**CS152A**

**Lab 4**

**Section 4 - Team 10**

**Ran Gong**

**Mu-Te Shen**

**Shen Teng**

**Introduction and Requirement**

In this lab, we are expected to design an useful program within the aspect of circuit, fulfilling through Verilog. This means that we have to design basically with the idea of wire and gates. This is a lot different from the programming nowadays, but the thought of circuit level will strengthen our understanding as a programmer.

Breakout is a very popular game. Traditionally in this game, users are expected to move the paddle which is located at the bottom of the screen horizontally to hit the dropping ball, and try to destroy all the bricks. However, if we hit the ball at the different position of the paddle, the change of direction (it might also change the acceleration depends on which version you are playing) might be different as well. To simplify the game, we decided to only allow 45 degree reflection upon any contact between the ball and the wall/bricks. However, there is a possibility that we might stuck into a dead loop, so we will allow the paddle to move vertically up to the middle of the screen.

Here are a list of requirements we want to achieve.

1. Ball properly moves upon the contact(45 degree reflection).
2. Bricks properly display and disappear after the contact.
3. Paddle properly moves (horizontally and vertically).
4. Appropriate display on the monitor through the VGA implementation (during the game and after the player win or lose the game).
5. Properly level up after player finishes each level. (reconstruct the bricks and speed up the gamel)
6. Properly display level and scores. (The first digit of the 7-digit display is level, and the rest are scores)
7. Pause the game if we lift up the pause switch.
8. Restart the game if we hit reset button.

**Design Description**

We divided our program into several modules to match object-oriented design, and make the functionalities of modules clean and clear.

**1. NERP\_demo\_top**

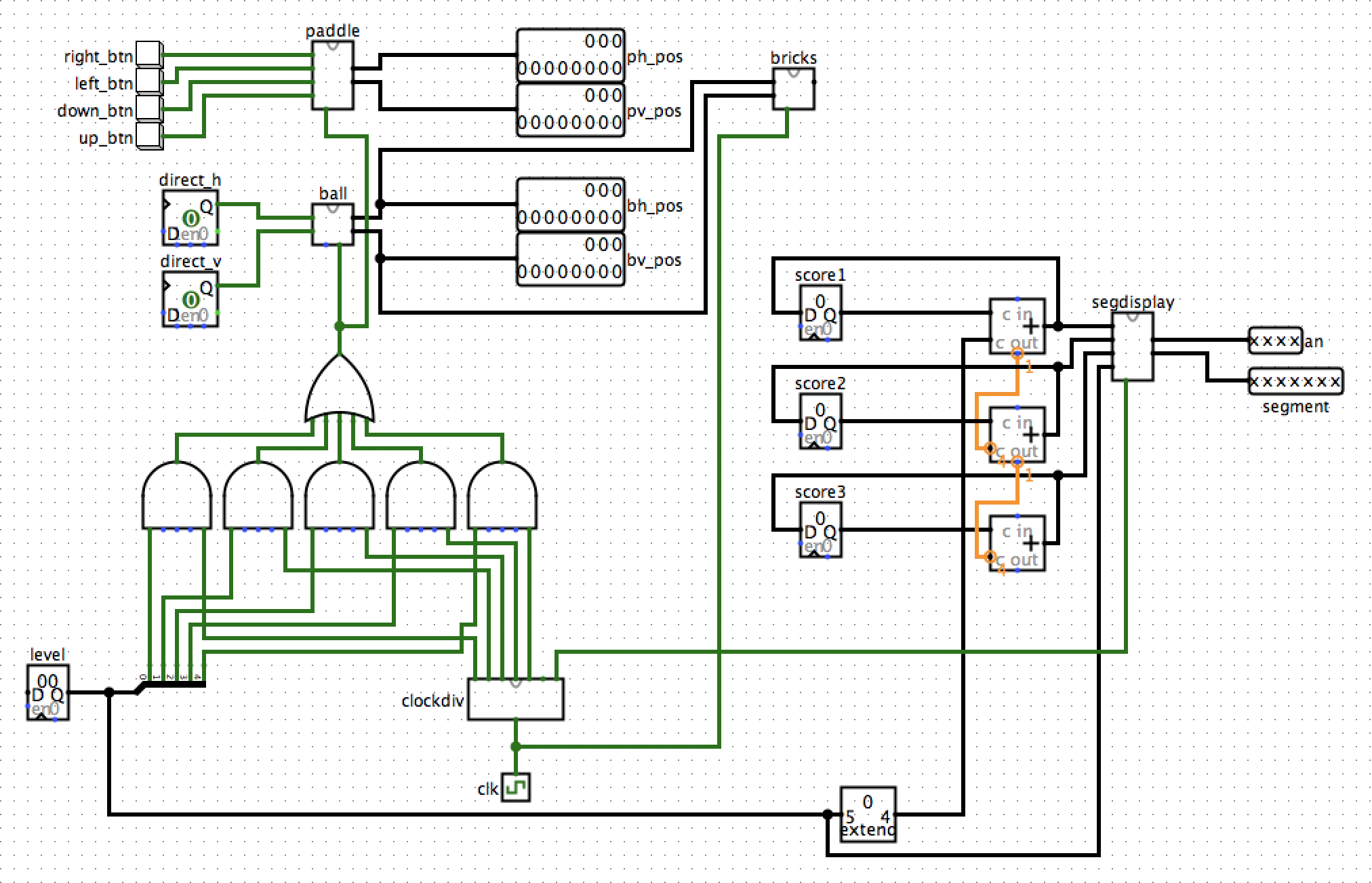
This module is the main module of our project, and it also serves as the control engine of all other modules. This module has the following tasks and calculation, so this can be treated as the main game engine.

