ECE 385

Fall 2022

Final Project

Pico Park

Ziyue Guo

Yuntong Gu

12/13/2022

TA: Tianhao Yu

1. Introduction

1.1 Overall Introduction

We create a game which is similar to the existing game Pico Park released by Microsoft Windows via video game retailer Steam. Pico Park is a cooperative multiplayer, action-puzzle independent game. We can control the blue character and red character to jump, go left, go right in the game. The goal of the game is to get through the obstacles, trigger the platforms and unlock the locked door at the end of the map. Players need to cooperate to reach the objective, but players also have the ability to prevent reaching this goal, for example by blocking other players' movement.

We design and implement the project on the FPGA board (intel MAX10) using SystemVerilog HDL. Moreover, we also implement a character controller, color mapper and VGA controller together with the controller of the barriers in the game. Using the NIOS II CPU, we will implement a USB interface thus allowing us to use the keyboard to interact with the game(like lab6). Final demo is carried out by using USD keyboard, FPGA board and a VGA monitor.

1.2 How to Run the Game

The first step is compiling the codes for both the hardware and the software. After doing this, a starting page will appear on the screen.

The second step is to press “Enter” button on the keyboard to start the game. The blue and red character will fall down to the ground.

The third step is to control the movement of the characters. For the blue one,”H”,”K”,”U” can control it to go left, right and jump. Similarly, for the red one, “Left”,”Right””Jump” can control it to go left, right and jump.

The forth step is to go through the obstacles. The characters are supposed to jump over the trap where they might falls down. They can also jump on the another one’s head and then jump for the wider trap. They can also stand on the red button to control the platform to move left. Finally, they can also trigger the lift by both standing on it.

The last step is go through the door and end the game. If anyone of them reach the area of the door and press ”J” for the blue one and “Down” for the red one, the game will end and return to the starting page.

2. Written Description

2.1 Description of the overview of the circuit

In general, we build our final project based on lab6.2 which we can control a red ball move in the screen and bouncing between the boarders. In lab6.2, we design SOC with USB and VGA interface which is also fundamental and vital in our project. On the basic of it, we design a file named finaltop.sv as the top-level entity of the circuit. We also add a lot of components into it to achieve our goal including the character control unit, the background unit, the barriers unit and so on.

2.2 Hardware components of the lab

The most of the hardware in this lab are created in the Platform Designer. We also connected them by buses and lines in this platform. The pins can also be assigned as input and output export in this platform. The main component is the NIOS II processor which acts as a high-level interface in the lab. We also add on-chip, off-chip memory, PIO, HEX Driver, LEDs, PLL, Clk and so on in our project. The detailed description of the hardware is given in the section later.

2.3 How the I/O works

We use the I/O interface through the NIOS II processor quit similar with lab 6.2. We connect the signals declared in the top module. On the platform designer, we instantiate a Parallel Input/Output module (PIO) in which the corresponding device is assigned a memory address. This module achieve the goal to connect the Avalon bus and the FPGA to input and output data we require.

2.4 Description of the general flow of the circuit

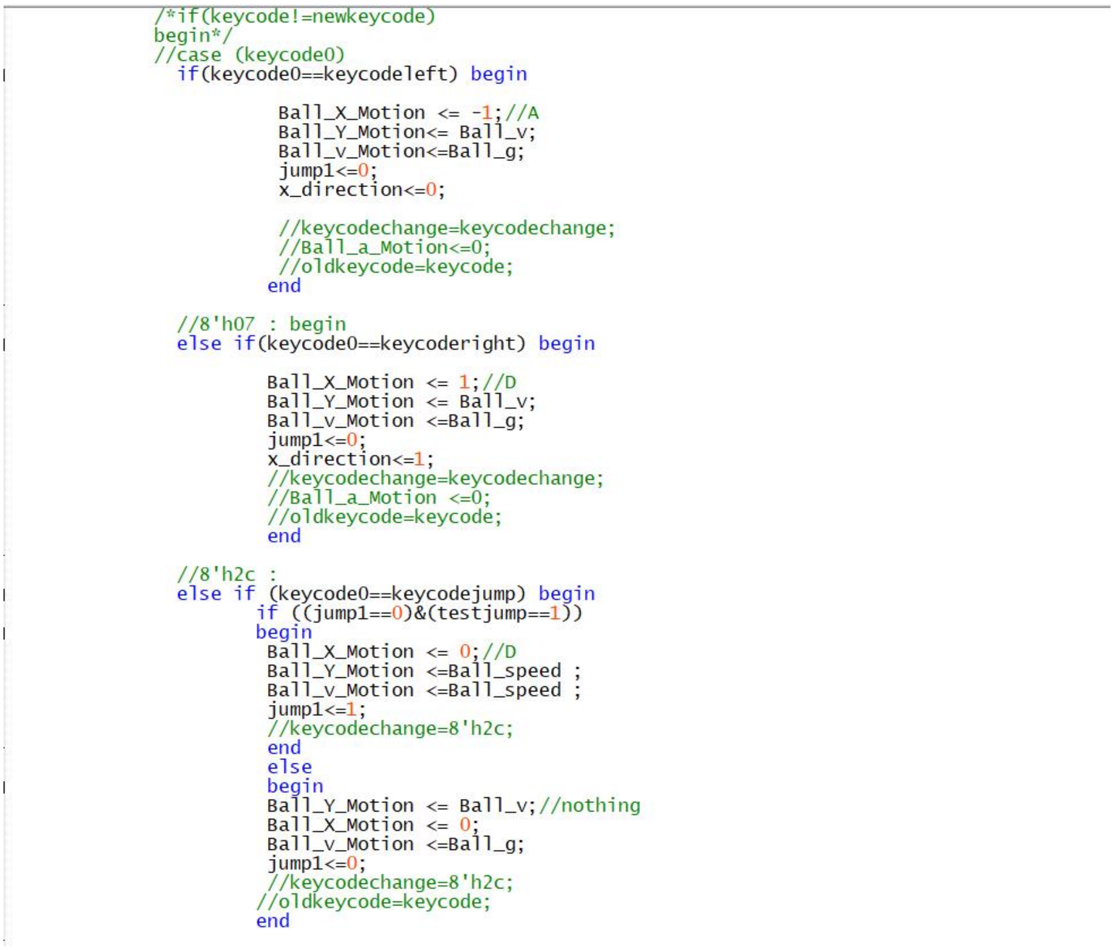
As we can see in the top-level file, the inputs are system clock, keys on the FPGA board and so on. The outputs are the HEX displays, the LEDs, outputs related to VGA, outputs related to the frame RAM and so on.

