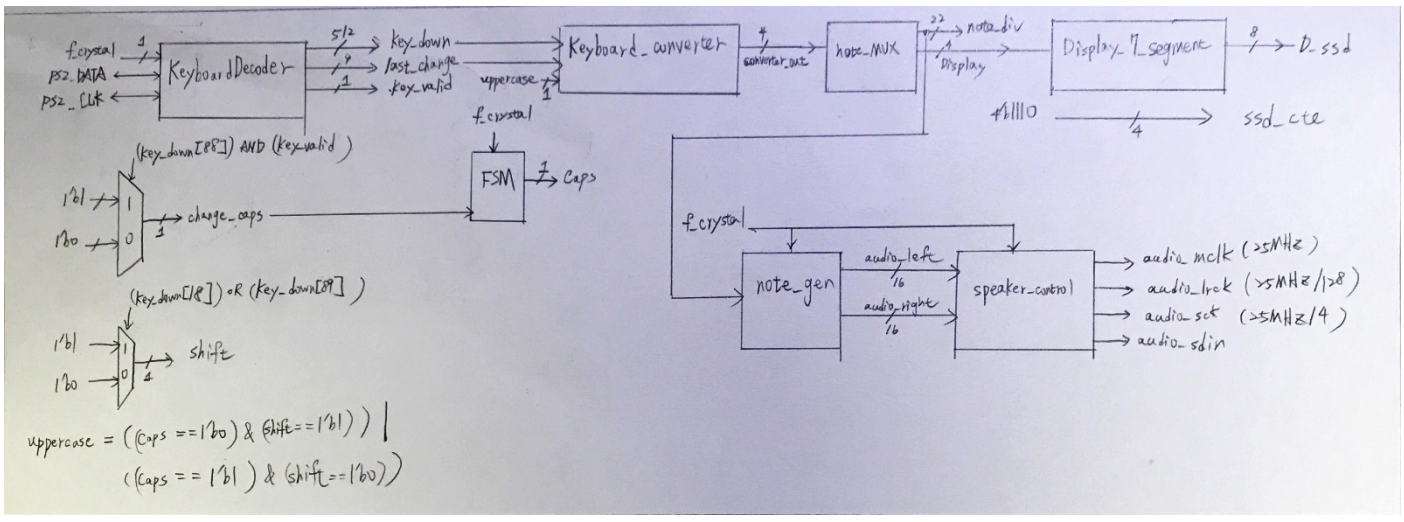
Design Specification:

Input: f_crystal, rst

Output: audio_mclk, audio_lrck, audio_sck, audio_sdin, Caps, uppercase, [7:0]D_ssd, [3:0]ssd_ctl

Inout: PS2_CLK, PS2_DATA

Block diagram:



Design Implementation:

I/O pin assignment:

f_crystal—W5

rst—V17

audio_mclk—A14

audio_lrck—A16
audio_sck—B15
audio_sdin—B16
PS2_CLK—C17
PS2_DATA—B17
uppercase—P1
ssd_ctl[3]—W4
ssd_ctl[2]—V4
ssd_ctl[1]—U4
ssd_ctl[0]—U2
D_ssd[7]—W7
D_ssd[6] — W6
D_ssd[5] — U8
D_ssd[4] — V8
D_ssd[3] — U5
D_ssd[2] — V5
D_ssd[1] — U7
D_ssd[0] — V7

As what we did in Lab9, I convert 9-bit signal last_change into a 4-bit signal converter_out. Also, I design a FSM to control “caps lock” signal and use key_down to check if shift is pressed so that I know when to output uppercase (higher pitch). For example, press “c” -> converter_out = 4’d1, press “A” -> converter_out = 4’d13. Also I used key_down to make sure only when the keyboard is pressed, it will output sound. I think it makes keyboard act more like a electronic piano.

Inside note_MUX, it will output different note_div based on the value of converter_out. And Display is basically equal to converter_out. But as we only use 1~7 to represent the note, if the converter_out is larger than 4’d7, I just subtract it by 7 (- 4’d7). Uppercase (LED) will light up if it’s higher octave note.

Finally, combining note_gen and speaker_control, we are able to hear different sounds of note when pressing keyboard.

Discussion:

This experiment would be relatively fast to complete as I just need to combine the functions of keyboard and speaker together. From the input of keyboard, I converted it into 4-bit number and used a MUX to output different sounds

3. (Bonus) Playback double tones by separate left and right channels. If you turn one DIP switch off, the electronic organ playback single tone when you press keyboard (as in Prob. 2). If you turn DIP switch on, left (right) channels play Do(Mi), Re(Fa), Mi(So), Fa(La), So(Si) when you press the keyboard the keyboard.