1. Initialize the variables when the game start. This includes directions of the ball, score, level, and an indicator of gameover.
2. Calculate the collisions. This includes the collision between the ball and the bricks, the ball and the paddle, and between the ball and the walls(up, left, right).
3. Calculate the score when the ball hits a brick.
4. Control the level. The game will advance to the next level when the player clears all the bricks, and this will also increase the difficulty of the game. In different levels we use clocks with different frequency, so the game will get faster and more difficult.
5. Terminate the game when either the ball falls off the bottom edge, or the player finish all the levels.
6. Restart the game when the player hits reset button.

The way we calculate directions of the ball after colliding with a brick is interesting. We divided our game field into several 64x32 blocks, the same size as the bricks. We then numbered these blocks with row and column. This way, we can simply compare the previous and current block the ball is in to indicate which direction should be after the collision.

Input: sw, r\_btn, l\_btn, u\_btn, d\_btn, clk, clr

Output: seg, an, red, green, blue, hsync, vsync

****

Schematic of top\_level

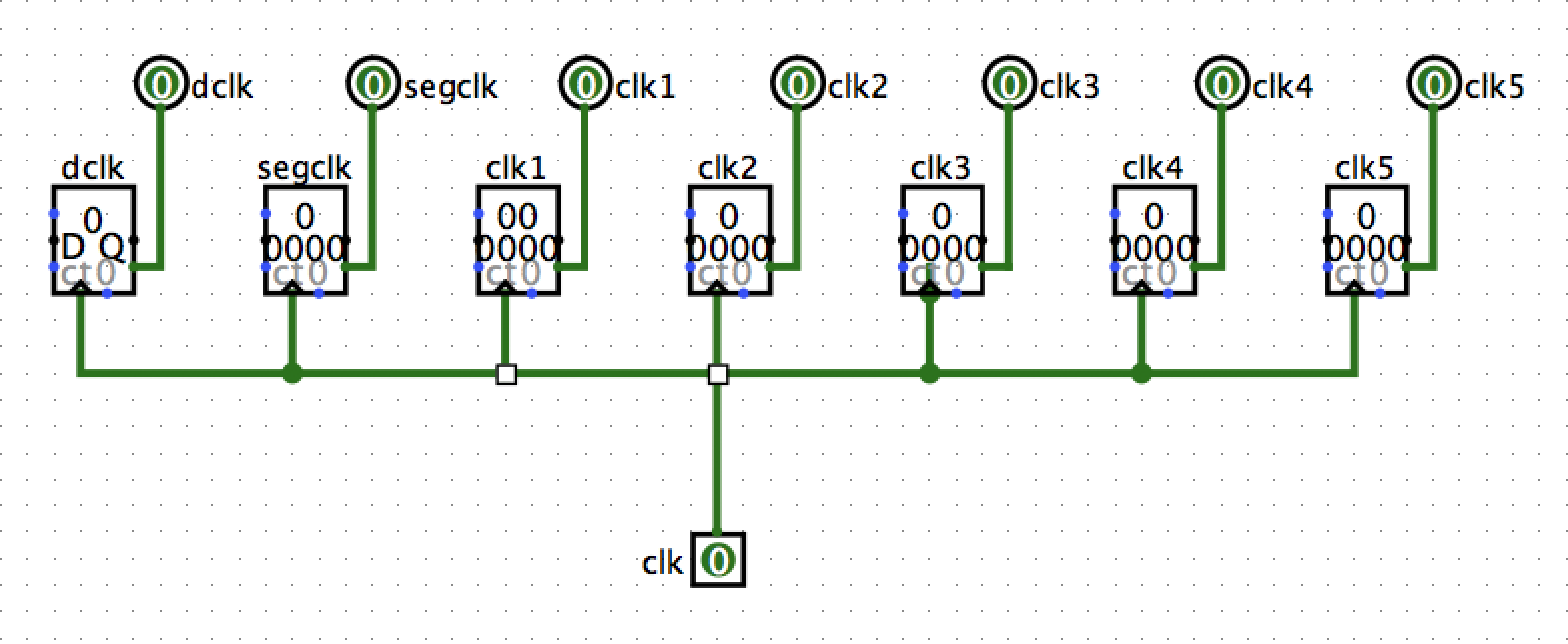
**2. clockdiv**

This module provides clocks of different frequencies for different uses throughout the program. We have the following three different categories of clock:

1. dclk: The 25MHz for the VGA display. This clock allows the vga640x480 module to run through each pixels fast enough to display a seemingly synchronized screen.
2. segclk: This clock is about 380Hz. This clock allows the 7-segment display to flash fast enough, so it seems like displaying constantly.
3. clk1~5: clk1 to clk5 are 47Hz, 59Hz, 71Hz, 82Hz, 94Hz. These five clocks are for different speeds in different levels.

Input: clk (master clock)

Output: dclk, segclk, clk1, clk2, clk3, clk4, clk5

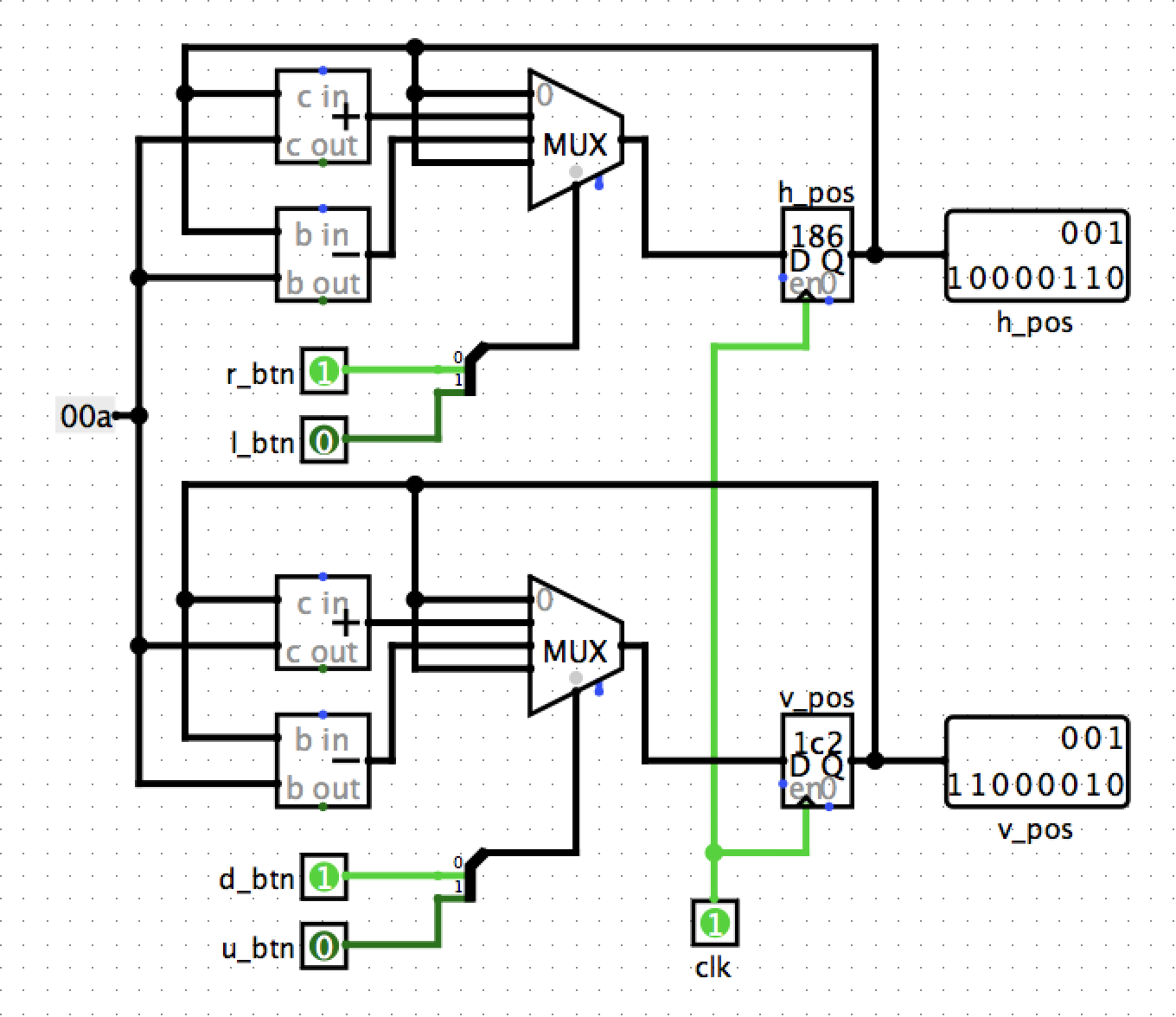
****

**3. paddle**

This module controls the position of paddle. Our paddle moves from leftmost point of the screen to the rightmost point. Also, because our ball always move in diagonal, the paddle can also move upward and downward to prevent the ball from moving in the same path all the time. The paddle can move upward till the half of the screen. Our paddle moves 10 pixels per press of