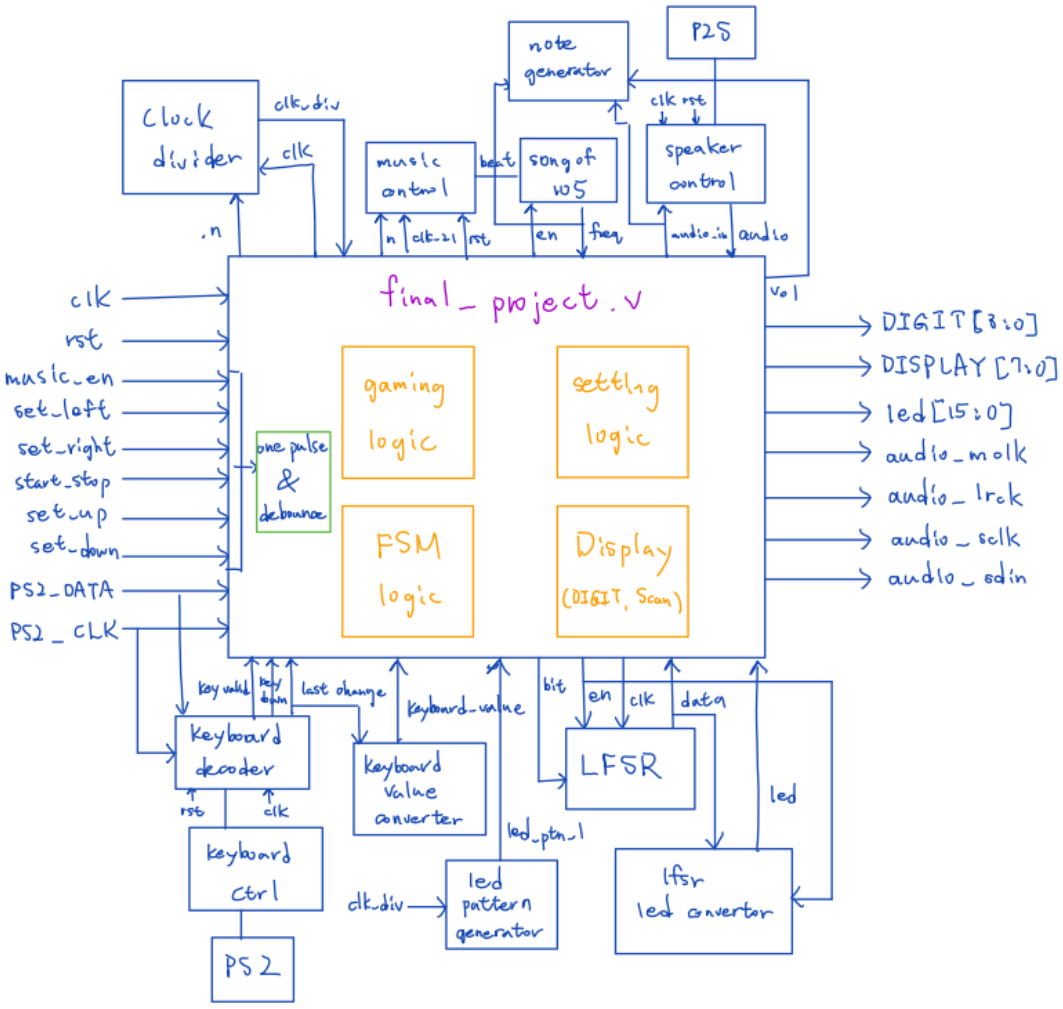
Final Project

Design Specification

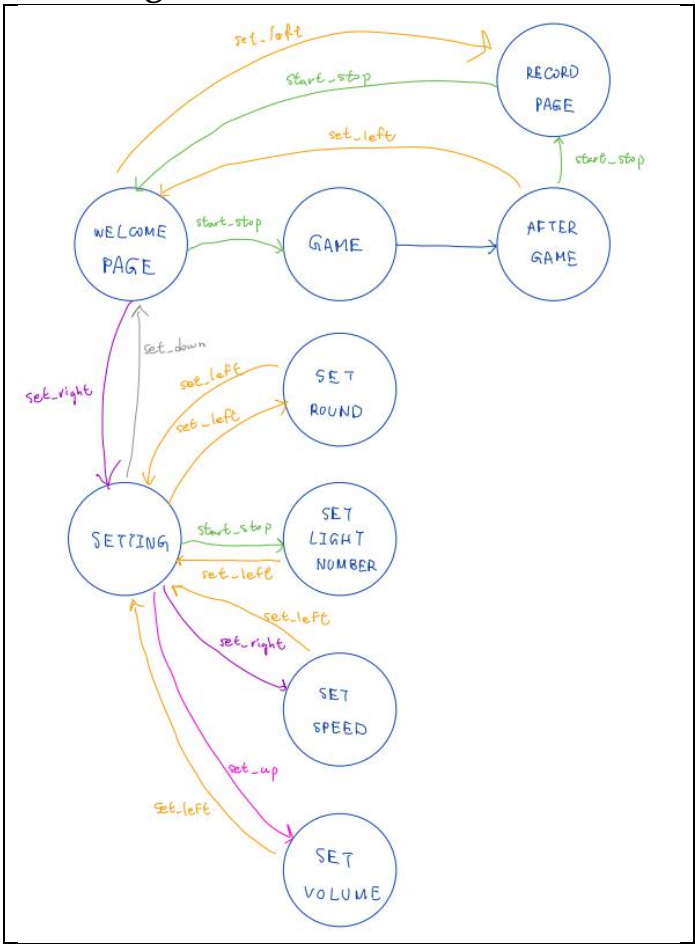
✓ Write down the specification of your design: inputs, outputs, and related bit widths. The name should be the same as what you use in the Verilog design files.
Whack-a-mole

- Input: clk, rst, music_en, PS2_DATA, PS2_CLK, set_left, set_right, set_up, set_down, start_stop
- Output: DIGIT [3:0], DISPLAY [7:0], led [15:0], audio_mclk, audio_lrck, audio_sclk, audio_sdin

✓ **Draw the block diagram of the design.**



✓ **FSM Diagram**



start_stop: U18
set_left: W19
set_right: T17
set_up: T18
set_down: U17

Design Implementation

✓ Derive the related logic functions ***and/or*** draw the related logic diagram.

- **final_project.v**
  This is the top module of the whole project. As you can see in the figure above, I derived some modules from this file, though some main functions I still keep in the top module, which are game control, setting block, FSM block (state reg, next state logic, and output logic), and display block. I'll explain game control and setting block in detail.
  - Game control
    In game control, it decides the led output. If the current state is in GAME, then the led signal is connected with lfsr_led_convertor. If the current state is in AFTER_GAME, the led signal is connected with led_pattern_generator_1. Otherwise, it will show curr_state in the convenience of debugging. Plus, game control also counts the number of rounds, and the points, and records the highest score. Note that the point-counting logic is also based on how fast the LEDs are showing (this is mentioned in the final project proposal), the faster the clock is, the higher points you get.

- Setting block
  In the proposal, we can set the number of rounds, clock speed, the number of LEDs lightened as well as the song playing. However, though I only finished one music file, the song selection function has been replaced with a volume setting. In the Setting block, there's a flip flop that controls the buttons' input and the corresponding parameters just like lab6.

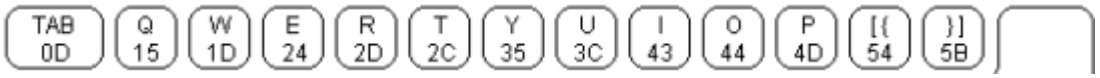
Next, I'll derive the rest into several parts according to the structural diagram below.