In details, we use a clock that connects all the components, a NIOS II economic version processor that reads and runs the instructions, an on-chip memory, SDRAM as the off-chip memory and a second clock which goes out to the SDRAM chip 1ns behind of the controller clock which is important for the RAM to capture and store the correct data. We create the JTAG UART which allows us to print to console using printf while debugging and to communicate with the NIOS II. The USB chip features as SPI communication protocol interface. Therefore, we use SPI IP to read and write to the target registers on the USB chips. When it comes to the VGA component, it provides the VGA hardware with the keycodes from the USB device. In this way, we can determine what to be shown on the VGA monitor.

3. List of Features

3.1 Basic operation of characters.

First of all, the basic operation of the character includes two parts: moving and jumping. The movement includes moving to the left and moving to the right. In the process of moving the character, we also added the function of avoiding obstacles, that is to say, when the character meets obstacles in the process of movement, it will stop. In addition, the character can also jump. When jumping, the character will generate downward acceleration according to the pattern of acceleration of gravity, which can make the character's jumping leisure more real, and will stop when encountering obstacles in the process of rising or falling. The principle and some code screenshots are as follows:



```
            /*if(keycode!=newkeycode)
            begin*/
            //case (keycode0)
              if(keycode0==keycodeleft) begin

                       Ball_X_Motion <= -1;//A
                       Ball_Y_Motion<= Ball_v;
                       Ball_v_Motion<=Ball_g;
                       jump1<=0;
                       x_direction<=0;

                       //keycodechange=keycodechange;
                       //Ball_a_Motion<=0;
                       //oldkeycode=keycode;
                      end

              //8'h07 : begin
              else if(keycode0==keycoderight) begin

                      Ball_X_Motion <= 1;//D
                      Ball_Y_Motion <= Ball_v;
                      Ball_v_Motion <=Ball_g;
                      jump1<=0;
                      x_direction<=1;
                      //keycodechange=keycodechange;
                      //Ball_a_Motion <=0;
                      //oldkeycode=keycode;
                      end

              //8'h2c :
              else if (keycode0==keycodejump) begin
                     if ((jump1==0)&(testjump==1))
                     begin
                      Ball_X_Motion <= 0;//D
                      Ball_Y_Motion <=Ball_speed ;
                      Ball_v_Motion <=Ball_speed ;
                      jump1<=1;
                      //keycodechange=8'h2c;
                      end
                      else
                      begin
                      Ball_Y_Motion <= Ball_v;//nothing
                      Ball_X_Motion <= 0;
                      Ball_v_Motion <=Ball_g;
                      jump1<=0;
                      //keycodechange=8'h2c;
                     //oldkeycode=keycode;
                     end
```

The core part of the code adopts the principle of ball movement in lab6.

The main principle of obstacle avoidance is to read the color signal of its next position. If the color corresponding to the signal is an obstacle, a stop signal will be fed back to the ball module, so that the figure will stop its next movement.

We also have our own animation for each character. When the character is on the ground, we will change its orientation according to its movement direction, and when jumping, we will show the movement of jumping left and right according to its movement direction. The realization principle of this part is that we store the different actions of the figure as different pictures in ram, and according to the different motion states of the figure, the figure module will give different control signals, so that the ram module can decide which picture to read.