button, which is faster than the ball. This ensures that the user can always catch the ball. The inputs include signal from 4 directional buttons. The outputs h\_pos and v\_pos are the horizontal and vertical position of the paddle, respectively. The point of reference of the paddle is the bottom left corner.

Input: clk, l\_btn, r\_btn, u\_btn, d\_btn

Output: h\_pos, v\_pos

****

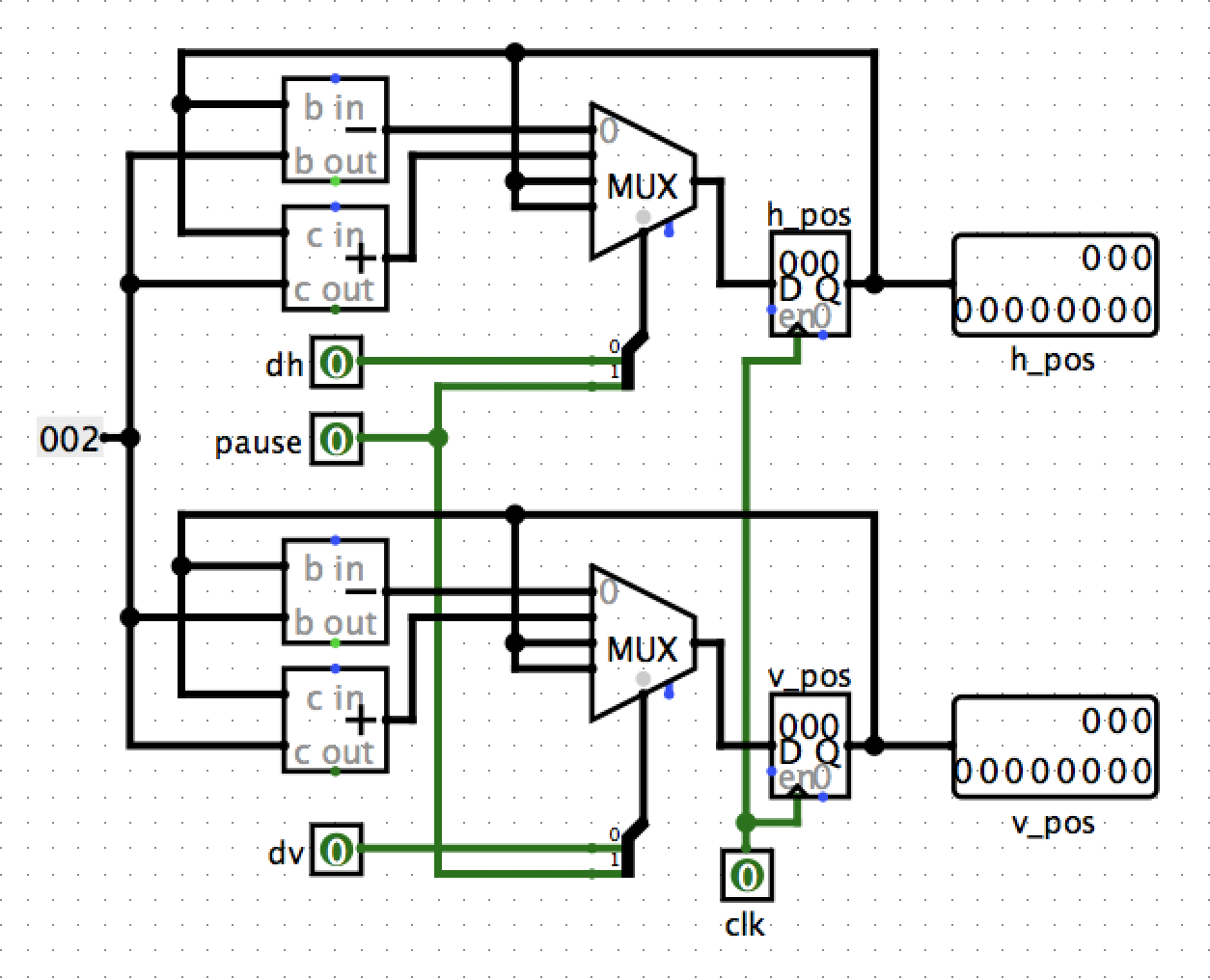
Schematic of paddle

**4. Ball**

This module controls the position and movement of the ball. The ball has two directional factors, horizontal(dh) and vertical(dv), with 4 combinations of up-right, up-left, down-right, and down-left. The controls of directions are in the top level module, since all the collisions are calculated there. The module will output the updated position of the ball, h\_pos and v\_pos, after each clock tick. When the game is in pause mode, the ball will stop moving until the pause switch is off.

