Final Project: Snake

Design Specification

To design a snake game, we have to think of what functions should the program be equipped with, for example, the score counting function, time counting function.　More importantly, when will the game be ended?　Otherwise, the game won't stop and fall into infinite loop, so it is vital for us to design the logic statements well.

Additionally, we can combine the program with other peripheral applications, such as keyboard and speaker.　Consequently, not only we apply all we have learned this semester, but it could optimize our program.

Clock divisor

Input:

clk (the original input frequency signal, f = 25MHz)

Output:

clk_1(the processed frequency signal, f = 1Hz)

Frequency division part

Input:

clk (the original input frequency signal, f = 100MHz)

rst (used to reset the status)

Output:

[1:0] clk_ctl (used to control the signal at scan control module)

clk_1 (the processed frequency signal, f = 1Hz)

clk_100 (the processed frequency signal, f = 100Hz)

clk_25 (the processed frequency signal, f = 25Hz)

clk_12_5 (the processed frequency signal, f = 12.5Hz)

debounce part

Input:

clk (input frequency signal with frequency of 100MHz)

rst (used to reset the module, and it is controlled by DIP switch at V17)

pb_in (input signal of button)

Output:

pb_debounced (debounced signal)

One_pulse

Input:

clk (input frequency signal with frequency of 100MHz)

rst (used to reset the module, and it is controlled by DIP switch at V17)

in_trig (input signal from button, and it is from debounce module)

Output:

out_pulse (output one-pulse signal)

fsm direction

Input:

clk (input frequency signal with frequency of 100MHz)
rst (used to reset the module, and it is controlled by DIP switch at V17)
one_up (one pulse signal, used to make snake move upward)
one_down (one pulse signal, used to make snake move downward)
one_left (one pulse signal, used to make snake move left)
one_right (one pulse signal, used to make snake move right)
one_start (one pulse signal, used to start the game)
gameover (used to change the status called "fail")
clk_1 (the processed frequency signal, f = 1Hz)
opposite (used to control the states, which are determined by the drug taken or not)

Output:
[2:0] direction (state name, used to control the direction of movement)
gaming (indicate whether we are playing game or not)
score_zero (used to reset the score)
LED_play (indicate that we lose the game, it will flicker when game is ended)(f = 1Hz)
[1:0] state2 (state name)

VGA control

Input:
pclk (scanning clock)
reset (reset the pixel counting)

Output:
hsync (video synchronization in horizontal direction)
vsync (video synchronization in vertical direction)
valid (determine the valid scanning area)
[9:0] h_cnt (scan the image in horizontal direction)
[9:0] v_cnt (scan the image in vertical direction)

Pixel generate

Input:
[9:0] h_cnt (scan the image in horizontal direction)
[9:0] v_cnt (scan the image in vertical direction)
[2:0] direction (state name, used to control the direction of movement)
gaming (indicate whether we are playing game)
clk (input frequency signal with frequency of 100MHz)
rst (used to reset the module, and it is controlled by DIP switch at V17)
[4:0] apple_x (indicate the position of apple in x-axis)
[4:0] apple_y (indicate the position of apple in y-axis)
[4:0] drug_x (indicate the position of drug in x-axis)
[4:0] drug_y (indicate the position of drug in y-axis)
valid (determine the valid scanning area)

Output:
[3:0] vgaRed (used to control a component for the final color displayed at the VGA board)
[3:0] vgaGreen (used to control a component for the final color displayed at the VGA board)
[3:0] vgaBlue (used to control a component for the final color displayed at the VGA board)
gameover (used to change the status called "fail")
apple_valid (when eating the apple, it will turn to 1. Afterward, it will do two things: first, it will be used to indicate that the module should generate a new apple. Second, it will be used to indicate that the snake should extend for a segment)

random apple generator

Input:

clk (input frequency signal with frequency of 100MHz)
rst (used to reset the module, and it is controlled by DIP switch at V17)
apple_valid (when eating the apple, it will turn to 1. Afterward, it will do two things: first, it will be used to indicate that the module should generate a new apple. Second, it will be used to indicate that the snake should extend for a segment)
one_start (the input signal from the center button, and it has been processed by debounce and one pulse modules. It can used to start or restart the game)

Output:

[4:0] apple_x (indicate the position of apple in x-axis)
[4:0] apple_y (indicate the position of apple in y-axis)

Time of playing

Input:

gaming (indicate whether we are playing game)
score_zero (used to reset the score)
clk (input frequency signal with frequency of 100MHz)
rst (used to reset the module, and it is controlled by DIP switch at V17)

Output:

[3:0] digit3, [3:0] digit2, [3:0] digit1, [3:0] digit0 (used to control the four digits of seven segment display)

One bit upcounter

Input:

clk (input frequency signal with frequency of 100MHz)
rst (used to reset the module, and it is controlled by DIP switch at V17)
increase (increase enable)
limit (upper bound of counting)
gaming (indicate whether we are playing game)
score_zero (used to reset the score)