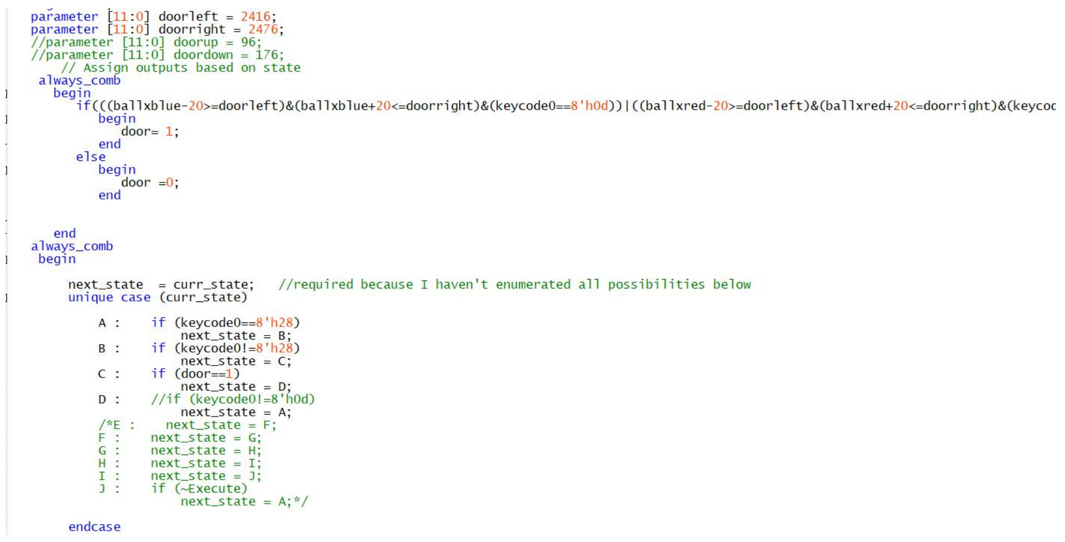
3.2 Multiplayer game module.

In this final project, we added the module of multiplayer game. Based on the basic movement of two people, we also added the interactive logic between two people. When two people intersect or collide in the horizontal or vertical direction, they will keep A static state in order to avoid mold penetration, which enables character A to stand on the head of character B. In addition, when character A stands on the head of character B, A can move freely on B's head and move with B in the horizontal direction.

3.3 Game scenarios

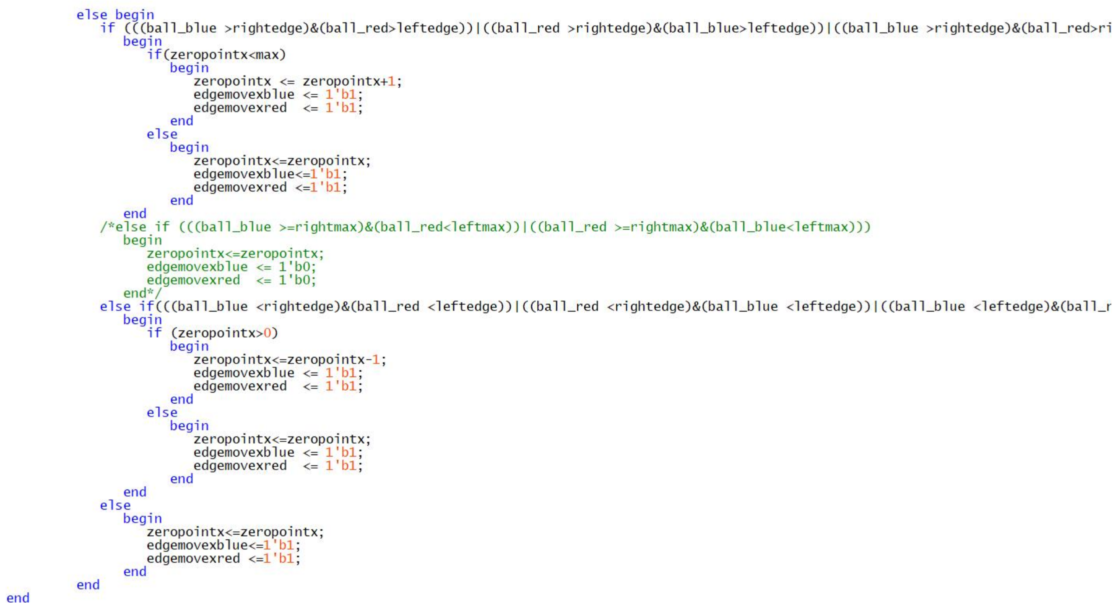
Our game scene has two, the main menu at the beginning of the game and the level interface of the game. The main menu is a static picture of 640*480. When we press the enter key as instructed, we can switch to the level interface. This is a long picture of 2560*480.