Input: clk, rst, pause, dh, dv

Output: h\_pos, v\_pos

****

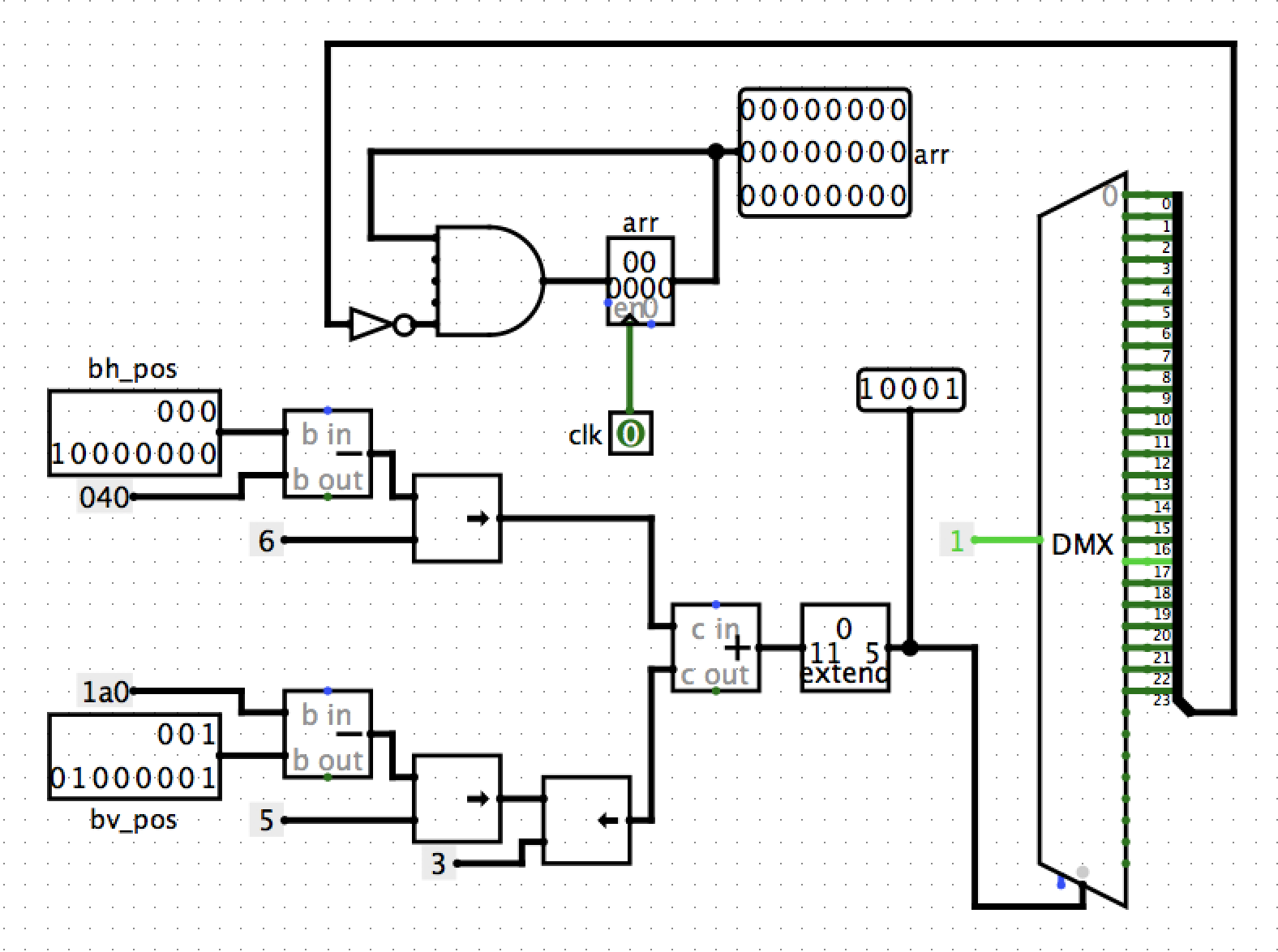
Schematic of ball

**5. bricks**

This module controls the initiation of the bricks, and the eliminations of bricks when the ball hits a brick. We create our bricks to be 3x8, with a total of 24 bricks. We use a 24 bit register (arr) to represent the status of each brick, 1 represents normal and 0 represents eliminated. The width of brick is 64 pixels and the height is 32 pixels. The division in Verilog is problematic, so we set these numbers to be the power of 2. We can then use shifting instead of division. The points of reference of each brick are the bottem left corner, the same as the paddle. The way we calculate the collision is simple. As long as the position of the ball (bv\_pos & bh\_pos) is within the boundary of a block, and the block is not eliminated yet, we eliminate that block.

