Electronic Systems Design

Design Document

Project Name: Cell Division — a brick-breaking game based on FPGA).

Faculty: School of Electronic Science and Engineering

Advisor: Prof. Yongming Tang

Group number: 24

Team Leader: Yuan Lei

Team members: Bozhi Wang, Guanglu Zhu, Jiayu Ren

Data: Dec. 14. 2019

(The document is machine translated from Chinese. Though I've done some proofreading, hope there's no more big typo or grammar issue. Thx for reading :p)

Contents

I. the system overview

1. Overview:

This project will use the VGA interface, DIP switch, ROM, buttons and parallel computing resources on Xilinx Basys3 to achieve a complete brick-and-tile game on the display.

2. Project Objectives:

(1) Learning objective: master the coding principle of the VGA interface, master the display principle of stationary and motion graphics, master the principle of motion control, and master the whole process of extracting, loading and displaying pictures with VGA display.

(2) Graphic Interface: The game background is pure black, the half of the screen is a rectangular brick with the overall layout style of "SEU", and there is a rectangular board that can be moved left and right in the middle of the bottom of the screen, and a small ball is placed on the board.

(3) Content and rules: After the game starts, the ball bounces freely in the screen with the set rules of movement. The player needs to catch the ball through the control rectangular board, and if the ball touches the bottom of the screen, the game fails. After the rectangular plate catches the ball, the ball changes its trajectory in a specular manner, and the game continues.

(4) Additional settings: The slider and the ball speed can be controlled, and the ball will trigger the splitting logic to split into three directions and small balls with different speed sizes after bouncing a certain number of times, destroying the bricks in one brain.

3. Main module:

(1) VGA encoding module
(2) Slider control module
(3) Digital tube coding module
(4) Crossover module
(5) Displays the module
(6) Three-ball state machine
(7) Ball motion logic module
(8) Collision logic determination module

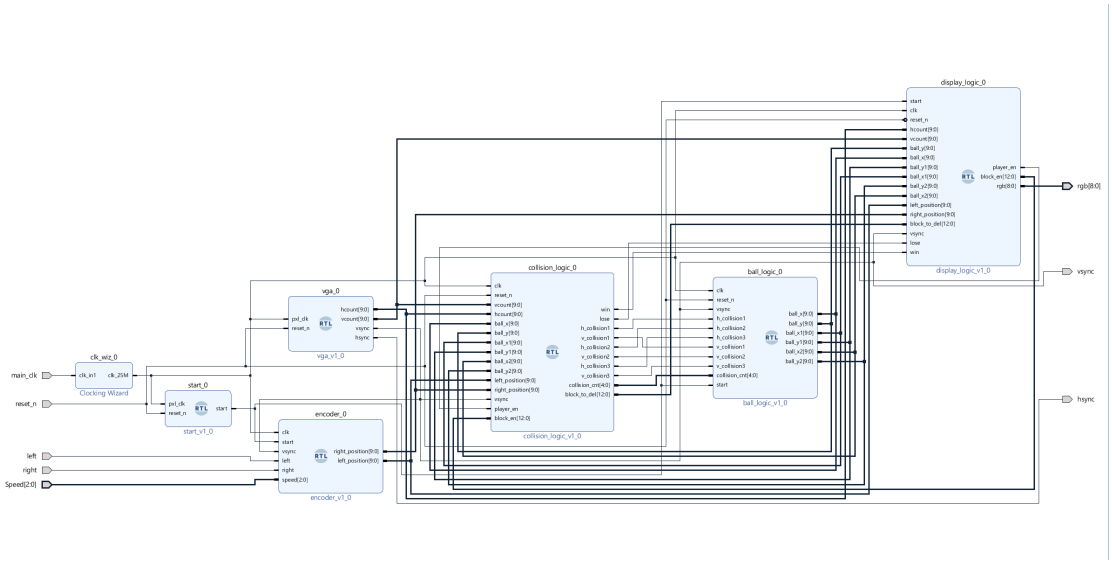
4. System top-level IO:

(1) Clock input port (main_clk): Use Basys3 with its own clock
(2) Left and right control ports (left, right): implemented using the built-in buttons

(3) Reset port (reset_n): 0 enables reset port
(4) Speed input port: Use four dip switches, two assigned to the slider speed and two assigned to the ball speed

II. the module design instructions

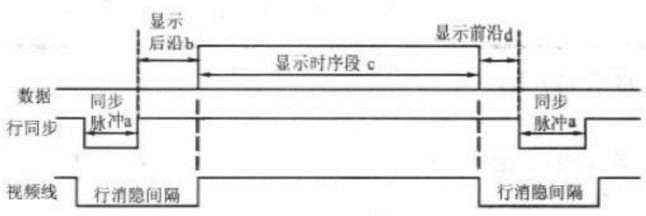
*. System Diagram:



1. VGA module

(1) Module function: Synchronize the program source code with the VGA interface and display in the same timing.

(2) I / O definition: the timing diagram of the VGA display is as follows. According to the VGA display principle, the electron gun scans each line progressively on the frequency screen, and after scanning, it will go to the beginning of the next line. After scanning to the bottom right corner, return to the starting position in the upper left corner, during this cycle, the electron gun variable 'hcount' +1 (line scan count) for each movement of



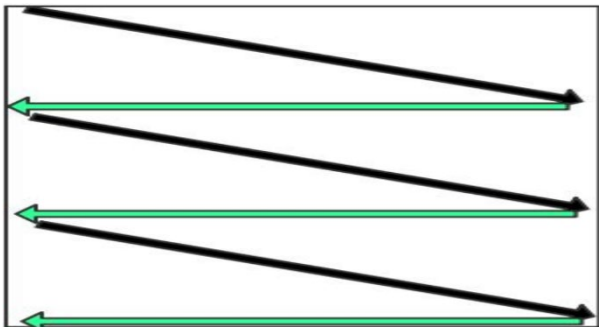


Figure 1: VGA timing and row and column scan illustrations

the electron gun. And after the scan, it returns to the beginning to synchronize to vcount +1 (field scan count), so that VGA will be Displays binary coordinates synchronized in code.

In particular, since the electron gun needs to skip the area for each line break and return to the starting point, and this time is not scanned. In this case, there will be a period of time vacated in the VGA scan timing (the front shoulder and rear shoulder of the scan), which corresponds to the two counting variables in the code. This value can be found in the VGA design manual according to different resolutions. The VGA display parameters in this project are 640*480 (60Hz), and the driving clock is 800*525*60Hz = 25.2MHz. Therefore, the condition that the output 'vsync' and 'hsync' of the program are valid (=1) on the two enable ends of the VGA is that the line is counted, and the field count is in the range, which is shown in the code

```
`define left_edge 8
`define right_edge 632
`define top_edge 8
`define bottom_edge 472

`define screen_left 0
`define screen_right 639
`define screen_top 0
`define screen_bottom 479
```

Figure 2: Define the start and end points of each pixel count with reference to the VGA standard.

```
always @ (hcount)
begin : hsync_decoder
    if (hcount >= 656 && hcount <= 752) hsync <= 0;

    else hsync <= 1;

end


always @ (vcount)
begin : vsync_decoder
    if (vcount >= 482 && vcount <= 492) vsync <= 0;

    else vsync <= 1;
end
```

Figure 3: The code counts the rows and columns scans

In summary, according to the definition of the function of the module, the input is a clock, the output hsync and vsync two enable terminals are the synchronous input pins required by the VGA, and the other two hcount and vcount count output variables. Other code modules are controlled as the primary coordinate variables of the code.