To achieve scene switching we added a finite state machine. We controlled the state switching to determine whether the background image was the main menu or the level scene. The code for the state machine is as follows:



```
    parameter [11:0] doorleft = 2416;
    parameter [11:0] doorright = 2476;
    //parameter [11:0] doorup = 96;
    //parameter [11:0] doordown = 176;
        // Assign outputs based on state
     always_comb
       begin
          if(((ballxblue-20>=doorleft)&(ballxblue+20<=doorright)&(keycode0==8'h0d))|((ballxred-20>=doorleft)&(ballxred+20<=doorright)&(keycoc
             begin
                door= 1;
             end
          else
             begin
                door =0;
             end


       end
    always_comb
     begin

         next_state  = curr_state;   //required because I haven't enumerated all possibilities below
         unique case (curr_state)

             A :    if (keycode0==8'h28)
                        next_state = B;
             B :    if (keycode0!=8'h28)
                        next_state = C;
             C :    if (door==1)
                        next_state = D;
             D :    //if (keycode0!=8'h0d)
                        next_state = A;
             /*E :    next_state = F;
             F :    next_state = G;
             G :    next_state = H;
             H :    next_state = I;
             I :    next_state = J;
             J :    if (~Execute)
                        next_state = A;*/

         endcase
```

However, since the screen can only display the range of 640*480, we added the module of screen scrolling. The screenshot of part of the code is as follows:



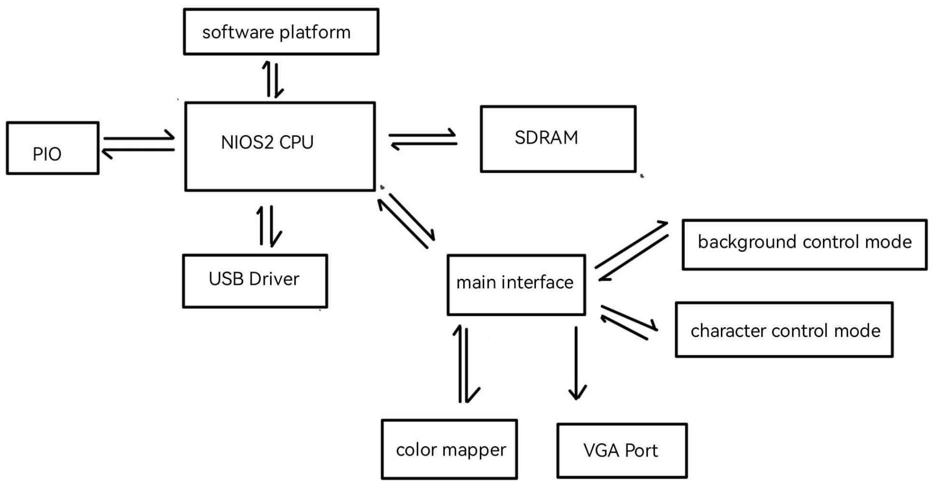
```
         else begin
            if (((ball_blue >rightedge)&(ball_red>leftedge))|((ball_red >rightedge)&(ball_blue>leftedge))|((ball_blue >rightedge)&(ball_red>ri
               begin
                  if(zeropointx<max)
                     begin
                        zeropointx  <= zeropointx+1;
                        edgemovexblue  <= 1'b1;
                        edgemovexred   <= 1'b1;
                     end
                  else
                     begin
                        zeropointx<=zeropointx;
                        edgemovexblue<=1'b1;
                        edgemovexred <=1'b1;
                     end
               end
            /*else if (((ball_blue >=rightmax)&(ball_red<leftmax))|((ball_red >=rightmax)&(ball_blue<leftmax)))
               begin
                  zeropointx<=zeropointx;
                  edgemovexblue  <= 1'b0;
                  edgemovexred   <= 1'b0;
               end*/
            else if(((ball_blue <rightedge)&(ball_red <leftedge))|((ball_red <rightedge)&(ball_blue <leftedge))|((ball_blue <leftedge)&(ball_r
               begin
                  if (zeropointx>0)
                     begin
                        zeropointx<=zeropointx-1;
                        edgemovexblue  <= 1'b1;
                        edgemovexred   <= 1'b1;
                     end
                  else
                     begin
                        zeropointx<=zeropointx;
                        edgemovexblue  <= 1'b1;
                        edgemovexred   <= 1'b1;
                     end
               end
            else
               begin
                  zeropointx<=zeropointx;
                  edgemovexblue<=1'b1;
                  edgemovexred <=1'b1;
               end
         end
end
```