Output:

value (counter value)
carry (carry value)

score

Input:

clk_mode (the frequency signal determined by the activation of speedup mode)
rst (used to reset the module, and it is controlled by DIP switch at V17)
apple_valid (when eating the apple, it will turn to 1. Afterward, it will do two things: first, it will be used to indicate that the module should generate a new apple. Second, it will be used to indicate that the snake should extend for a segment)
score_zero (used to reset the score)

Output:

[3:0] score_digit3, [3:0] score_digit2, [3:0] score_digit1, [3:0] score_digit0 (used to store the temporary value of counting)

random drug generator

Input:

clk (input frequency signal with frequency of 100MHz)

rst (used to reset the module, and it is controlled by DIP switch at V17)
drug_valid (used to control the drug effect, that is, if drug_valid == 1, then the moving direction would be altered)
one_start (the input signal from the center button, and it has been processed by debounce and one pulse modules. It can be used to start or restart the game)

Output:
[4:0] drug_x (indicate the position of drug in x-axis)
[4:0] drug_y (indicate the position of drug in y-axis)

Top speaker

Input:
clk (the input frequency signal, f = 100MHz)
rst_n (used to reset the status)
up (used to increase the level of volume)
down (used to decrease the level of volume)
gaming (indicate whether we are playing game)

Output:
audio_lrck (control the sequence of the serial stereo output)
audio_mclk (synchronizes the audio data transmission)
audio_sck (control the shifting of data into the input)
audio_sdin (receiving the processed serial signal from input)
[7:0] segs (used the value at "ssd_in" to represent different image at the seven segment display)
[3:0] ssd_stl (used to enable seven segment display to be on or off)

speaker part

Input:
clk (the input frequency signal, f = 100MHz)
rst_n (used to reset the status)
[21:0] note_div (the value which is reponsible to control the frequency of output voice, and it is from "note_control" module)
[15:0] audio_max (the outut signal to control the maximum volume of the voice, and it is from "volume_control" part)
[15:0] audio_min (the outut signal to control the minimum volume of the voice, and it is from "volume_control" part)

Output:
audio_lrck (control the sequence of the serial stereo output)
audio_mclk (synchronizes the audio data transmission)
audio_sck (control the shifting of data into the input)
audio_sdin (receiving the processed serial signal from input)

Frequency division part

Input:
clk (the original input frequency signal, f = 100MHz)
rst_n (used to reset the status)

Output:
[1:0] clk_ctl (used to control the signal at scan control module)
clk_1 (the processed frequency signal, f = 1Hz, but this signal won't be used in this experiment)
clk_5 (the processed frequency signal, f = 5Hz, and this signal will be sent to up_and_down

counter module to display the level of volume)

volume_control part
Input:
[3:0] volume_amplitude (the input signal and will be sent to buzzer_control module to control the volume of the voice)
Output:
[15:0] audio_max (the outut signal to control the maximum volume of the voice)
[15:0] audio_min (the outut signal to control the minimum volume of the voice)

Buzzer_control part
Input:
clk (the input frequency signal, f = 100MHz)
rst_n (used to reset the status)
[21:0] note_div (the value which is reponsible to control the frequency of output voice, and it is from "note_control" module)
[15:0] audio_max (the outut signal to control the maximum volume of the voice, and it is from "volume_control" part)
[15:0] audio_min (the outut signal to control the minimum volume of the voice, and it is from "volume_control" part)
Output:
[15:0] audio_left (the output voice signal, and it will be sent to the left earphone)
[15:0] audio_right (the output voice signal, and it will be sent to the right earphone)

Up and down counter
Input:
clk (the frequency signal, and it will be clk_5 in top module)
rst (used to reset the status)
increase (used to increase the level of volume)
decrease (used to decrease the level of volume)
Output:
digit0 (the number used to store the level of volume)

Keyboard decoder part
Input:
clk (the original input frequency signal, f = 100MHz)
rst (used to reset the status)
Output:
[511:0] key_down (control the buttons of the keyboard)
[8:0] last_change (composed of extend code and make code)
key_valid (used to indicate the press and the release action to the buttons)
Inout:
PS2_DATA (input/output signal from the keyboard port)
PS2_CLK (input/output signal from the keyboard port)

scan control part
Input:
[3:0] in3, [3:0] in2, [3:0] in1, [3:0] in0 (used to control the four digits of seven segment display)
rst_n (used to reset the status)

[1:0] clk_ctl_en (used to control the display of value)

Output:

[3:0] sse_in (used to control the four digits of seven segment display)