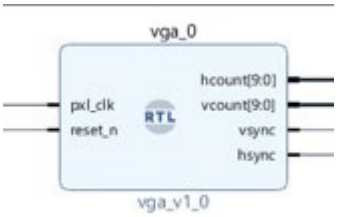


Figure 4: VGA module diagram

2. Crossover module

According to the official VGA standard, the 640*480 (60Hz) resolution display driver clock is 25MHz. Using Vivado's own Clocking Wizard can directly create a frequency divider IP. After reviewing, it is known that creating a phase-locked loop (PLL) can make the created clock more stable, remove the phase skew of the output clock, and avoid clock distribution delay. The input clock is the 100MHz clock that comes with the Basys3 and the output is 25MHz. The parameter settings and module diagrams are as follows:

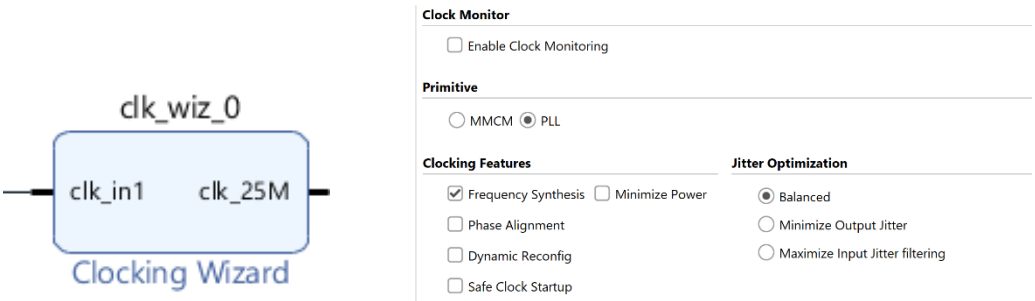


Figure 5: Frequency divider IP parameter setting and clock module diagram

3. Display start delay module

Since the signal has a certain delay in the transmission between different modules, if all the modules work at the same time when power-on may cause display or counting confusion, so we set a display delay module, let the display start synchronization between the other modules to start running, so as not to cause errors. The module uses a count count delay, and the logic is simple and will not be repeated.

```
always@(posedge pxl_clk or negedge reset_n)
begin
   if (!reset_n)
   begin
      start <= 0;
      count <= 0;
   end

   else if (count < 100000000)
   begin
      count <= count + 1;
      start <= 0;
   end

   else
   begin
      count <= count;
      start <= 1;
   end
end
```

Figure 6: Delay code implementation

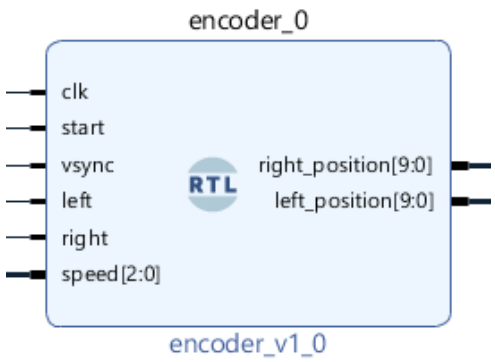
4. Sliding Block Control Module

(1) Design idea:

The sliding block in the game of bricking does not move in the vertical direction, so as long as the vertical starting position (defined as vcount==460 position in the project) and the slider height (defined as 10 units in the project), the specific position of the slider can be determined through the left and right boundaries of the slider.

(2) Logical algorithm implementation:

First define the initial left and right boundary positions in the module, and at the same time refresh the position of the left and right boundaries at the time of each field synchronization in an always, and directly use the initial left and right boundary positions to add or subtract the slider speed of the external input.




```
always@(posedge vsync)
begin
    if( a && (left_position >= `left_edge) )
     begin
       left_position <= left_position - speed;
       right_position <= right_position - speed;
     end

     else if( b && (right_position <= `right_edge))
     begin
       left_position <= left_po...  speed;
       right_position <= right_position + speed;
     end
end
```

output

Figure 7: Slider control module and code implementation

5. Ball Finite State Machine and Motion Logic Module

(1) IO definition:

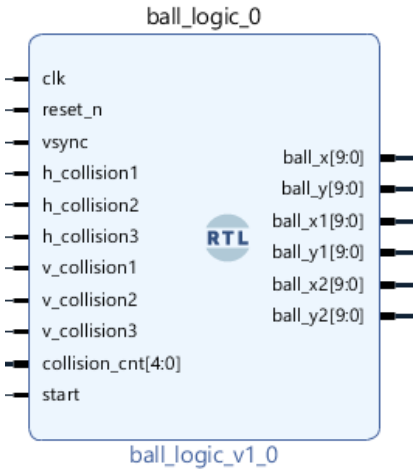


Figure 8, the state machine of the ball and the motion logic module

input port: clk (clock signal).
reset_n (reset).
vsync
h_collision1 (hit the left/right side of the brick).
h_collision2
h_collision3
v_collision1 (hitting the upper/lower side of the brick).
v_collision2
v_collision3
[4:0] collision_cnt (number of times the No. 1 ball hits the slider).
Start signal

Output port: ball_x (1 abscissa of the pellet).
ball_y (1 vertical coordinate of the pellet)
ball_x1 (sphere 2 abscissa).
ball_y1 (sphere 2 ordinate).
ball_x2 (3 abscissa of the sphere).
ball_y2 (3 ordinate of the sphere).

(2) Function overview:

This module is used to determine the direction in which the ball moves after a collision to achieve the following effects: hitting the block from ↘ the ball bounce ↗, hitting the block from ↙ the ball bounce ↖; hitting from ↗ then ball bounce off ↘; hit the block from ↖ then bounce to ↙. Functions also include change to sliding blocks and boundaries, as well as control of the initial position and velocity of the ball, and the realization of the split ball effect.

(3) Module design and logic determination:

1) Define 5 states: down_right = 3'b000; down_left = 3'b001; up_right = 3'b010; up_left = 3'b011; stop = 3'b100; respectively ↘; ↙; ↗; ↖; stop. The definition ball_state represents the state in which the pellet is located, the next_ball_state represents the next state of the sphere, and the ball_x, ball_y represent the horizontal ordinate coordinates of the pellet.

2) Implementation of the current state of the ball:

stop: ball_x, ball_y unchanged;
down_right: ball_x, ball_y increased;
down_left: ball_x decreases, ball_y increases;
up_right: ball_x increases, ball_y decreases;
up_left: ball_x, ball_y are reduced;
Default is defined as the initial state of motion.
Increasing and decreasing values control the speed at which the ball moves.
reset_n signal input to 0, reset the pellet position to the initial position.

(3) Implementation of the change of the collision state of the ball:

a. When the ball_state is stop, if the start input signal is 1, assign a value to the next_ball_state to start moving;

b. If the ball_state is down_right, the ball next_ball_state next_ball_state if it hits the lower boundary (ball_y > = 'bottom_edge)up_". If the ball hits the right boundary (ball_x > = 'right_edge), the next_ball_state is down_left. Hitting bricks changes in the same way.

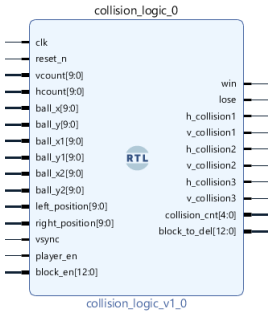
c. When the ball_state is down_left, if the ball hits the lower boundary (ball_y > = 'bottom_edge), it next_ball_state up_left If the ball hits the left boundary (ball_x < = 'left_edge), the next_ball_state is down_right. Hitting bricks changes in the same way.

d. When the ball_state is up_right, if the ball hits the upper boundary (ball_y < = 'top_edge), it next_ball_state down_right If the ball hits the right boundary (ball_x > = 'right_edge), the next_ball_state is up_left. Hitting bricks changes in the same way.

e. When the ball_state is up_left, if the ball hits the upper boundary (ball_y < = 'top_edge), it next_ball_state down_left If the ball hits the left boundary (ball_x < = 'left_edge), the next_ball_state is up_right. Hitting bricks changes in the same way.