Input: clk, bv\_pos, bh\_pos, rst

Output: arr (the array represents all bricks)

****

Schematic of bricks

**6. segdisplay**

This module controls the 7-segment display on the FPGA board. We use the first digit of the display to show the current level of the game, and the rest 3 digits for the score the player gets. This module is pretty much the same as the one we used in lab3. We use segclk, roughly 380Hz, to trigger the switching between each digit, since the board can only show one digit at a time. The input s1, s2, s3 represent the first, second, and third digit of the score. The input l represents the current level. The output seg represents the value of each segment of the display, and the output "an" controls which digit to be lit.

Input: segclk, s1, s2, s3, l

Output: seg, an

**7. vga640x480**

This module controls all the display on the screen. We use a very fast clock, dclk 25MHz, to iterate through each pixel of the display. In this module, the actual size of paddle, ball, and bricks are shown here. The size of the paddle is 100x20, the size of the ball is 5x5, and the size of each brick is 64x32, width by height. Also, we set the paddle to be white, the ball to be yellow, and the background is black. The color of bricks are calculated with respect to their coordinate, so we create a cool spectrum of bricks. Also, when calculating the movement of the paddle and the ball, the vertical axis is reversed. The vertical axis of VGA display is different from the ordinary mathematical axis, it is upside down. We have to adjust with that. The input paddle\_h, paddle\_v, ball\_h, ball\_v are the horizontal and vertical position coordinate or the paddle and the ball. barr is the array of the bricks. hsync and vsync are the horizontal sync out. red, green and blue are the RGB for each pixel.

Input: dclk, clr, paddle\_h, paddle\_v, ball\_h, ball\_v, barr, gameover,

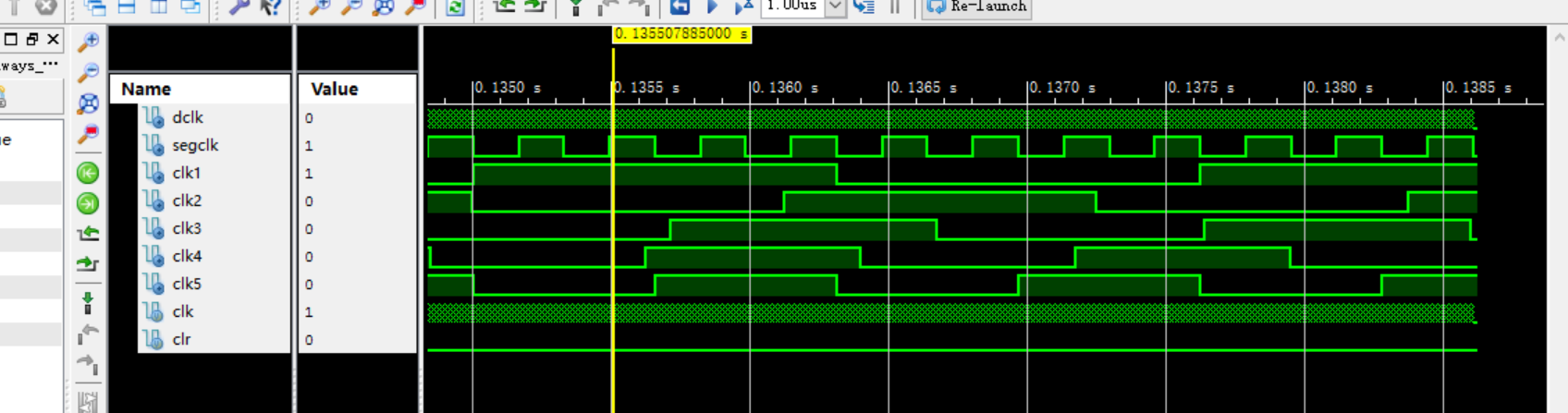
Output: hsync, vsync, red, green, blue

**Simulation Documentation**

Debouncer: We use the same module as the one in last lab.