- Final Project (top module)
  |--- Clock_divider(for several uses)
  |--- music_control
  |--- SongOf105
  |--- speaker_control
  | |--- P2S
  |--- note_gen
  |--- clk_divider (for the time of every game round)
  |--- KeyboardDecoder
  | |--- KeyboardCtrl
  | |--- Ps2Interface
  |--- keyboard_value_converter
  |--- led_pattern_generator_1
  |--- debounce
  |--- one_pulse
  |--- LFSR
  |--- lfsr_led_convertor

- **Clock_divider.v (for several uses)**
  Aside from the clock for the game round, the project still needs other clock signals. This module is responsible for generating the clock for music control, the neon frequency that appears after the game, and the clock for 7-segment displays. This module can take a variable in it just like other high-level programming languages so that we can reduce the number of modules, but generates several different clock frequencies at the same time.
- **music_control.v**
  This module generates the beat for the music. The beat will affect the tone that SongOf105.v plays. If the music playing function isn't enabled, the beat will be 0 thus preventing SongOf105.v from playing the music.
- **SongOf105.v**
  The left and right frequencies have eight bars (小節) respectively, and 16 beatNums are used to represent a bar, and there are 512 beatnums in total. Control the tempo of clk so that the tempo of beatnum from 0 to 511 matches the tempo of the song. Use cases from 0 to 511, and concatenate each tone to form a song.

© Ma, Hsi-Pin, Lab for Reliable Computing (LaRC), EE, NTHU.

- **speaker_control.v and P2S.v**
  The same as lab7.
- **note_gen.v**
  This module generates the notes of the song, the volume is also controlled here.
- **clk_divider (for the time of every game round)**
  This module generates one second, 1.6 seconds, and 2.2 seconds according to the set of the speed of the moles appearing.
- **KeyboardDecoder.v, KeyboardCtrl.v, and Ps2Interface.v**
  The same as lab7(provided in lecture attachments)
- **keyboard_value_converter.v**
  This module detects keyboard activity and outputs keyboard_value to the game control logic in final_project.v . In the final project, we only use this row on the keyboard to indicate the position of mole(s).



Since we use 12 LEDs (LD1~LD12) in the game to indicate the position of mole(s), there are also 12 keys (Q, W……. [,]) that correspond to the LEDs. For example, if LD12 lightened, then the user can earn points by pressing Q. If LD1 lightened, presses] can let the user earn points. Thus, with the keyboard value, we can detect whether the user presses the correct key or not.

- **led_pattern_generator_1.v**
  This module generates the led signal that appears after the game. It looks like neon lighting, to celebrate the completion of the game.
- **debounce.v, one_pulse.v**
  the same as previous labs.
- **LFSR.v**
  LFSR (Linear-feedback shift register) is a special type of shift register whose input depends on its previous state. By taking advantage of LFSR, we can stimulate generating random numbers. The output of this module will be connected to lfsr_led_convertor.v before showing its led result.
- **lfsr_led_convertor.v**
  The result we get from LFSR.v is completely raw. First, it generates a 4-bit number in each cycle, in other words, the number ranges from 0 to 15. However, we only use 12 LEDs, which means only 1-12 can indicate the position of the LED lightened. Second, since we can set the number of led lights, if we want three LEDs lightened at the same time, for example, instead of connecting three LFSR modules with the LEDs, I'd rather only use one LFSR module. This is because we can't guarantee every time the three numbers generated by the LFSR modules are unique, if two of the numbers are the same, then there will only be two LEDs lightened, but not three. This contradicts the result we want. In conclusion, lfsr_led_convertor.v is responsible for processing the raw data from the LFSR module before generating the led signal we want.

© Ma, Hsi-Pin, Lab for Reliable Computing (LaRC), EE, NTHU.

✓ **List the I/O pin assignment for your design.**

- Clock signal

I/O	clk
LOC	W5

- Buttons

I/O	start_stop	set_up	set_left	set_right	set_down
LOC	U18	T18	W19	T17	U17

- Switches

I/O	rst	music_en
LOC	V17	V16

- LEDs

I/O	led[0]	led [1]	led [2]	led [3]	led [4]	led [5]	led [6]	led [7]
LOC	U16	E19	U19	V19	W18	U15	U14	V14

I/O	led [8]	led [9]	led [10]	led [11]	led[12]	led[13]	led[14]	led[15]
LOC	V13	V3	W3	U3	P3	N3	P1	L1