[3:0] ssd_ctl (used to enable seven segment display to be on or off)

display part

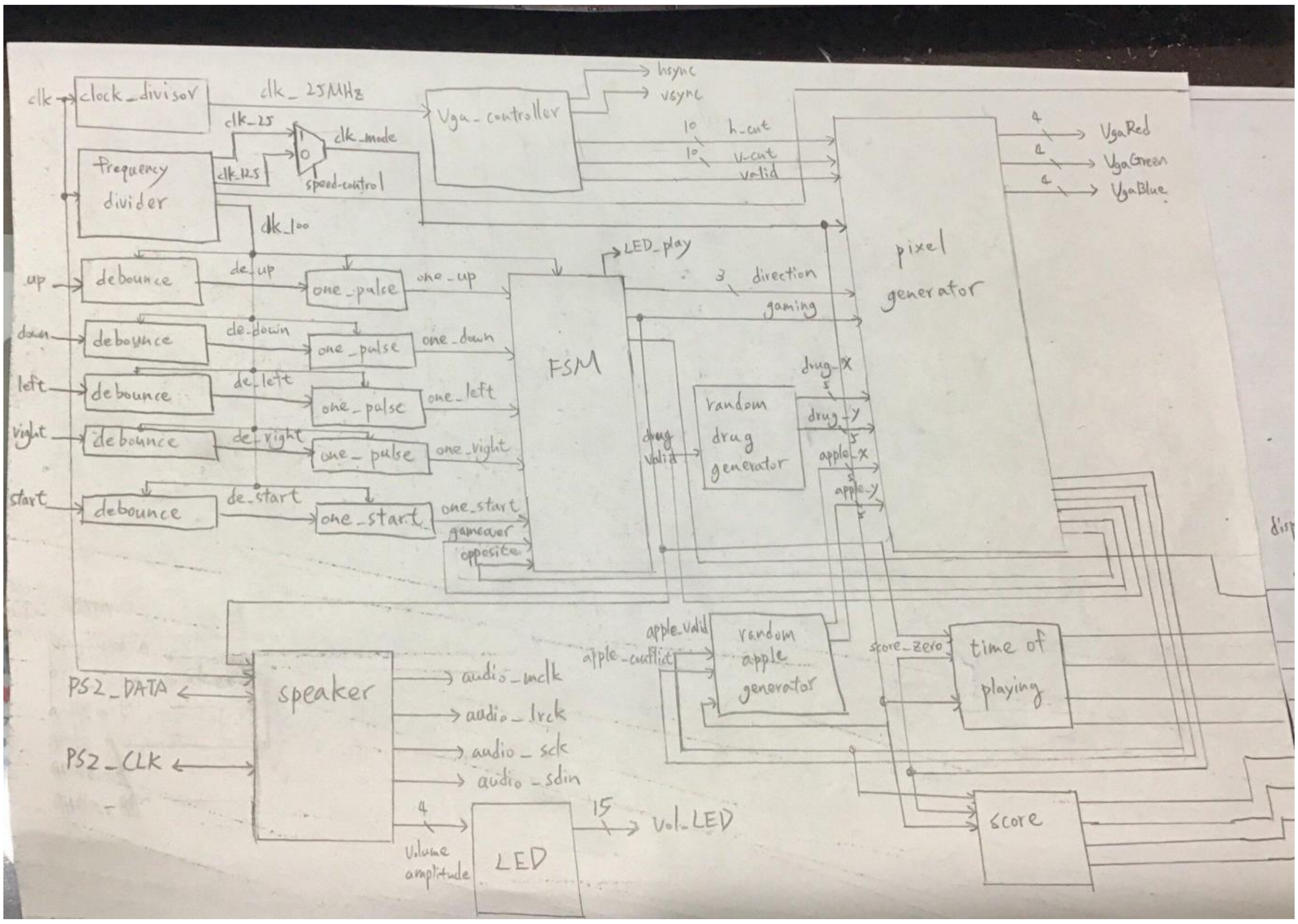
Input:

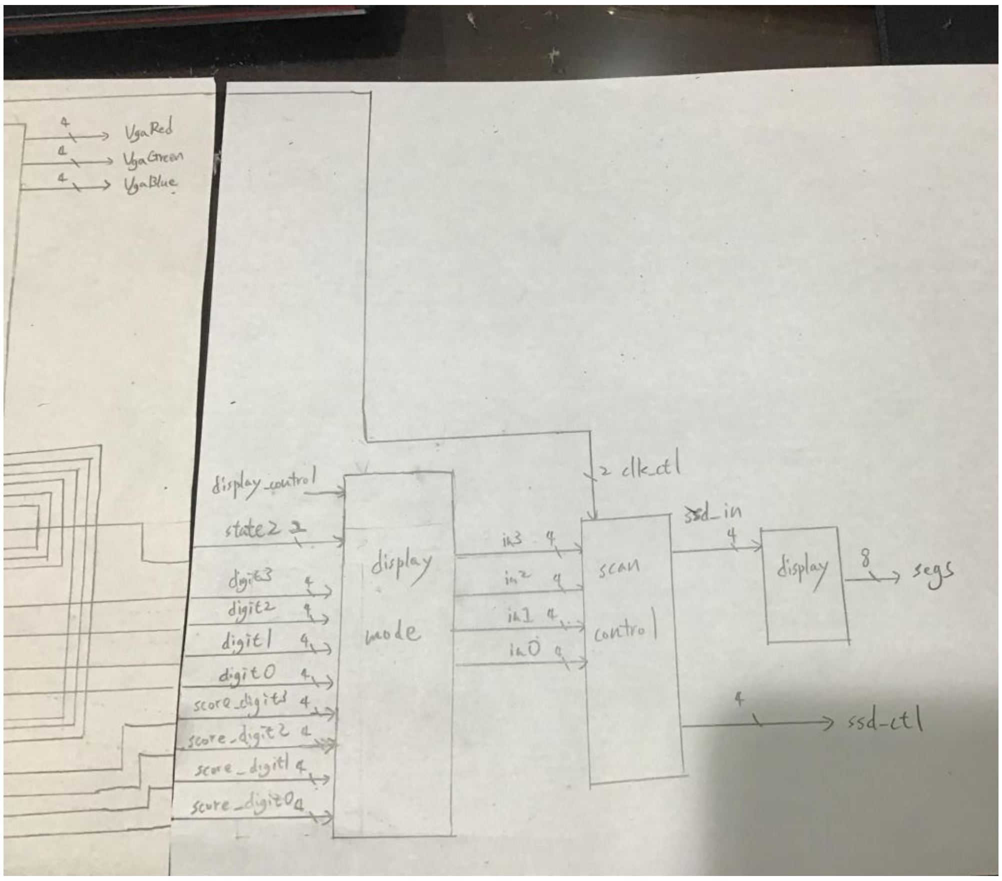
[3:0] binary (used to identify the status of counting)

Output:

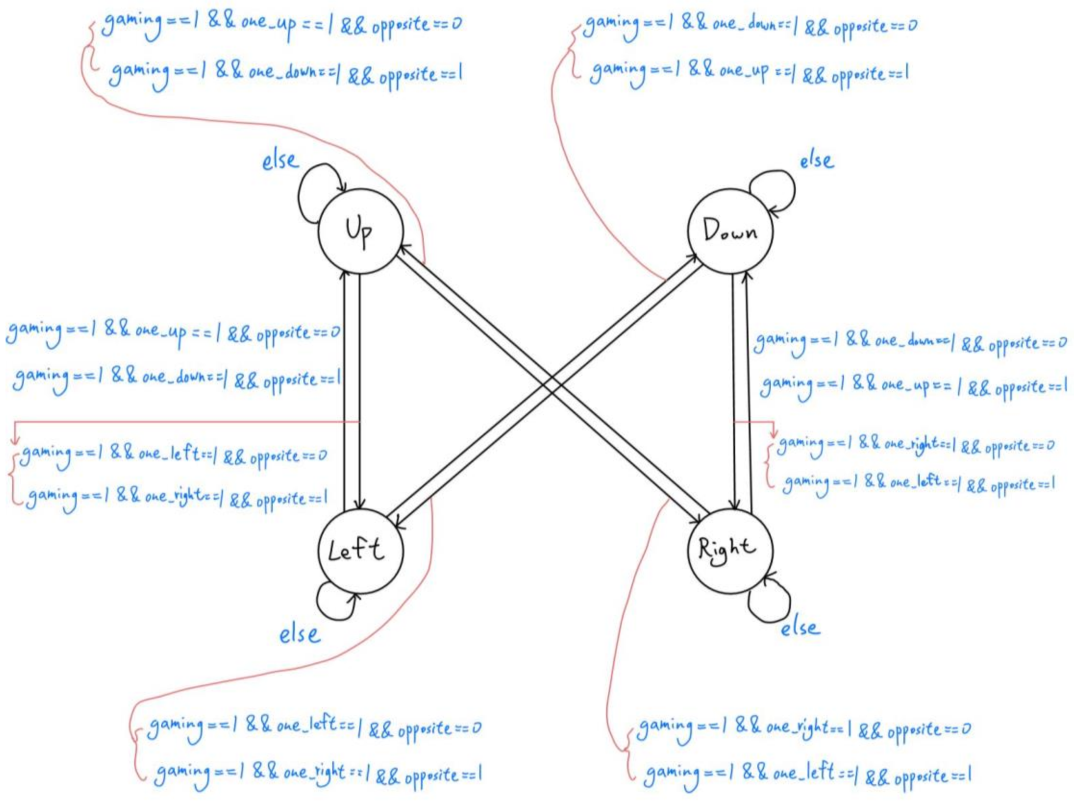
[7:0] segs (used the value at “ssd_in” to represent different image at the seven segment display)