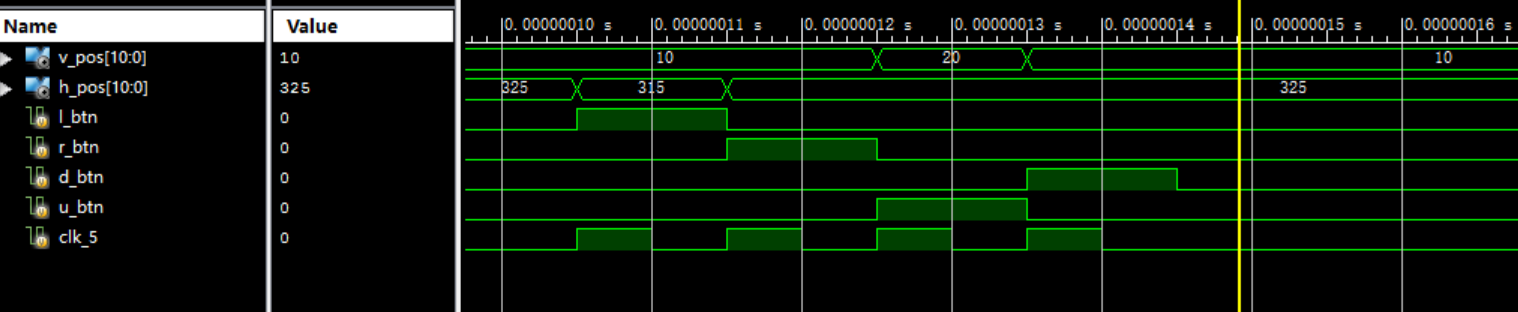
Segdisplay: We use the same module as the one in last lab.

**Clk\_divider module:**

****

This module is mainly used for providing different clocks derived from master clock. In order to do the simulation effectively, We shifted all counters right by three. For our game, we need 5 clocks for different levels of game. In addition to the segment clk, VGA clk, we also checked and test clk1-5 see if the frequencies are relatively correct.

**Paddle module:**

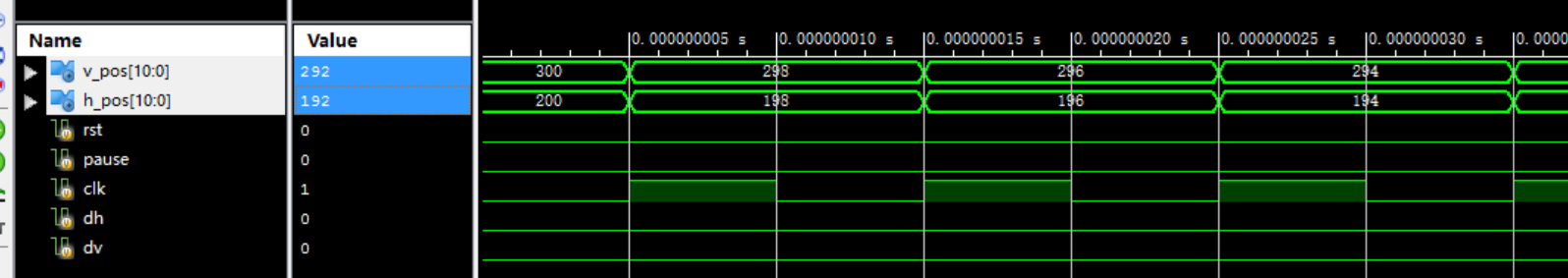


This module handles the movement of the paddle through the button pushed by player. Four buttons and corresponding coordinates changes are checked in a simplest case.(Boundary check is in top module).

**Ball module:**

This module handles the movement of the ball based on the directions passed in. The directions provided and corresponding coordinates changes are checked.(Boundary check is in top module)

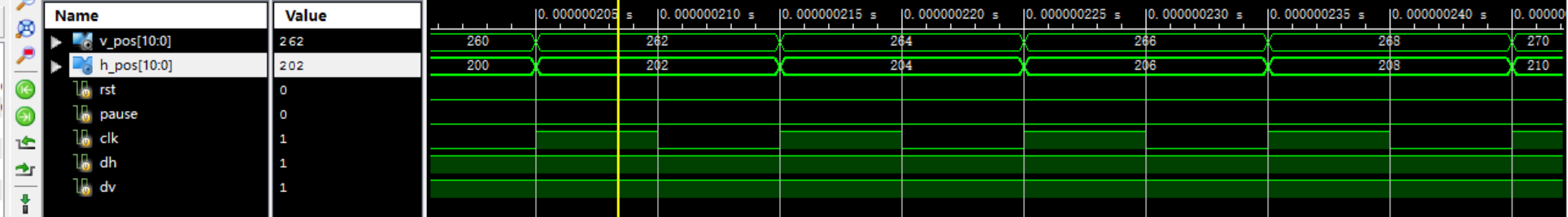
Down\_Left:



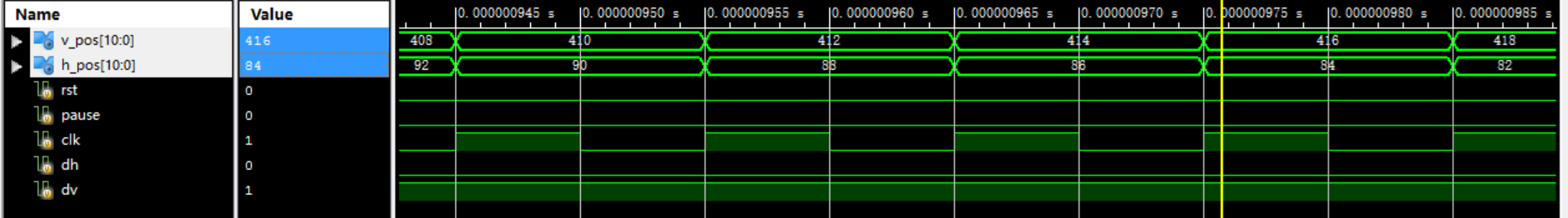
Down\_Right:

****

Up\_Right:

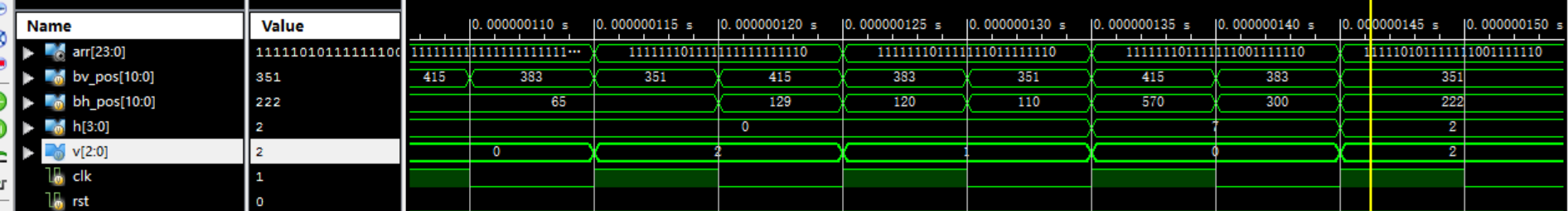
****

Up\_Left:



**Bricks module:**

This module contains information of bricks. The changes of status based on various ball positions are tested.

****

In this simulation,we can see the calculated indices and corresponding brick is set to zero.

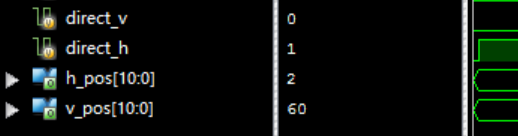
**Top module:**

It is hard to simulate this part of the project, since this part controls natural flows of the game. It will be complicated to play the game by writing code in testbench. Instead, we change the initial status of different modules to see if the module is handling each case correctly.

For example:

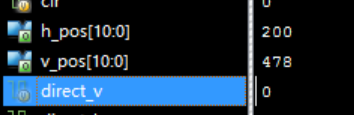
When ball hits the wall:

Left\_wall:

****

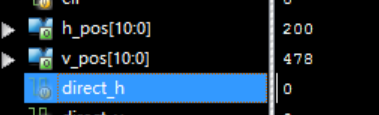
The direction is set to right.

Top wall:

****

The direction is set to down.

Right wall:



The direction is set to left.

There are more different situations which can be covered. However the waveform representation is much more limited here. For further tests, we have played games for many times. When bugs are found, we use the same method to generate waveform and then conduct debugging task.

**Bugs found:**

A lot of bugs were found in the process of building this game. One of them is that one of our teammate named variables in different modules with the same name, and there was not any notification. However, when the program compile, the circuit actually get messed up, and show unexpected behavior. Also, another bug was detected when we wrote the code for collision handling. We’ve tried different methods and finally settle to the current one, divided the entire game field with various blocks, and record the previous and current position of the ball to calculate the direction after collision.

**Conclusion:**

We have encountered several difficulties, including: having trouble keeping the paddle not moving out of the range. We set up a small margin between the wall and the paddle. So it won’t cross the boundaries. This is because, if we get a negative number, in verilog, it will overflow and become a positive number.

Another difficulty we encountered is that we can’t which directive to reflect the ball. Since there are 4 different possibility when we detect a collision . We tried several strategies and finally settle down for this one that divide that block areas into several sections, and track which section do we come from when we hit the brick.

Another huge problem is that we realize that division is not reliable in verilog, so we redesign our brick size to the power of 2 and use bit shift to achieve the same effect.

Improvements for the lab. We need to define a better interface to achieve modularity so we can test our code independently, and write simulation code early. We can also design more interesting bricks such as an explosive or a brick will slow down/speed up the movement of the paddle.

Lab4 is the first time we use verilog to design something that is interesting. We are very grateful for this opportunity to design an FPGA program that runs our favorite game in our childhood. We learned a lot of details about how FPGA functions throughout this lab. From solving the emerging problems in the lab, we are more informed about how hardwares and softwares are integrated together.