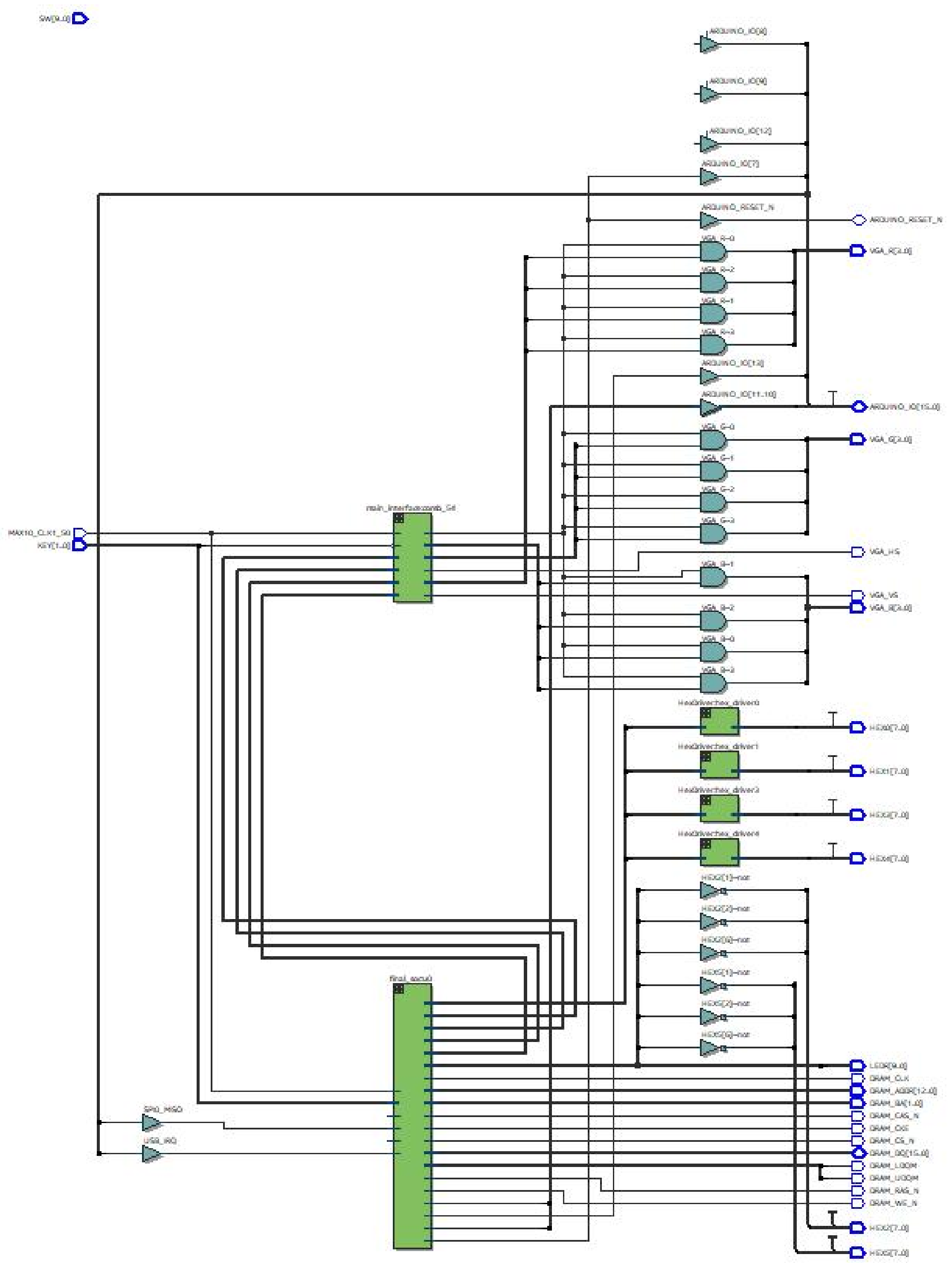
The principle we use here is to add an initial point variable that determines which position in the long graph we are reading from at this moment. When any character from the left exceeds our threshold, the initial point variable will become smaller, and if not, it will become larger. As the initial point variable changes, the scope of the background image we read will change, which makes the screen scroll.

3.4 Obstacles and Platforms

In order to increase the difficulty of the game, we added some obstacles in the game, the first is the lifting platform. When two characters walk on the lifting platform at the same time, the platform will fall; otherwise, it will rise. The design principle of this part of the code is to judge the lower endpoint coordinates of the characters through the if statement and compare with the endpoint coordinates of the platform, so as to output control logic signals to the movement of the characters. There is also a flex platform. The logic of this platform is that when a character presses a button on the right side of the platform, the platform moves to the left until it reaches the left wall.

4. Block Diagram





5. Module Description

finaltop.sv

Input: MAX10_CLK1_50, [1: 0] KEY, [9: 0] SW

Output:[9: 0] LEDR,[7: 0] HEX0, [7: 0] HEX1, [7: 0] HEX2, [7:0] HEX3, [7:0]HEX4, [7:0] HEX5, DRAM_CLK, DRAM_CKE, [12: 0] DRAM_ADDR, [1:0]DRAM_BA, DRAM_LDQM, DRAM_UDQM, DRAM_CS_N, DRAM_WE_N,DRAM_CAS_N, DRAM_RAS_N, VGA_HS, VGA_VS, [3: 0]VGA_R, [3: 0]VGA_G, [3: 0] VGA_B,

Inout: [15:0]DRAM_DQ,[15:0] ARDUINO_IO, ARDUINO_RESET_N

Description: This is the top-level file of lab7, which integrates all the functions of the hardware part of lab7 and communicates with the software part.