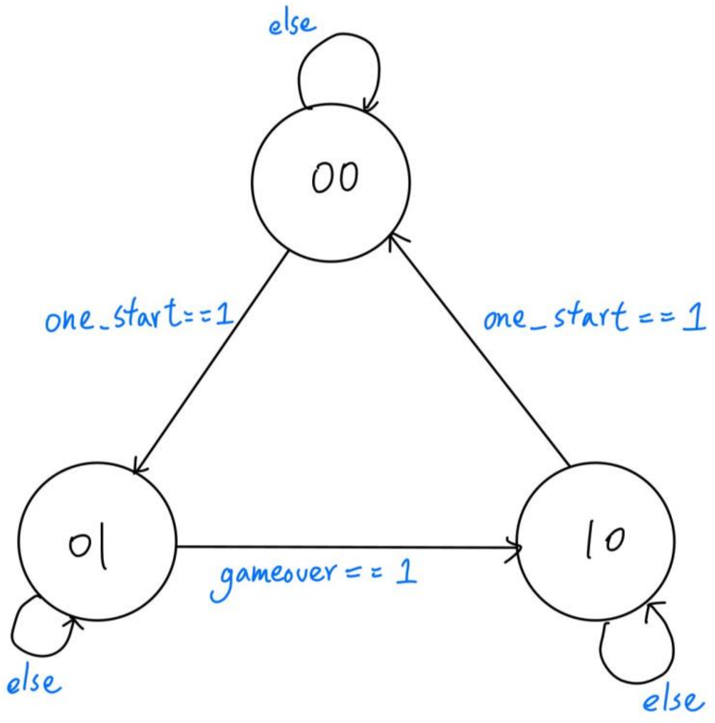
logic diagram:





State diagram:


gaming==1 && one_up==1 && opposite==0
gaming==1 && one_down==1 && opposite==1
gaming==1 && one_down==1 && opposite==0
gaming==1 && one_up==1 && opposite==1
else
Up
else
Down
gaming==1 && one_up==1 && opposite==0
gaming==1 && one_down==1 && opposite==1
gaming==1 && one_down==1 && opposite==0
gaming==1 && one_up==1 && opposite==1
gaming==1 && one_left==1 && opposite==0
gaming==1 && one_right==1 && opposite==1
gaming==1 && one_right==1 && opposite==0
gaming==1 && one_left==1 && opposite==1
Left
Right
else
else
gaming==1 && one_left==1 && opposite==0
gaming==1 && one_right==1 && opposite==1
gaming==1 && one_right==1 && opposite==0
gaming==1 && one_left==1 && opposite==1



Design method:

- Pre-thinking:

  In order to realize the snake game, we thought that we should apply keyboard and speaker to design BGM, apply display so that we could see the snake moving and be able to play the game, apply different counters to count time consuming and score. In addition, we could also design a accelerating mode to make our game more exciting. After checking what we might need to design, we started to design our program.

- Main design procedure:

1. Import the basic modules we may need, including frequency divider, keyboard decoder, speaker control, debounce and one pulse modules.
2. In BGM module, we divided the design procedure into a few steps:
   I. The up_and_down_counter (show the volume level), frequency divider, speaker (control the voice output), scan control (control the scanning), display (control the seven segment display) and volume control (control the volume level) are all identical to the ones in lab8 and lab10, so we imported them from lab8 and lab10 to this program directly.

   II. Define the upper bound of each sound so that we can design the music more easily. For example, `define do = 22'd191571.

   III. Use the counter to control the frequency of speaker, and the design procedure is divided to two parts:
   - ➢ D FF and counter design:
     Define when to reset the counter so that the BGM could be played again and again. Also, if the program is still in gaming status (gaming = 1), then the counter should plus one continuously in a specific frequency (f = 5Hz). On the other hand, if gaming = 0, then the music should be stopped.
   - ➢ Assign the frequency to each second
     After designing the counter, we have to assign the frequency to each number so that a song could be formed. For instance, 8'd0: note_choose (used to record the upper bound of the counter in speaker module) = `do (the defined variable which stands for upper bound for counter so that the frequency of "do" could be formed),

8'd1: note_choose = `re.

IV. Summarize all the module into a top module called "top_speaker".

3. We start to design the VGA part, and it is the most important part in this program, and the design procedure is shown as below:

   I. First, in "pixel_gen" module, the size of a unit square is 20*20, and it divides a display into 32*24 squares. (640/20 = 32, 480/20 = 24). Afterward, the squares will be used as the sections of snake's body. Consequently, h_cnt/10'd20 and v_cnt/10'd20 will be used as x_axis and y_axis (unit length)

   II. Declare [5:0] x and [5:0] y to store the address of snake's "head", and [5:0] x1, [5:0] y1, …. , [5:0] x30, [5:0] y30 are used to store the address of snake's body in section. That is, there will be 31 sections in total.

   III. In order to display the snake's head at the display, we set the snake's head to be green by vga_rgb (12'h0F0 -> green). Also, we use x_axis and y_axis to record the address of snake's head, and the x and y are used to record the address of current scanning pixel. Consequently, if x_axis = x, y_axis = y, then we could find where we have to set the color to be green, that is, it is the address of snake's head.

   IV. In order to control the snake's head to be able to move, we use a "counter" and "finite state machine" to control the moving direction. The default direction is "right", and each state is shown as below:

direction:	gaming:
2'b00: up	1'b0:非遊玩狀態
2'b01: down	1'b1:遊玩狀態
2'b10: left	
2'b11: right	

   V. After designing the moving direction of the snake, we start to design "snake moving method". Since the snake moves in segment by segment, so we can use a "shifter" to realize this thought.

