Lab 4 report CS M152A

Student name: ZHAOHAO MAI

UID: 504668217

Student name: JU MING

UID: 104146197

1. Introduction and requirement:

This is our final project, we design, implement, and demo a game implemented by Verilog or some other compatible programming language with the circuit board provided by the class. And the complexity should be the same level as the example presented by the Lab 4 sample proposal. Prior to the process of designing structure of code and implementing games, we write the proposal for our design. Here is the core part of Lab proposal of our design, which was approved by the Teaching assistant:

Overview of game:

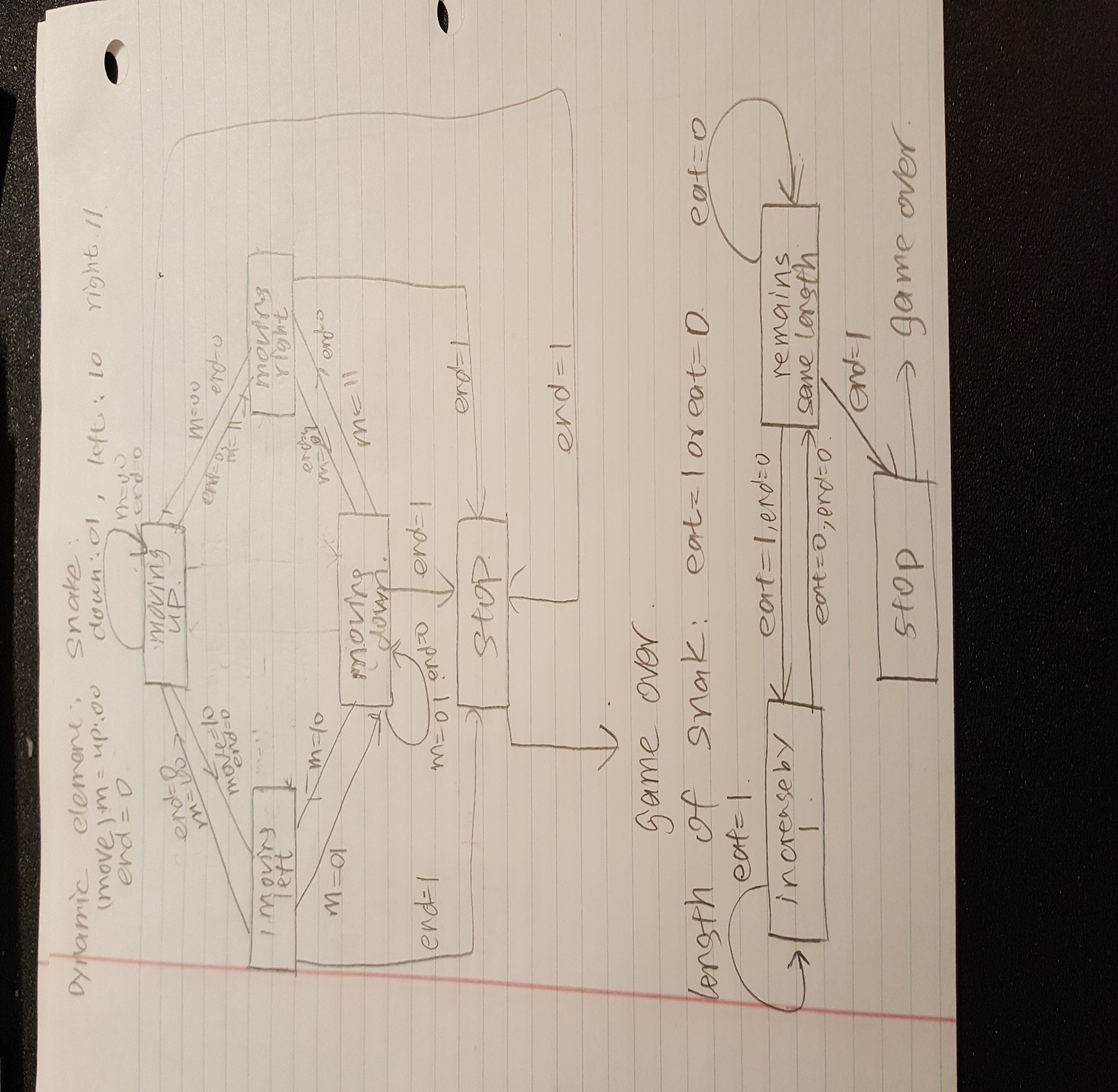
The idea of the Snake game is based on the NOKIA mobile phone game Snake. The player controls a line segment on a bordered plane. As it moves forward, it leaves a trail behind, resembling a moving snake. The snake has a specific length, so there is a moving tail a fixed number of units away from the head. The player loses when the snake runs into the screen border, a trail, or another obstacle. Therefore, we implement a greedy snake game on the FPGA board.