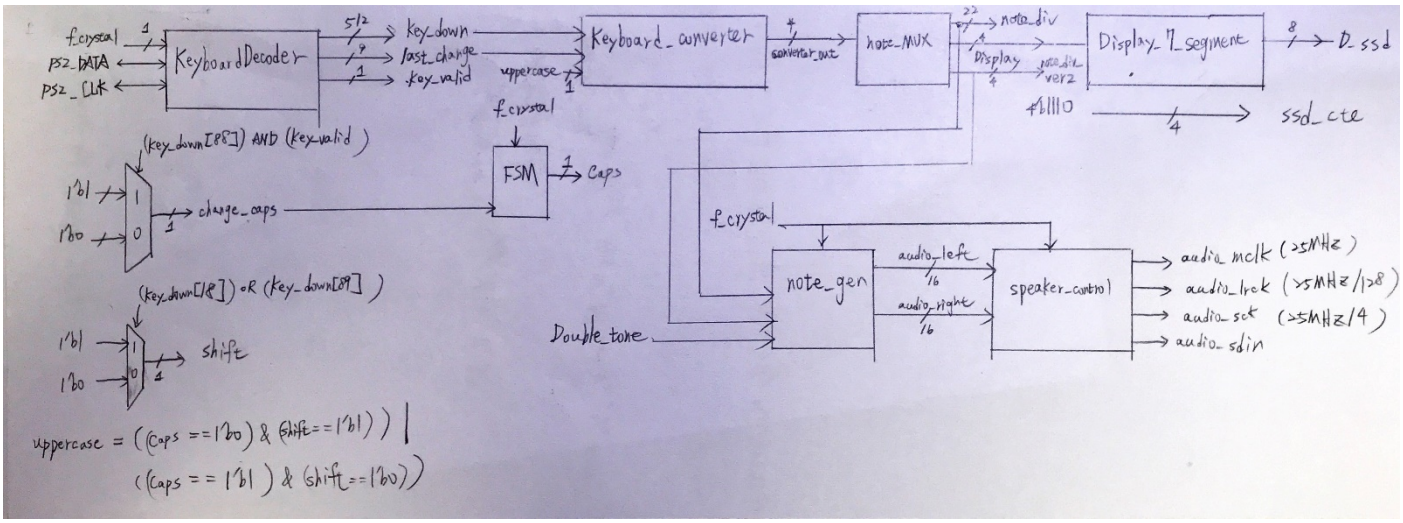
Design Specification:

Input: f_crystal, rst, Double_tone

Output: audio_mclk, audio_lrck, audio_sck, audio_sdin, Caps, uppercase, [7:0]D_ssd, [3:0]ssd_ctl

Inout: PS2_CLK, PS2_DATA

Block Diagram:



Design Implementation:

I/O pin assignment:

f_crystal—W5

rst—V17

audio_mclk—A14

audio_lrck—A16

audio_sck—B15

audio_sdin—B16

PS2_CLK—C17

PS2_DATA—B17

uppercase—P1

ssd_ctl[3]—W4

ssd_ctl[2]—V4

ssd_ctl[1]—U4

ssd_ctl[0]—U2

D_ssd[7]—W7

D_ssd[6] — W6

D_ssd[5] — U8

D_ssd[4] — V8

D_ssd[3] — U5

D_ssd[2] — V5

D_ssd[1] — U7

D_ssd[0] — V7

This experiment is quite similar to experiment #2. The most important part is how to create double tone. As double tone is to create different sound through audio_left and audio_right, I create the requested double tone in the problem in note_MUX module. For example, if note_div represents Do, then note_div_ver2 represents Mi. If note_div represents Fa, then note_div_ver2 represents La.

Both note_div and note_div_ver2 are inputs for note_gen module. Inside note_gen, it will create two different clock frequency based on note_div and note_div_ver2 respectively. Then if the value of Double_tone is 1'b1 (switch on), the left channel (左聲道) will use clock frequency from note_div_ver2. Otherwise, both left channel and right channel (右聲道) will use the same clock from note_div only.

And the experiment doesn't describe the double tone for La and Si, their note_div and note_div_ver2 are the same. In other words, there is no double tone for these two sounds.

And Other parts of the function are the same as the experiment #2.

Discussion:

I think I used the most straightforward way to implement the double tone. I made two sounds at the same time (ex: Do and Mi). Then I just need to use the value of Double_tone to decide if I need to output double tone.

Conclusion for Lab10:

Through this lab, I understand how to combine the function of keyboard and speaker so that it can be viewed as the electronic piano. We can even play 小蜜蜂 by pressing the corresponded keypads. Also, experiment #3 helps me understand the concept of double tone and how to implement it through Verilog. I just need to make left channel (左聲道) and right channel (右聲道) produce different sounds so that we are able to hear the double tone.