Purpose: It encapsulates the parts as a top-level file.

RTL Diagram: This part is given in the previous section.

VGA_controller.sv

Input: Clk, Reset

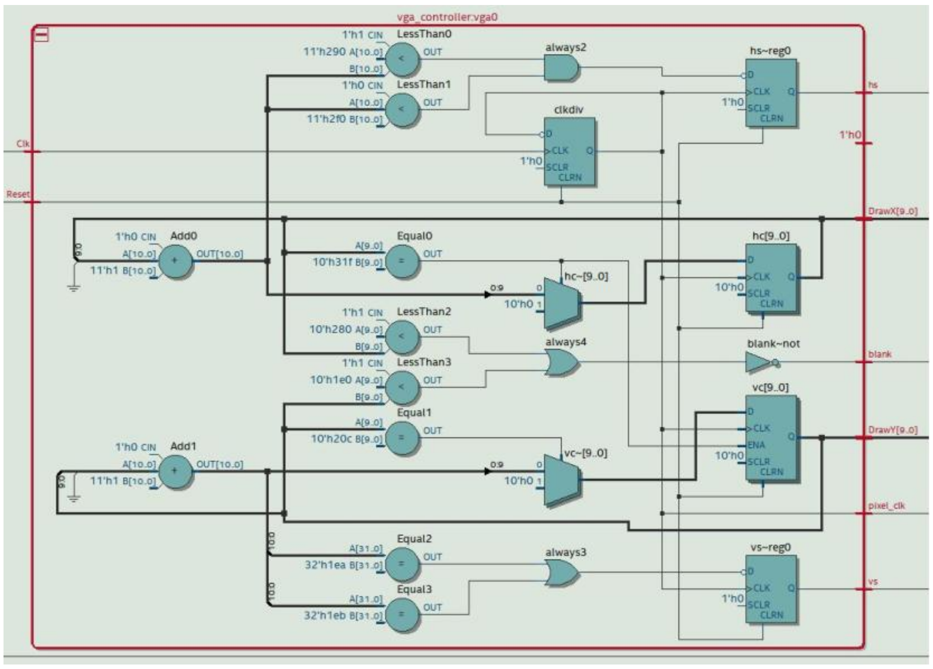
Output: hs, vs, pixel_clk, blank, sync, [9:0] DrawX, DrawY

Description: This module is mainly used to control the output of vga signal. It can decide the signals of each part of VGA according to the output of the ball module. drawX and drawY represent the pixels at this time, while RGB output is its color.

Purpose: The function of this module is to adjust the ball module, and provide the scanning frequency of pixels and the scanning position at this time. His output VS and hs are both used as the output of the final

VGA signal.

RTL Diagram:



HexDriver.sv

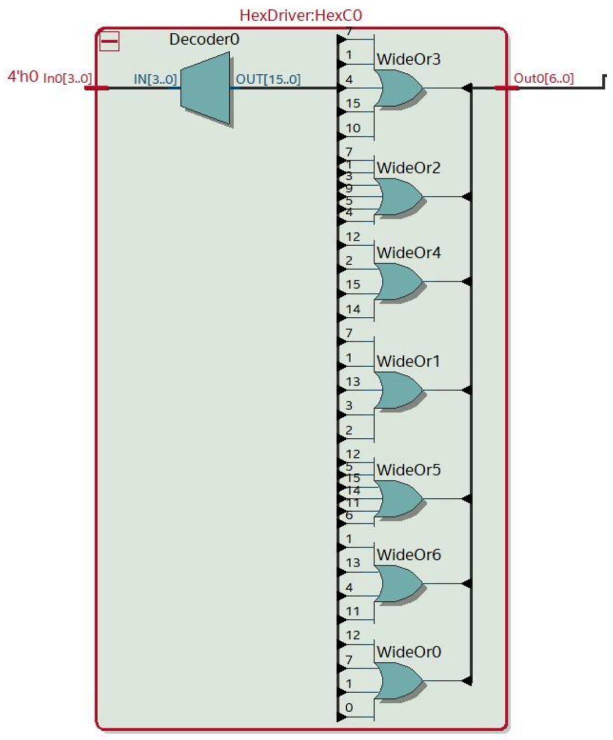
Inputs: [3:0] In0

Outputs: [6:0] Out0

Description: This is a 4-bit Hexdriver which can modify the input value to a hexadecimal number.

Purpose: This module is used to change the output so it can be viewed on the On-Board seven-segment display segment.