   VI. Next, we come to a quite important part: When will the game end? That is, we have to define the border condition. First, the game will end if the snake touches the edge of display. Since the range of display is: "0 ≤ x_axis ≤ 31", "0 ≤ y_axis ≤ 23", so if the snake's head goes to x = -1, x = 32, y = -1, or y = 24, the game will end. However, since the answer of "0 minus 1" would be "6'd63" instead of "-1" in 6-bit system, so we have to be cautious when writing the border condition (if-else statement). For instance, we have to write "x == 6'd63" instead of "x == -1".

   VII. After defining the border condition, we want to design the "apple". In order to realize this function, we have to design a "random generator" to generate apple, and the design procedure can be divided as below:

   - 9-bit linear feedback shift register (LFSR)
     We declare r0, r1, …, r8 and initialize with (1,0,1,0,1,1,0,1,1). The "in" is assigned with r7^r8. Since the value of "in" is determined by r7 and r8, and the value will shift when passing through a clock. As a result, the value of r1, r2, …, r8, in will be "random".
   - Process and record the address of apple
     Next, we define some registers and a D FF to restore the value of sum, and the value of "sum" will be used to decide the address of apple. For instance, apple_x = sum%32, apple_y = sum%24, then we could know the address of apple.

   VIII. In order to combine the function of VGA and random_apple _generator (reveal the apple at the display, and a new apple will be generated when the snake eats apple), we connect the apple_x and apple_y to the pixel_generator, and write the if-else statement to assign the address of apple and show "red". For instance, if x_axis = apple_x, y_axis = apple_y, and the

value of "vga_rgb" would be "12'hF00" (red).
Next, we define a variable called "apple valid" to avoid the condition that the addresses of apple and snake are same. Unless this condition is true (apple_valid = 1), or the random apple generator won't generate new apples.

IX. To control the length of snake, the design procedure can be divided as below:
- Define the length of snake:
  - A. When apple_valid = 1 (eats the apple) and the length is less than 30, the length of snake will extend for one segment.
  - B. When we haven't started the game (gaming = 0), then the length of snake is 1 in default.
  - C. The length of snake won't change in other cases.
- Extend the snake:
  In the concept of "unlock" to extend the snake, we could imagine the scenario that the original length of snake is "31" already, however, the body segments are black in the beginning. Consequently, we see the snake moving with "head only". If it eats the apple, then the next segment will be "unlocked" to be green, so the snake "extends". (use "case" statement to determine the length of snake")

X. Also, we want to set another ending condition, "when the snake contact with its body, the game will be ended". To realize this thought, we use the address of head and the addresses of body segments to determine, if the address of snake's head is equal to the address of whatever segment of the body, the game will end. Take length = 5(snake's body length) for instance, if the address of snake head (x, y) is equal to the address of whatever segment of snake's body ((x1, y1), (x2, y2), …, (x4, y4)), then the game will end.

XI. After the design above, an idea came up to us, "why don't we design a drug to confound the snake so that the game will become more intriguing?" Consequently, we design a new module called "drug_random_generator" to generate the drug randomly. The design method is quite similar to the one in "apple_random_generateor", however, we have to declare a new variable called "apple_conflict" in "pixel generator" to avoid the condition that the address of apple and drug are the same. Moreover, "drug_valid" is also declared, whose function is as same as "apple_valid". As a result, we can not only generate the drug randomly (by the similar method in apple random generator), but we can also avoid the conflict condition by adding some if-else statements to pixel generator. For instance:
   A. If head_x = drug_x, head_y = drug_y, then (drug_valid, apple_valid, apple_conflict) = (1, 0, 0)
   B. If head_x = apple_x, head_y = apple_y, then (drug_valid, apple_valid, apple_conflict) = (0, 1, 0)
   C. If apple_x = drug_x, apple_y = drug_y, then (drug_valid, apple_valid, apple_conflict) = (0, 0, 1)

   If the "apple_conflict" is detected to be 1, then the generator will generate a new apple until both addresses are not same.

XII. We set a variable called "opposite" in "FSM direction" module to control the "meaning" of each button. For instance, if the snake hasn't had the drug, then up means "up", down means "down", left means "left", right means "right". However, if it has had the drug, then up means "down", down means "up", left means "right", right means "left" instead.

4. Design the score and time counting function:
   In order to complete this function, we have to import the up counters and summarize four four-bit counters into a top module (since there are four digits). On the other hand, since the score we get when the snake eats apples is different in normal mode and in speed up mode, so we have to use if-else statement when designing the score module. For instance, if apple_valid = 1 (eats the apple) and speed = 0 (in normal mode), then we get 10 points per apple; however, if apple_valid = 1 and speed = 1 (in speed up mode), then we get 30 points per apple instead. By the way, the speed

of snake is controlled by frequency divider and DIP switch (R2) in top module later, if DIP switch is on, f = 25Hz; if DIP switch is off, f = 12.5Hz.

