Final Project _ group29

Final_project snake game

Design Specification

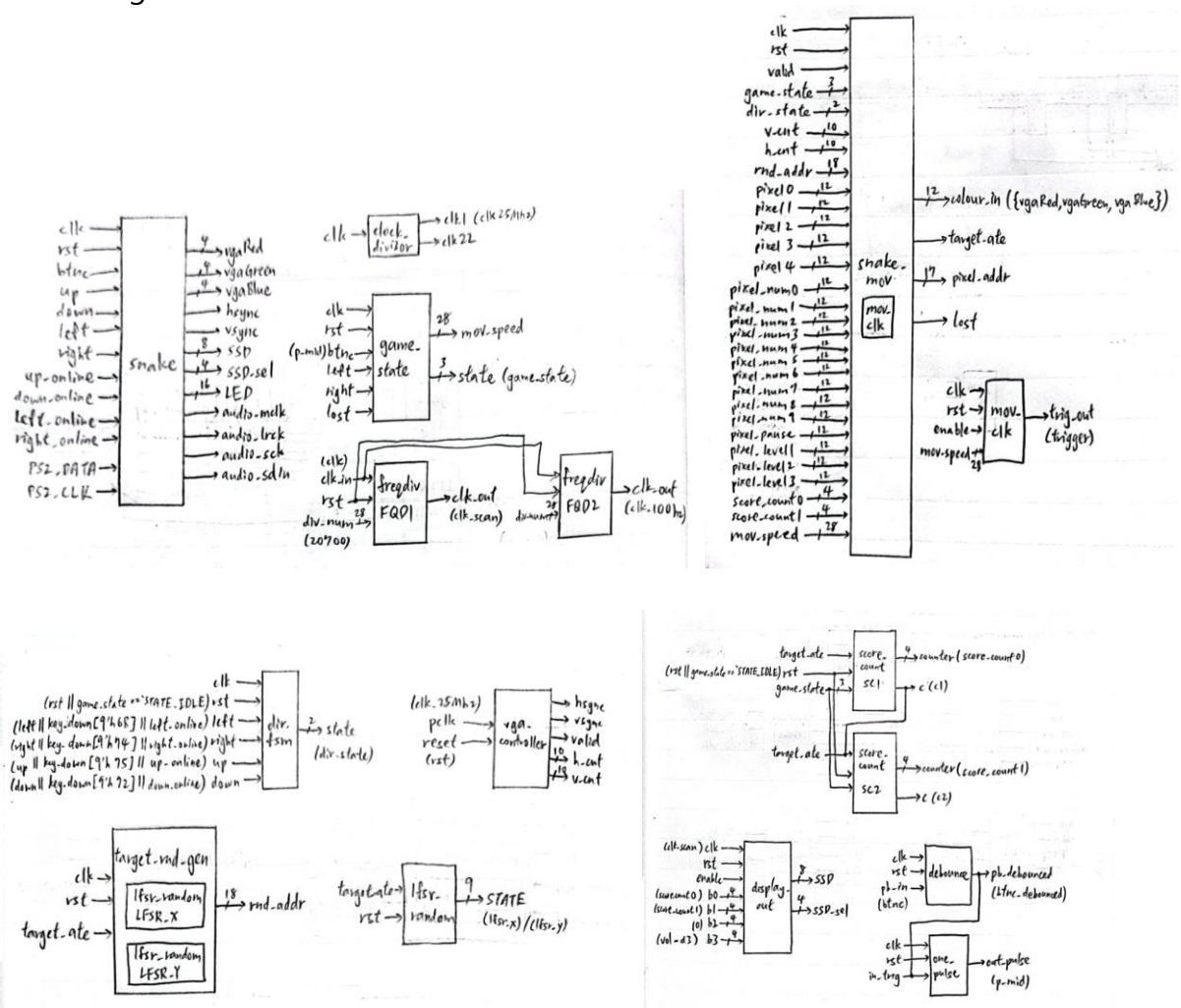
IO:

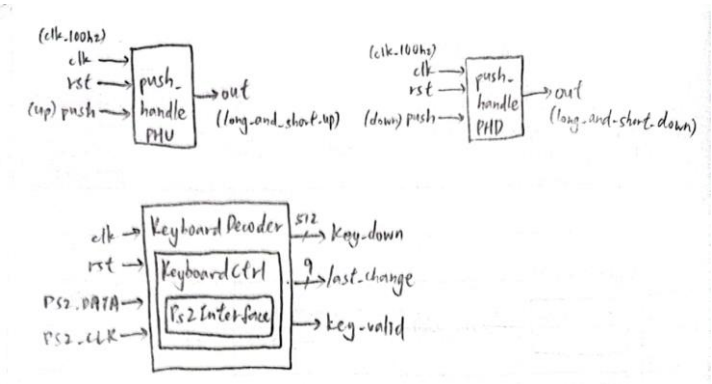
Input: clk, rst, btnc, left, right, up, down, up_online, down_online, left_online, right_online