RTL Diagram:



main_interface.sv

Inputs: Clk, Reset, [7:0] keycode0, keycode1, keycode2, keycode3,

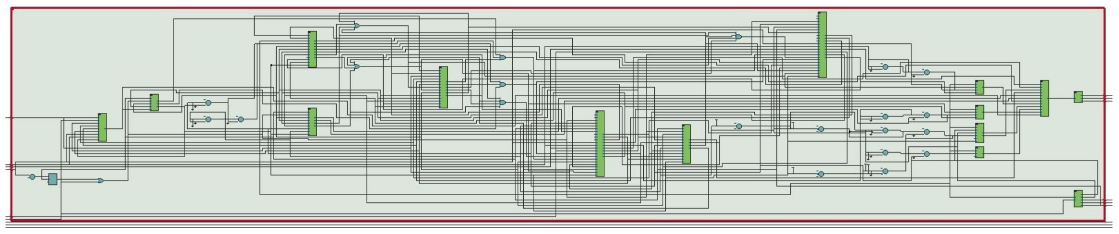
Outputs: [7:0] red, green, blue, hs, vs, blank

Description: In this module, we almost contain all the modules in the project. We declare a number of module in this parts together with some important signals.

The details is as below: the size of the character, drawx, drawy for VGA and the signals given by the ball.sv which controls the basic movement of the characters, the signals controlling the scrolling of the screen. Especially, there are some signals that are really important --bluexmove, redxmove, blueymove, redymove, jump_blue and jump_red which control whether the character can achieve the basic movement.

Purpose: This module is the main interface of the whole project which combines the remaining modules and important signals.

RTL Diagram:



frameRam.sv

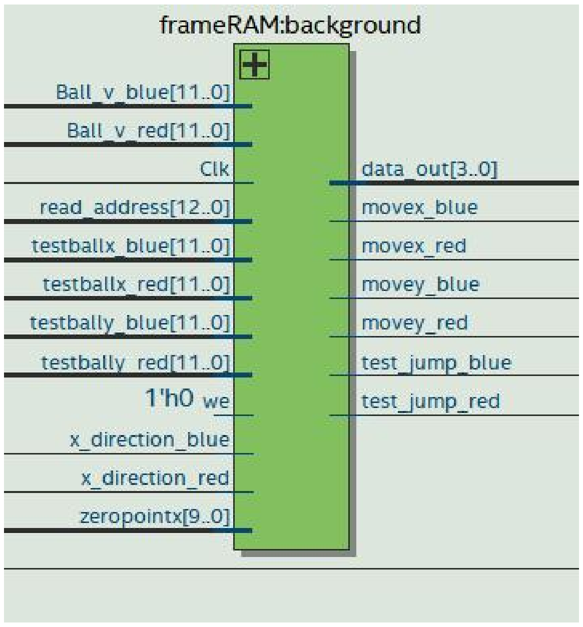
Inputs: [3:0] data_in, [11:0] testballx_blue,testbally_blue,Ball_v_blue, [11:0] testballx_red,testbally_red,Ball_v_red, [12:0]write_address, read_address, logic we, Clk, x_direction_blue, jump_blue, x_direction_red, jump_red, [11:0] ballxsig_blue, ballysig_blue, ballxsig_red, ballysig_red, [9:0] zeropointx,

Outputs: [3:0] data_out, logic test_jump_blue,test_jump_red, logic movex_blue,movey_blue,movex_red,movey_red

Description: In this module, we create a memory contains the information of the background of the project which is stored in a txt file. And then we create a structure to read and write to the memory. Also we use some signals like test_up, test_down, test_left, test_right_jest jump to control the character not to go into the ground and walls. Test_jump is 1 when the character is standing on the background and there is nothing above its head.

Purpose: This module provides the information of the background and the relationship between the characters and the background.

RTL Diagram:



Character1RAM.sv

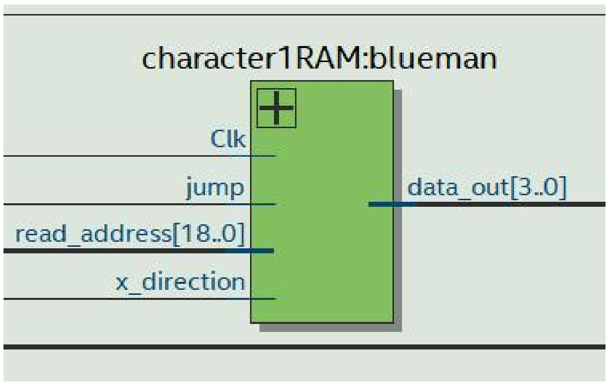
Inputs: [3:0] data_in, [18:0]read_address, Clk,Reset, logic x_direction, jump

Outputs: [3:0] data_out

Description: In this module, we create four memories to store the information of the characters with different states including stand left right and jump left right. We use the input signal jump and x_direction which refers to the direction of movement to control the switch of the states.

Purpose: We store the image of the blue character and control the switch of the movement of the character.