Also, we follow those rules to implement our game:

1. Grading Rubric:
   1. Direction Button (25%): Use 4 direction button – up, down, left, right – to change the direction of the snake head.
   2. Reset Button (10%): This button could be treated as manually game over – its behavior is basically the same as game over, reset the snake’s length to minimal and restarts the game.
   3. Semi-random placement of “food” (10%): Use a semi-random (x, y) such as using something related with time counter, as each time the “food” was eaten by the snake, place a new one on the coordinate (x, y).
   4. Time-update (10%): The Snake will move towards the head direction 1 unit length as 1 unit period of time passed.
   5. Increment as food eaten (15%): As the snake eats a food, the length of a snake increment by 1 unit length
   6. Judgement of death (15%): If the head of snake was hit by the wall of the snake itself, GAME OVER.
   7. Score display (5%): Display the length of snake on the seven-segment display.
   8. “Texture” of the snake (10%): We want the snake resembles real-life snake – with textures. The NOKIA game provides a good example.

According to the description of game rubric above, the game consist of three major parts, the first part is code for displaying score for the game, the second part is about designing the behavior of snake (with most work), and the third part is about VGA controlling.

The state diagram:



1. Design description:

Due to the great size of project, here we discuss the core code block. We have six segments of code, there are “BCD.v”, “bin27.v”, “Snake.v”, “clock.v”, “vga\_control.v” and “big2small.v”. We will only talk about vga\_control.v and the main module Snake.v , because the rest of them are used to control the display of total score in the current state of game, which was talked a lot in the previous lab (Lab 3).

* 1. VGA controller

The code block of vga\_controller has one input, the input carries information of the clock and pixel. The clock input synchronizes all the process and forces all the change happen simultaneously. The pixel input carries the information of pixel and vga\_controller will update the information on every pixel. The output variables, HS, VS, hcounter, vcounter, and blank indicates the updated information for corresponding pixels. Form the code, we use 5 always block to continuously update the value of output variables by 1, these implementation can make the snake moves horizontally or vertically by one step when the clock hits the positive edge. Also, the output variables “hcounter” and “vcounter” indicates the horizontal length of snake and vertical length of snake. Vga\_controller handles the video show on the screen, and with the input clock, the screen will show the movement of snake.

Moreover, it would be required for the main module to communicate to VGA controller, we need to send which pixel is in the blocks to be shown by the VGA controller and its color. This would be accomplished by the following scan function (for example, just for the snakes).

// judgement of each pixel -- whether it is in a segment of snake.

for (i = 1; i<=25; i = i+1)

if (score >= i)

snakeseg[i] = ~blank & (hcount >=x1[i]+1 & hcount <= x1[i] + 19 &vcount >= y1[i]+1 & vcount <= y1[i]+19);

Snakeseg[i] would be a Boolean function that returns whether this pixel is in the snake segment, while food and wall serves as the same function for food and wall, respectively.

* 1. Snake

The code block of “Snake” was the heart of the project because it contains all the block we discussed above. It could be clearly divided in to the same portions just as in the grading rubric. We will analyze them below:

* + 1. Direction Button (25%): Use 4 direction button – up, down, left, right – to change the direction of the snake head.