5. After the design above, we summarize all the modules into a top module and add some functions to make it more perfect.
   - Show the volume level by the number of LED:
     In order to realize the volume level more clearly, we design the number of shining LEDs to represent the volume level. For instance, if the volume level is 10, then there will be 10 LEDs shining.
   - Switch the mode between score and time consuming:
     We use the state in "FSM direction" and the DIP switch (T1) to change the mode:
     A. If state = 00, then we initialize the value of digit3, digit2, digit1, and digit0 to be 0.
     B. If state = 01 or 10 and DIP switch is off, then the seven segment display will show the current score.
     C. If state = 01 or 10 and DIP switch is on, then the seven segment display will show the playing time.
   - Control the speed of snake moving:
     We use a DIP switch (R2) to control the speed.
     A. If DIP switch is off, then it means that it is in normal mode, and the frequency is 12.5Hz. Moreover, we get 10 points per apple.
     B. If DIP switch is on, then it means that it is in speed up mode, and the frequency is 25Hz. Moreover, we get 30 points per apple.

After the design above, our final project program is accomplished.

clk	vgaRed [0]	vgaRed [1]	vgaRed [2]	vgaRed [3]	vgaBlue [0]	vgaBlue [1]	vgaBlue [2]	vgaBlue [3]
W5	G19	H19	J19	N19	N18	L18	K18	J18
rst	vgaGreen [0]	vgaGreen [1]	vgaGreen [2]	vgaGreen [3]	hsync	Vsync	PS2_ DATA	PS2_ CLK
V17	J17	H17	G17	D17	P19	R19	B17	C17
up	down	left	right	start	speed_ control	LED_ play	segs [7]	segs [6]
T18	U17	W19	T17	U18	R2	U16	W7	W6
	segs [5]	segs [4]	segs [3]	segs [2]	segs [1]	segs [0]	display_ control	audio_ mclk
	U8	V8	U5	V5	U7	V7	T1	A14
	audio_ lrck	audio_ sck	audio_ sdin	vol_ LED [14]	vol_ LED [13]	vol_ LED [12]	vol_ LED [11]	vol_ LED [10]
	A16	B15	B16	L1	P1	N3	P3	U3
	vol_ LED [9]	vol_ LED [8]	vol_ LED [7]	vol_ LED [6]	vol_ LED [5]	vol_ LED [4]	vol_ LED [3]	vol_ LED [2]
	W3	V3	V13	V14	U14	U15	W18	V19

	vol_ LED [1]	vol_ LED [0]						
	U19	E19						

Discussion

✧ Check the accuracy of our program:

To check whether my design is accurate, we think of that there are some points need to be checked:

1. Whether the speaker is able to output specific music.
2. Whether the keyboard is able to modify the level of volume, and the level should be displayed with LEDs.
3. Whether we can use the DIP switches to know the current score and time consuming.
4. Whether game will be ended if the snake touches the edge of screen.
5. Whether game will be ended if the snake's head touches its body.
6. Whether the apple (red square) is able to generate automatically and randomly.
7. Whether the snake will extend whenever it eats the apple.
8. Whether a LED is able to flicker when the game is ended.
9. Whether the speed-up mode is set up properly, including the movement of snake (it should move as the real snake moves) and the change of velocity.
10. Whether the score per apple is altered if we change to different speed mode (normal: 10 points per apple, fast: 30 points per apple)
11. Whether the drug (blue square) can affect the direction of snake.

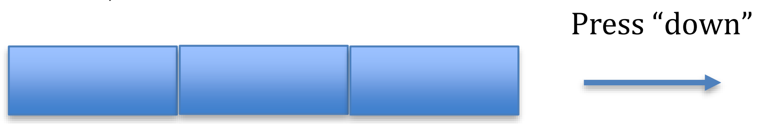
After implementing the program to the FPGA board, we found that the program performs as expected, so it can vindicate that our design method was accurate.

✧ Problem solving:

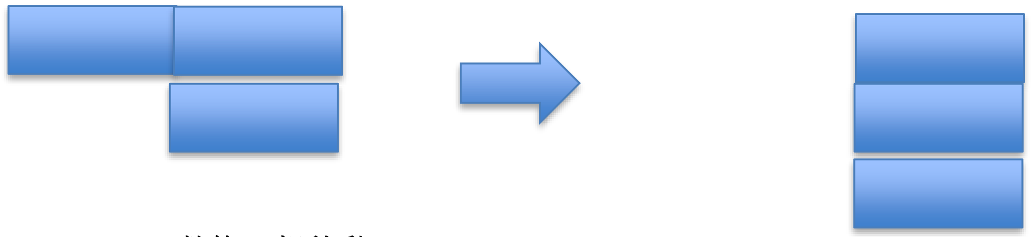
- BGM part:
  1. The music plays more slowly than expected, and it seems that the tone changing frequency needs to be fixed.
     ➔ Change the frequency of counter from 1Hz to 5Hz, and the music plays faster than before.
  2. Even if we import the keyboard related module into the top speaker module, we can't control the volume with the pad at the keyboard.
     ➔ After checking the top module, we found that the input and output ports are connected inaccurately, so we fixed this problem and it finally worked well after doing so.