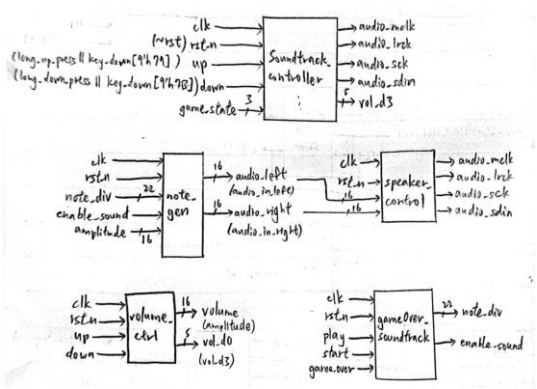
Inout: PS2_DATA, PS2_CLK

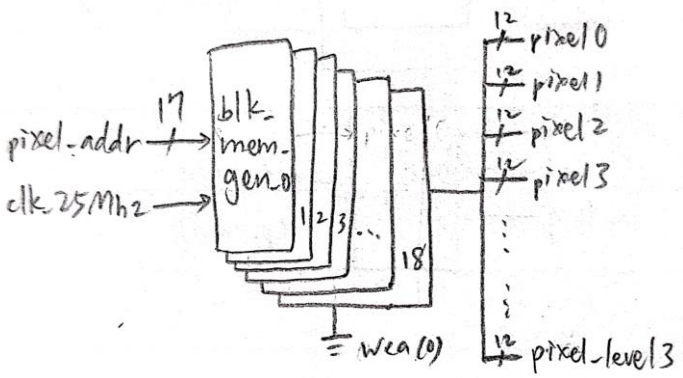
Output: vgaRed[3:0], vgaGreen[3:0], vgaBlue[3:0], hsync, vsync, SSD[7:0], SSD_sel[3:0], audio_mclk, audio_lrck, audio_sck, audio_sdin

Block Diagram:



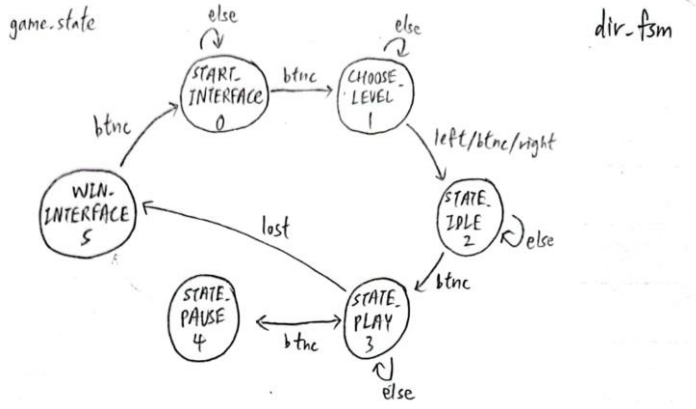


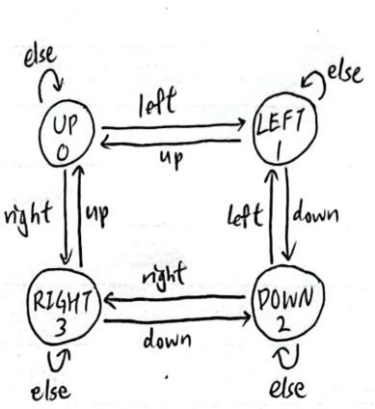




Design Implementation

Logic diagram:





Discussion

1. Firstly, we implemented an FSM (game_state) to choose level/start/pause/resume the game. We use left, btnc and right push buttons to choose level1, level2 or level3, respectively. After choosing the desired game level, we will enter STATE_IDLE and we can start the game when we press btnc again. When we are in playing mode, we can press btnc to pause and resume the game. When the snake head hit the boundary or its body, the game is over and we will be at the last state STATE_WIN_INTERFACE (which represents game over). Then if the btnc is pressed, we will be back to the first state STATE_START_INTERFACE, and we can press the btnc again to choose the levels and

start playing the game all over again.