This would be very sample:

// a) update the snake direction according to the button

else begin

foodxcount = (foodxcount + 5'd20) % 10'd600;

foodycount = (foodycount + 5'd20) % 9'd420;

// you cannot move 180 degrees...

if (up==1'b1 && d\_down == 1'b0) begin

d\_up =1; d\_down=0; d\_left=0; d\_right =0;

state = next\_state;

end

else if (down==1'b1 && d\_up== 1'b0) begin

d\_up =0; d\_down=1; d\_left=0; d\_right =0;

state = next\_state;

end

else if (left==1'b1 && d\_right==1'b0) begin

d\_up =0; d\_down=0; d\_left=1; d\_right =0;

state = next\_state;

end

else if (right==1'b1 && d\_left==1'b0) begin

d\_up =0; d\_down=0; d\_left=0; d\_right =1;

state = next\_state;

end

else

state=state;

end

* + 1. Reset Button (10%): This button could be treated as manually game over – its behavior is basically the same as game over, reset the snake’s length to minimal and restarts the game.

// b) reset button

if (reset == 1'b1) begin

d\_up =0; d\_down=0; d\_left=0; d\_right =0;

foodx = foodxcount;

foody = foodycount;

score = 0;

lose = 0;

dead = 0;

state = next\_state;

end

* + 1. Semi-random placement of “food” (10%): Use a semi-random (x, y) such as using something related with time counter, as each time the “food” was eaten by the snake, place a new one on the coordinate (x, y).

// c) semi-random placement of food

if (snakehead && food) begin

foodx = foodxcount;

foody = foodycount;

end

While…

foodxcount = (foodxcount + 5'd20) % 10'd600;

foodycount = (foodycount + 5'd20) % 9'd420;

every posedge of clk.

* + 1. Time-update (10%): The Snake will move towards the head direction 1 unit length as 1 unit period of time passed.
       1. More than rubric, in order to faster the debugging, I set 2 levels of speed of snake – fast and slow. Slow moves a unit length per about 1/3 sec, while fast speed doubles the speed.

reg [25:0] slow\_count;

always @ (posedge clk) begin

slow\_count = slow\_count + 1'b1;

if (speed == 1'd0) begin

slow\_clk = slow\_count[24];

end

else if (speed == 1'd1) begin

slow\_clk = slow\_count[23];

end

end

* + 1. Increment as food eaten (15%): As the snake eats a food, the length of a snake increment by 1 unit length

if (snakehead && food)

score = score+1'd1;

* + 1. Judgement of death (15%): If the head of snake was hit by the wall of the snake itself, GAME OVER.

for (i = 1; i<=25; i = i+1)

if (snakehead && snakeseg[i])

lose=1'b1;

for (i = 1; i<=4; i = i+1)

if (snakehead && wall[i])

lose=1'b1;

* + 1. Score display (5%): Display the length of snake on the seven-segment display. – shown in the lab 3.
    2. “Texture” of the snake (10%): We want the snake resembles real-life snake – with textures. The NOKIA game provides a good example.

We just shrinked the blocks of the snake by 2 pixels to show the “texture” of the snake.