- VGA part:
  1. When the snake is moving in a direction (ie. right) and we press a button whose direction is the anti-direction (ie. left), the snake won't ignore this instruction and move in anti-direction (ie. left).
     ➔ We did some revision to FSM so that the state won't change when we press the anti-direction button.

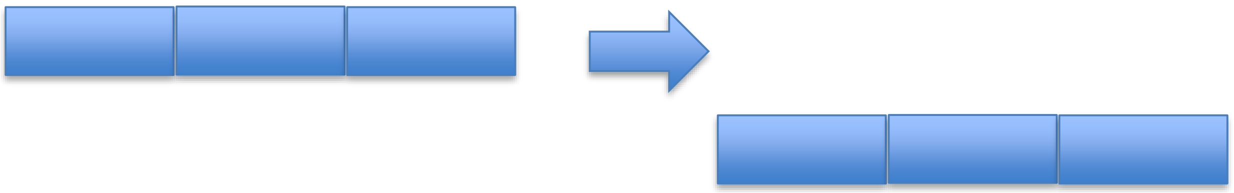
2. When the snake is larger than 2 segments, the snake will move in weird way. For instance,



Accurate: (彎曲移動)



Inaccurate: (整條一起移動)



➔ Use the characteristic "flip flop delays for a clock" and chain lots of flip flops together. Whenever a clock passes through, send the address of head to first segment of body; send the address of the first segment of body to the second segment of body; send the address of the second segment of body to the third segment of body, and so on.

3. The appearance of apple is in a specific sequence. Even if we reset the program, the sequence is still the same.

➔ Restore the value generated by a flip flop. Although LFSR generates values continuously, it renews the stored value until the snake eats the apple.

4. After adding the design of drug, the addresses of apple and drug are always same.

➔ Since we initialize two LFSRs with same values, so we can solve this problem simply by revising the initial values of registers of apple or drug.

5. The changing of finite state machine is not stable.

➔ Since I forgot to add the "( )"when writing the if-else statement, and it is used to determine the "gameover" to be 0 or 1. However, "gameover" is mainly used to change states, so if I forgot to add the "( )", then the value of gameover will be unstable, and the state change will be unstable as well. The revision is shown as below:

> Original:
else if(length==5'd5 && (x==x1 && y==y1)|| (x==x2 && y==y2) || (x==x3 && y==y3) || (x==x4 && y==y4))

> After revising:
else if(length==5'd5 && ((x==x1 && y==y1) || (x==x2 && y==y2) || (x==x3 && y==y3) || (x==x4 && y==y4)))

6. The brightness of left most column is much different with others.

➔ We assume that we forgot to assign value to vga_rgb in VGA controller when valid = 0, so the column will be unstable and reveal weird color, that is, the logic determination statement was not complete. Consequently, we assigned vga_rgb = 12'h000 (black) if valid (whether to reveal things at display) = 0 to solve this question.

7. When the snake touches the left and up border, the game didn't end.

➔ Since I divide the display into 32*24 unit squares, and the range of x axis and y axis fall in the range 0 ≤ x_axis ≤ 31", "0 ≤ y_axis ≤ 23. That is, if x = -1, x = 32, y = -1, y = 24, then the game will be ended. However, in 6-bit system, the answer of "0 minus 1" is "6'd63" instead of "-1". Consequently, I replaced x = -1 with "x = 6'd63", replaced y = -1 with y = 6'd63.

Conclusion

In this project, we spent lots of time thinking of what functions we should design, and how could we make the program perform more perfectly. Although we met tremendous of problems and spent lots of time analyzing the solution, we still consider that we learn a lot from the designing and debugging procedure of final project. Not only we learn the technique of solving problems, most importantly, we know how to "design" a program to make it more familiar to people, that is, people are more willing to use our program without any significant problem.

In our opinion, the core spirit we should know through the logic design lab this semester is that we have to "design" a program not only it can be used, moreover, but we should think of what users want. For instance, we can simply design electronic clock with counter, however, if we add calendar or even stopwatch function, then the user will be more willing to use the program we design, and this spirit will become more and more vital in the future. Hope that we could apply what we have learned in logic design lab to the further courses, or even jobs in the future.

Work contribution

張嘉祐: 整合和設計 VGA module (蛇的移動、隨機產生蘋果、迷幻藥)，計分、計時、加速功能設計、debug。
柳奕丞: 設計 BGM module (speaker + keyboard)、整合書面報告、debug。

References

1. Verilog 硬體描述語言數位電路 設計實務，鄭信源著，儒林圖書公司發行，2016 年 4 月九版， ISBN:978-957-499-980-4