- 7 segment display

I/O	DISPLAY[0]	DISPLAY[1]	DISPLAY[2]	DISPLAY[3]	DISPLAY[4]	DISPLAY[5]	DISPLAY[6]	DISPLAY[7]
LOC	V7	U7	V5	U5	V8	U8	W6	W7

I/O	DIGIT[0]	DIGIT[1]	DIGIT[2]	DIGIT[3]
LOC	U2	U4	V4	W4

- Pmod Header JB

I/O	audio_mclk	audio_lrck	audio_sclk	audio_sdin
LOC	A14	A16	B15	B16

- USB HID (PS/2)

I/O	PS2_CLK	PS2_DATA
LOC	C17	B17

Discussion

✓ **Result Analysis**

In this project, I successfully implemented a Whac-A-Mole game using an FPGA board and Verilog. It takes the button input for the control settings and takes the keyboard input for gameplay control. The output is the LED signal as the position of moles and the seven-segment display to display the score and setting figures. I also use an audio extension module to play background music, users can set the volume as well.

In my opinion, the key point of this project is the introduction of LFSR, which generates random-like numbers. However, the raw data requires further processing as I mentioned above. If this function can be implemented successfully, then there would be little problem with the whole project.

© Ma, Hsi-Pin, Lab for Reliable Computing (LaRC), EE, NTHU.

- ✓ **Error Analysis**

  There wasn't any huge error during the implementation of my final project, despite some little bugs. Nevertheless, when I was about to wind up the whole project, it came to my mind that maybe it would be more close to reality if the LED will darken after the user click the correct key, just like the mole would disappear after someone hit it with a hammer. This was quite annoying because a bunch of code blocks needed to be added to the lfsr_led_convertor to decide the 12 LEDs, which will keep lightened, which will not. In the end, I tried to implement this function but failed. Thus, the gameplay outcome remains the same, players will get points whenever they hit the correct spot of the LED(s).

- ✓ **Thoughts/Comments/Suggestions**

  Since I have done some bigger coding projects, this final project wasn't too complicated for me from a technical perspective. What we only need is a detailed blueprint and to be mindful while working with the code. Plus, it is important to set checkpoints often, and test the function you have just implemented instead of debugging after a long time of work.

  There are some points that I hope I can do better next time. First, is the darken function I mentioned above. Second, lfsr_led_convertor requires a better design. Even though the function works fine, I hope the coding can be cleaner and reduce duplicate code blocks if possible. Verilog is a more rigorous programming language compared to C++ or python, I didn't want to make any mistakes so I literally table-matching the LED signal. This makes the module even longer than my top module. I hope I can do better with that. Third, I gave up using the VGA display when I was planning my final project. I think VGA is the most interesting part of this lecture, and it expands the possibility and creativity of the project. Unfortunately, I predicted that I would be super busy at the end of the semester because of the five main courses I take, and I'm a dumb kid as well, so I was not sure whether I'm going to implement the desired function wanted.

  Aside from the joy of successfully meeting all the requirements on the proposal, I was very proud that I put the black ground music into practice. In comparison to lab7, the music function in this project is more than a hundred times more complicated. Instead of a single tone in lab7, I have to take the tones, speed, and even the main theme, and accompanying theme into consideration. There are about five hundred lines just to take care of the different tones of the song. Even though I'm not able to finish more than one song, I'm still very proud of the work. There's a link below to the background music I simulate.

Conclusion

As the final assignment for this class, we made a game of Whac-A-Mole through the FPGA board and Verilog implementation. It is the ultimate combination of all we learned throughout the semester, and I used everything except the VGA display due to technical difficulties. Even though there are some defects in the overall design, I still give high credit for my contribution. I'm proud of myself for doing this alone. Of course, there were burdens, but I eventually find a way out.

In conclusion, I had a deeper understanding of the basics of hardware implementation

© Ma, Hsi-Pin, Lab for Reliable Computing (LaRC), EE, NTHU.

and am ready for harder tasks in the future. I would like to seize this chance as well to thank the professor and TAs for all you have done.

References

1. FPGA产生基于LFSR的伪随机数
   https://blog.csdn.net/kebu12345678/article/details/54834763
   This is the website I used to understand how LFSR works and how to implement it into my project
2. 阿肆 - 熱愛105℃的你【動態歌詞】「Super Idol的笑容 都沒你的甜」
   https://www.youtube.com/watch?v=iJAxeafvn8Y
   the source of the background music