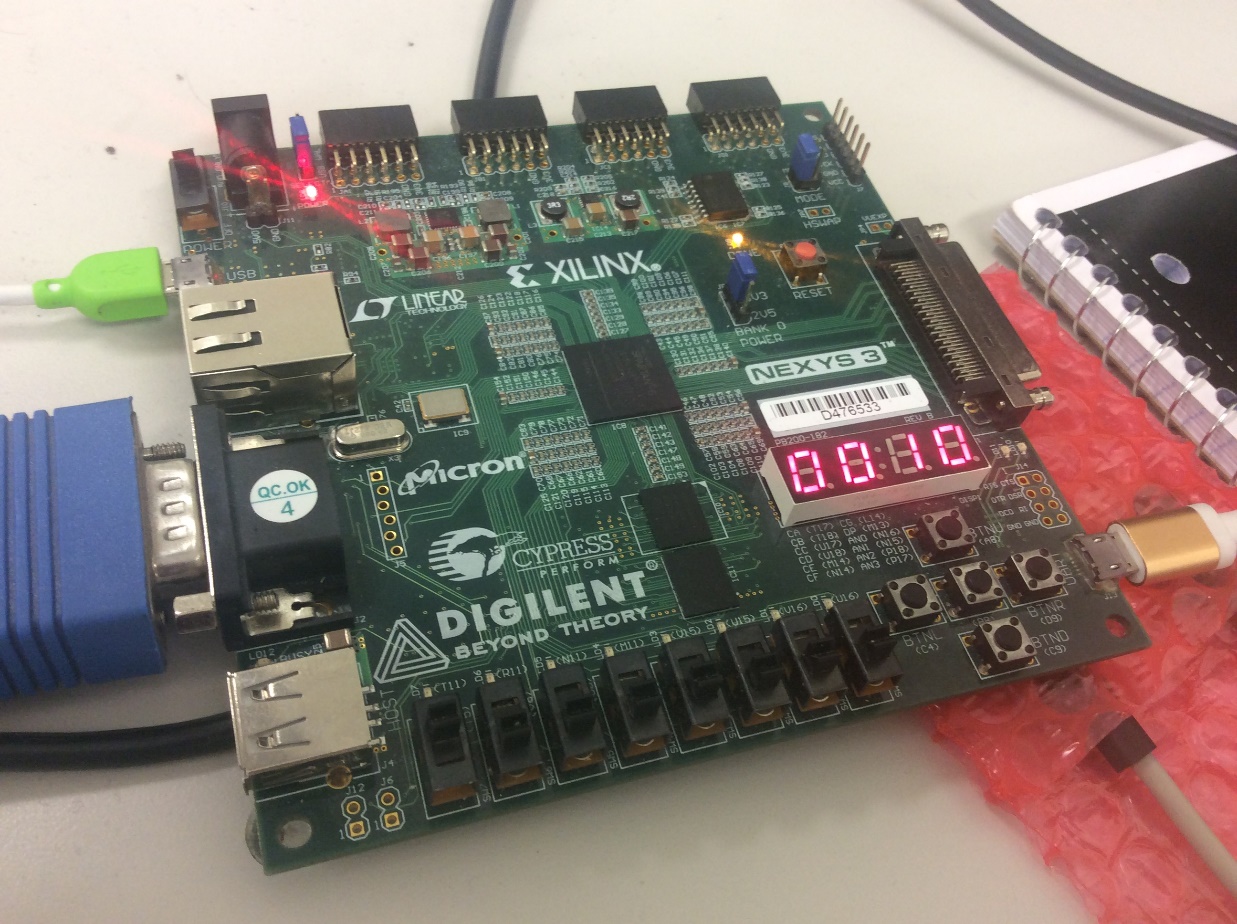
1. Simulation:
   1. Initial



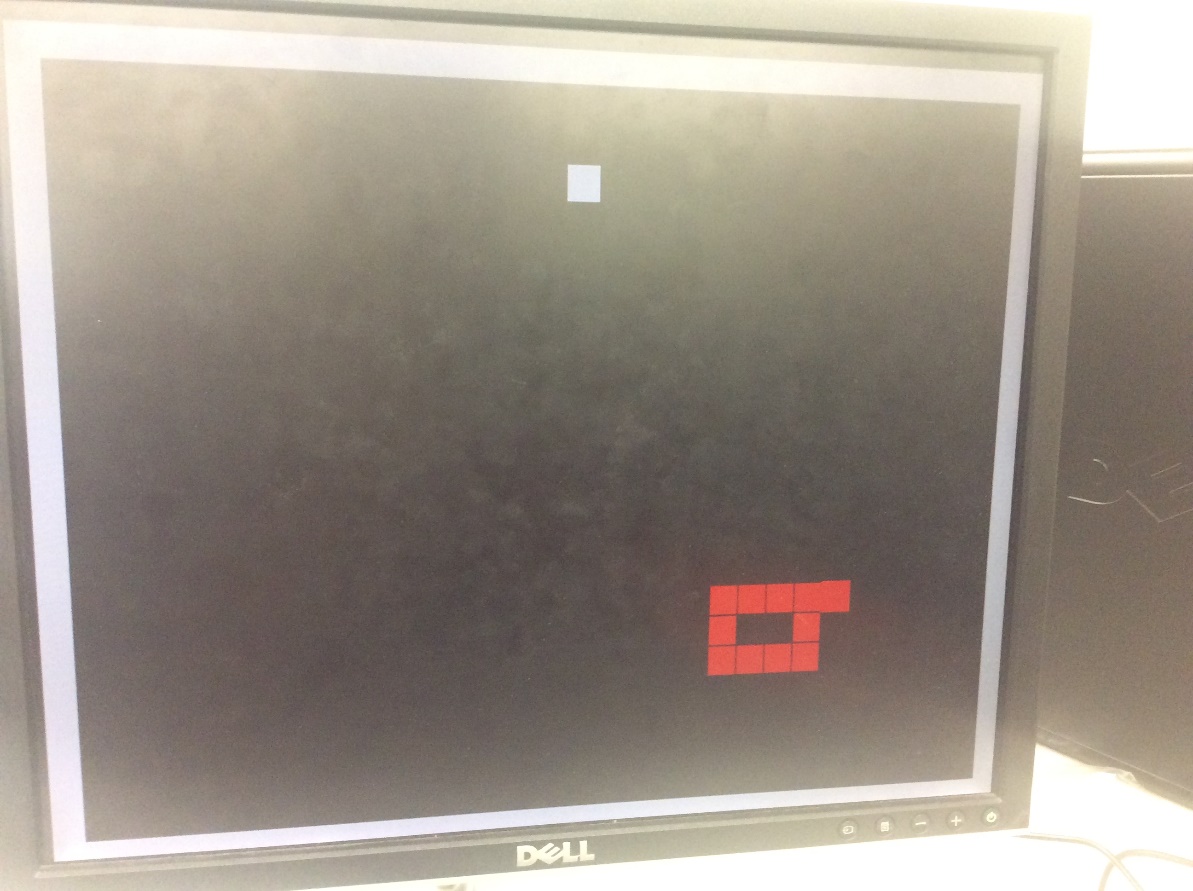
* 1. Hit by wall



* 1. Score display



* 1. Hit by itself



1. Conclusion:

We can control the snake and play the game under the support of the FPGA board. The size of this project is huge and we have to deal with all kinds of glitches and bugs. The most difficult kind of bugs created by VGA output module. For example, we spend a lot of time on achieving the vertical movement of snake and horizontal movement of snake. However, initially, the block moves diagonally on the screen and it never hits the wall. Then, we modify the codes and try different combinations of parameters to force the snake head, which is the block, to move horizontally and vertically. Moreover, we also found that some states fail to happen even if the inputs satisfy the conditions, and sometimes, the changes of dynamic components and static components fail to happen simultaneously. Then, we approach solutions by separating the whole implementation block into some smaller size implementation block, for example, wall, snakes, position, colors and continuity of the game. Moreover, we force all the relevant blocks under control by the same clock, and synchronize all the changes happening at the positive-edge of time click. Also, at the very beginning of the project, we have different perspectives of the game and do not agree with each other in the mechanism of game. In order to resolve difference between us in designing the prototype of snake, we individually write prototype of the snake and compare with each other for better scheme, and we reach agreement in writing code of the prototype after lots of times of communication and explanation. The suggestion we get for improving efficiency for writing code is that communications should be prior to scheme and reaching agreement in fundamental design should be made.