(4) The implementation of the split-ball logic: after the No. 1 ball hits the slider a certain number of times, trigger a delay of a certain time to let the ball move a certain distance, and then assign the coordinates of the No. 1 ball to the No. 2 and No. 3 balls. At this point, the equivalent of two other balls is triggered, and the coordinates of the displayed logical balls 1 and 2 also begin to update continuously. ball_x1, ball_y1, ball_x2, and ball_y2 are the horizontal and vertical coordinates of the second and third spheres, respectively. When dividing the ball, set the decision condition of the collision_cnt == x of the split ball signal, and pass the ball_x, ball_y signal values through the temp_x temp_y give Ball 2 and Ball 3, ball 2 and ball 3 have the same movement and collision decisions as Ball 1.

6. Collision logic module



(1) Module function:

Determine separately whether the moving balls collide with the player's baffle, or collide with bricks and eliminate the corresponding bricks.

(2) Module design ideas:

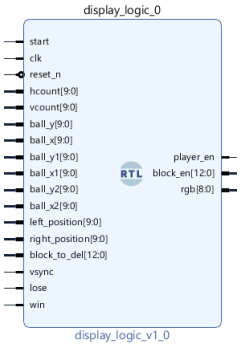
First, in the display module, in a scanning cycle, it is determined whether the hcount coordinates are within the coordinate range of the brick and the baffle, and in the range, the coloring is also controlled to control an enable signal (draw_player, block_en) to 1 The two enable signals generated in the display module are input into this module to give the judgment condition of whether the coordinates are in the range. In this module, if the if enabler is valid, it is determined at the same time whether the coordinates are within the range of the pellet, and if the coordinates are in the brick or baffle and the pellet at the same time, it is judged that the sphere has hit the baffle or brick at this time After the judgment encounters, a judgment enable end (block_to_del) is output to the display module, and when the enabling end is not touched in the display module, the bricks are displayed in color, and when the collision elimination enable end is effective, the brick color is changed to black and the same as the background color in the display module is displayed as elimination. At the same time, after judging the collision, it will also output the h_collision or v_collision the trajectory state machine of the ball to change the movement state of the ball according to whether the collision occurs on the vertical side or the horizontal edge.

(3) IO definition:

In summary, the specific input of the module: clock signal, judgment enable terminal [14:0] from the display module block_en and draw_player. The output is: the judgment to delete the enabling end [14:0] block_to_del and the module will subtract the total number of bricks set by 1 after not judging the deletion of a brick, and when the number of bricks is judged to be 0, the output win enable will judge the game victory. When it is determined that the bezel has not caught the ball (and when the ball is judged to hit the lower edge of the game boundary), the output loss enables the judgment game to fail, and both output ports are linked to the display module to control the game.

7. Display logic modules

1. Function Overview:



The main function of this module is to read the object position generated by other modules, perform logical processing of pixel display range and color, and output refresh enable RGA tricolor signal with VGA.

2. I/O Signal:

INPUT: pxl_clk (pixel clock), hcount (horizontal scan count), vcount (field scan count), ball_x (abscissa of the ball), ball_y (Ordinate of the ball), player_position (slider position), block_num (number of bricks), vsync (field synchronization signal), lose(failure decision), win (victory decision).

OUTPUT: rgb (signal for VGA tricolor line), player_en (slider refresh enable signal), block_en (brick refresh enable signal).

3. Module design ideas and logic algorithms:

(1) Key position and color global macro definition: Before implementing the display function, the most important thing is to define the width of each segment and the color of the tricolor line in the VGA timing with global macro variables so that the project content can be called. For details of the timing length parameters of each VGA at different refresh rates and resolutions, see the VGA module design document. RGB tricolor lines use three-digit binary encoding to define eight colors, of which each has four, controlling how many color lines are connected to control the shade of color.