2. The three different levels represent the different moving speed of the snake. We achieve this by implementing a simple frequency divider with a mov_speed input. When we press the left push button, mov_speed is 50 000 000(divide 100Mhz by 2), which means the counter will toggle when it reaches this number, so it can generates a 1Hz clock frequency, this let the snake to have a 1Hz moving speed; when btnc is pressed, mov_speed is 25 000 000(divide 100Mhz by 4), which can generates a 2Hz clock frequency, this let the snake to have a 2Hz moving speed; lastly when right push button is pressed, mov_speed is 12 500 000(divide 100Mhz by 4), which a 4Hz clock frequency can be genearted, so the snake will have a 4Hz moving speed.
3. To make the snake move, we need to shift the snake position by 20x20 pixel on every trigger. We use a generate block to create multiple instances of a similar piece of logic in order to manage the shifting of the snake's body segments, ensuring that each segment follows the segment in front of it.
4. We need another FSM (dir_fsm) to control the direction of movement of the snake. We used the push buttons up, down, left, right to control the snake. When the snake is moving upwards and downwards, SNAKE_Y will be SNAKE_Y-20 and SNAKE_Y+20 respectively, while SNAKE_X remains the same; when the snake is moving leftwards and rightwards, SNAKE_X will be SNAKE_X-20 and SNAKE_X+20 respectively, while SNAKE_Y remains the same. When the game is paused, the snake position remains the same, which means both SNAKE_X and SNAKE_Y will not have any changes.
5. Since we are taking input signals from push buttons, supposedly we need to have debounce circuit for each button, but there will not be a difference despite the duration we pressed the up, down, left, right button, so we only use a debounce circuit and one pulse for the btnc to change the game states. Moreover, since we need long and short press signals from the up, down push buttons, we use push_handle (which has a counter in it) to detect the timing of the buttons being pressed. The short press signals of up and down button are used to control the snake to move up and down, while the long press signals are used to control the volume of the music, long press up to increase and long press down to decrease the volume.
6. Other than using the push buttons as direction and volume control, we also implemented the keypad 8, 2, 4, 6 to control the direction of movement up, down, left, right, and +, - to increase and decrease the volume. We just need to import the given modules KeyboardDecoder, KeyboardCtrl and Ps2Interface, and use the key_down signals from the module to achieve the keyboard functions.
7. For the volume control part, we just use the modules of previously done lab, lab7_2.
8. To play music when we are playing the game, we first search the music sheets online and define all the notes' frequency and rests' frequency we need according to the music sheets. Then, we write down the notes and rests from the music sheets and use a up counter (note_index=note_index+1) to play the notes in order. To repeat the ordered notes, we use another down counter that resets when it reaches 0, this is when the note_index up counter will start from 0 again. We use a total of 3 different music for different situations (one for before starting the game, one for when playing the game, and one for game over).

9. Now for the main part of our design, we first set the initial snake length as 3, and every time the snake collides with the food, which means the position of snake head is the same as target, the score will +1 and the length of snake +1 either.
10. For the score counting part, we use two decimal up counter to achieve a 2 digits decimal up counter. The first counter counts from 0 to 9 and the carry==1 for the second counter to start counting. The counters will reset in STATE_IDLE or the DIP switch for reset (we can see this as the power button of the game machine) is turned on (since we use a posedge rst).
11. For the collisions, whenever the snake position is out of the game space we defined, it means the snake hits the boundary; whenever the snake head position is the same as any segments of its body, it means the snake hits its own body. Both situations will lead to game over, and all the LEDs will light up.
12. To randomly generate the food, we use a 9-bit LFSR, using XOR of the fourth and ninth bit as the input to the shift register. Then, we output the state as random position of the x-axis and y-axis, we only output this random position when the food is eaten (target_ate==1).
13. The SSD display part is just the same as what we have done for almost the whole semester in majority of labs since lab2, so I omitted the explanation for it here. We use the left most digit to display the volume, the right most 2 digits to display the score.
14. In addition, we also added the input from JXADC1, JXADC2, JXADC3 and JXADC4 as up, down, left, right direction control, respectively.
15. For the VGA display, we define where we want the images or elements to be:

Images and elements	Positions
Snake on start interface	X ∈ [0,640], Y ∈ [0,480]
Game over on win interface	X ∈ [0,640], Y ∈ [0,480]
Snake head	X ∈ [crt_pos,crt_pos+20], Y ∈ [crt_pos,crt_pos+20]
Snake body	X ∈ [crt_x[i+1],crt_x[i+1]+20), Y ∈ [crt_y[i+1],crt_y[i+1]+20), for i<snake_length
Apple (target)	X ∈ [lfsr_pos,crt_pos+20], Y ∈ [lfsr_pos,crt_pos+20]
Score	X ∈ (464,615), Y ∈ (100,200]
Score0 (0~9)	X ∈ (539,616), Y ∈ (201,301]
Score1 (0~9)	X ∈ (464,540), Y ∈ (201,301]
Pause	X ∈ [155,305), Y ∈ [170,270)
Level 1	X ∈ [20,175), Y ∈ [20,420)
Level 2	X ∈ [225,375), Y ∈ [20,420)
Level 3	X ∈ [425,575), Y ∈ [20,420)
out of the playing space	X<40, X>440, Y<20, Y>460

16. After implementing all the modules in Verilog, it will then be synthesized and implemented. Lastly, bitstream can be generated and it can be programmed to the FPGA board and with the following parameters:

I/O	clk	rst	btnc	left	right	up	down	
site	W5	R2	U18	W19	T17	T18	U17	
I/O	vgaRed3	vgaRed2	vgaRed1	vgaRed0	vgaGreen3	vgaGreen2	vgaGreen1	vgaGreen0
site	N19	J19	H19	G19	D17	G17	H17	J17

I/O	vgaBlue3	vgaBlue2	vgaBlue1	vgaBlue0	hsync	vsync	PS2_DATA	PS2_CLK
site	J18	K18	L18	N18	P19	R19	B17	C17
I/O	SSD7	SSD6	SSD5	SSD4	SSD3	SSD2	SSD1	SSD0
site	W7	W6	U8	V8	U5	V5	U7	V7
I/O	SSD_sel3	SSD_sel2	SSD_sel1	SSD_sel0	audio_mclk	audio_lrck	audio_sck	audio_sdin
site	W4	V4	U4	U2	A14	A16	B15	B16
I/O	LED15	LED14	LED13	LED12	LED11	LED10	LED9	LED8
site	L1	P1	N3	P3	U3	W3	V3	V13
I/O	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0
site	V14	U14	U15	W18	V19	U19	E19	U16
I/O	up_online	down_online	left_online	right_oline				
site	J3	L3	M2	N2				

Then, we can connect an earphone to Pmod I2S2, a PS2 keyboard, a LCD monitor with VGA cable as well as ESP-32 board and verify the result on FPGA board.

Conclusion

I. Challenges and Solutions

1. At first, we have no clue on how to define the snake position, snake head, snake body etc. and spend a lot of time figuring how to set up all the elements. Then we search for information online and found out that it is much easier to use the X-axis and Y-axis alike idea to define the positions of our playing space and game elements.
2. The challenge we face is how to define the chosen cut down image to our desired range in the display area with normal size, which is the most complex part for VGA display. After countless time of calculating and trying to display on the monitor, we finally get all the positions correct.
3. The second challenge is to complete all the game states without any error, which we thought it shouldn't consume too much time. We initially use only 2-bits for our game state FSM, but we added more states in the latest version which required 3-bits. And due to our carelessness, we often neglect the bits of states in various modules, which make the change of states to be confusing and incomplete, which leads to long time debugging.

II. Unsolvable Bugs

1. There are a few bugs that we aren't able to solve. The first is the randomly generated target position by the LFSR will mostly appear on upper left corner. We wonder if it is because LFSR does not truly has a random output but pseudorandom.
2. The second bug is when the snake collides with the first target and the snake length should +1, there will be a block of the snake body's color appears in a blink on the upper left corner. We tried to solve this but we could not define where the problem is.
3. The third bug is the pause image could not cover up the snake head and apple image, so when these images are in the range of the pause image, there will be a blank block which cause the incomplete display of the pause image. However, this is not the case for the snake body, the pause image can cover the snake body and the full image can be displayed.

III. Thoughts and Reflections

1. For the additional component, since we used JXADC as input ports, if we did not connect the ESP-32 board, BASYS3 board will get interference from the noise, which leads to unstable input signals for the direction control. So, the better way to avoid this situation is to change our ADC input ports to JC input ports.
2. Through this project, we have a lot of practice on how to define the boundaries of various image to be display on the monitor, which allow us to be more familiar with and have deeper understanding on the VGA display logics. Other than the VGA part, we learn to play music using the speaker, and revise all the use of keyboard, SSD display, LEDs and DIP switches. Moreover, we also studied on our own about how to import the signal from breadboard to FPGA board.
3. Lastly, to complete this final project, we have the opportunities to admire the beautiful sunrise this semester.

Other modules/daughter boards (optional)

1. ESP-32 board
2. Dupont wire
3. 200Ω resisters

Division of labor & cooperation

Proposal	顏維萱，王籽穎
Speaker	王籽穎，顏維萱
Keyboard	顏維萱
Snake game logic	王籽穎
Length increasement of snake	王籽穎，顏維萱
FSMs	顏維萱，王籽穎
Debug	王籽穎，顏維萱
Additional modules (ESP-32 board)	王籽穎
Report	顏維萱，王籽穎

References

GitHub – spencer-tb/snake-game-fpga
https://github.com/spencer-tb/snake-game-fpga

Musescore – Pacman
https://musescore.com/user/10673311/scores/2827506

Musescore – Super Mario Bros.: Main Theme
https://musescore.com/user/27687306/scores/4913846

Musescore – Game Over-Super Mario Bros.
https://musescore.com/user/33605901/scores/10164289

Additional References

籽穎 GitHub
https://github.com/wongzinc/VerilogSnakeGame/tree/main
We will/might upload our codes and logic diagrams on GitHub after this final exam week, when we have leisure time.