VGA_RED	VGA_GREEN	VGA_BLUE	Resulting Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

Figure 9. Color defination of VGA

(2) Custom shape content display:

Since the display of VGA is a parallel logic that uses the phenomenon of human eye visual persistence, fixed content, such as boundaries, bricks, diagrams, etc., directly use **if-else** statements to hcount and vcount in the case of sensitive signals as clocks The counting signal calibrates a fixed area with or logic and assigns a value to the color output line rgb. Using if-else parallel statements causes a logical nature error that is not 'debugable'. An example is as follows:

```
always @ (hcount, vcount, ball_y, ball_x, player_position, block_num, lose, win, flash)
begin
    player_en = 0;
    block_en = 0;

    if (hcount > `screen_right || vcount > `screen_bottom) rgb <= 3'b000;

    else if (win) rgb <= flash;

    else if (lose) rgb <= `red;

    else if (vcount < `bottom_edge && hcount < `left_edge) rgb <= `green;

    else if (vcount < `bottom_edge && hcount > `right_edge) rgb <= `green;

    else if (vcount < `top_edge ) rgb <= `green;

    else if (vcount > `bottom_edge) rgb <= `red;
```

The description of the shape of the sphere is determined using the planar equation of the circle in the analytic geometry, as follows:

$$(x-a)^2+(y-b)^2=r^2$$

```
always @(*)
begin
    distance = (hcount - ball_x) * (hcount - ball_x) + (vcount - ball_y) * (vcount - ball_y);
end
```

Add the following judgment conditions to the if-else statement of the scope judgment:

```
else if (distance < 9) rgb <= `white;
```

The display of other geometries uses the knowledge of planar analytical geometry and so on, and will not be repeated.

(3) Picture display

In this project, to achieve the display of the logo of the school emblem, it is necessary to use the MATLAB program to extract the RGB color of the JPG picture to obtain the RGB pixel arrangement matrix, and the ROM IP is initialized to be available coe file.

The main functions of MATLAB to achieve pixel extraction is bin2dec function. This function converts the binary position of each RGB pixel read to decimal.

The RAM IP is then initialized to form a RAM that stores the VGA picture display signal for call at any time

(4) Screen flashing effect display

First set a blink trigger enable signal, when the signal is 1 starts an 8-clock delay signal, the first 4 clock cycles assign RGB to white, the last four-cycle clocks set RGB to red, and then described above The if-else statement makes a judgment on the win-loss condition and outputs the RGB to the VGA tricolor.

III. Conclusions

In this project, we have mastered the underlying hardware understanding of VGA in FPGAs and the principles of graphic picture display and motion control, especially the parallel computing characteristics with FPGAs. At the beginning of the actual project, we dug a big hole for ourselves. Due to the lack of deep understanding of parallel logic, most of the multi-ball FSM and collision determination are written using a serial logic idea, which can only fit for a single ball. And there is no concern about warning in synthesis and implementation, which leads to many unexpected bugs in the later stage. After writing the various modules at the beginning, we considered encapsulating the FSM and collision logic of the single ball into IP to achieve code reuse, but there were too many BUGs after calling the IP, and due to a momentary panic, we decided to integrate the contents of the three balls into one module. The next step involves modifying the same register in multiple ways in a module, but this is not allowed in the Verilog logic. As a result, we have difficulties in synchronizing multiple signals at different levels with the new clock, resulting in many state refresh delays, and competition-risk phenomena are obvious (such as when the same brick is hit by multiple balls, block_to_del signals and block_en show out-of-sync issues with signal updates).

At the end of the course, this project still has a lot of room for expansion, such as in-depth learning of IP calls, using RAM to store and delete the address of the bricks, so that the display of bricks will be much simpler, and many complicated enable signals can be omitted; different brick layouts can be set, different slider length levels can be set, different playthrough animations can be set; and also the state machine can be implemented in MicroBlaze IP, which is much more flexible than writing the state machine directly with verilog; you can set the option to adjust the resolution and optimize the display at different resolutions, after all, it is now the era of 1080P Full HD.