RTL Diagram:



Character2RAM.sv

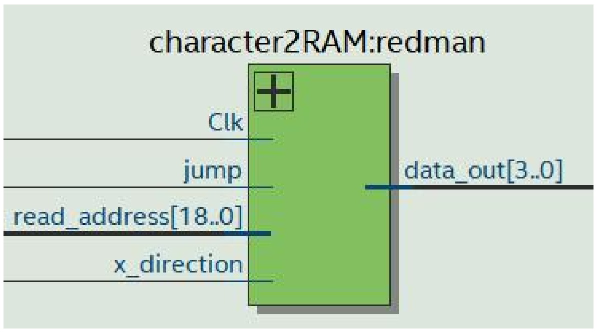
Inputs: [3:0] data_in, [18:0]read_address, Clk,Reset, logic x_direction, jump

Outputs: [3:0] data_out

Description: In this module, we create four memories to store the information of the characters with different states including stand left right and jump left right. We use the input signal jump and x_direction which refers to the direction of movement to control the switch of the states.

Purpose: We store the image of the red character and control the switch of the movement of the character.

RTL Diagram:



character_relation.sv

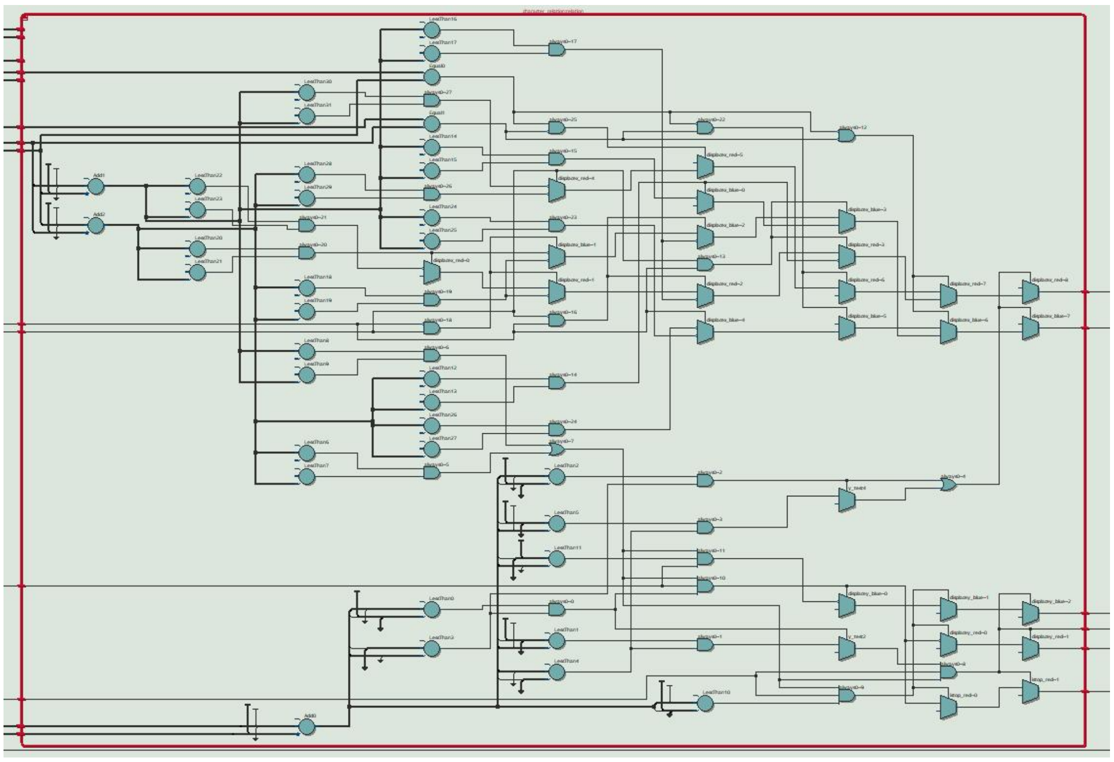
Inputs: [11:0] testballx_blue, testbally_blue, Ball_v_blue, [11:0] testballx_red, testbally_red, Ball_v_red, [11:0] ballxsig_blue, ballysig_blue, ballxsig_red, ballysig_red, logic x_direction_blue, test_jump_blue, x_direction_red, test_jump_red

Outputs: logic displacex_blue, displacex_red, displacey_blue, displacey_red, istop_blue, istop_red

Description: In this module, we use the positions of the two character to control the relationship between the two characters. If the two characters has intersection in y direction, we should not let one character walk through another one's body, we use displacex to control it. Also, we determine and give out a signal is_top to justify whether one character is on another one's head.

Purpose: We use positions of the two characters to control the relationship of the characters and let one character stand on the other's head come into happen.

RTL Diagram:



frameRAM_graph.sv

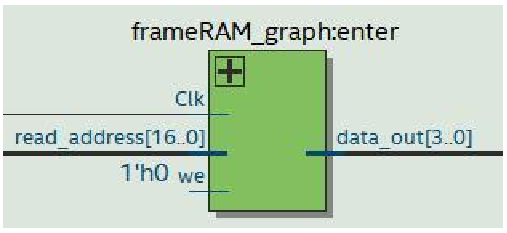
Inputs: [3:0] data_in, [16:0]write_address, read_address, we, Clk

Outputs: [3:0] data_out

Description: We create a memory to store the image of the starting page.

Purpose: This module stores the image information of the starting page.

RTL Diagram:



frameRAM_door.sv

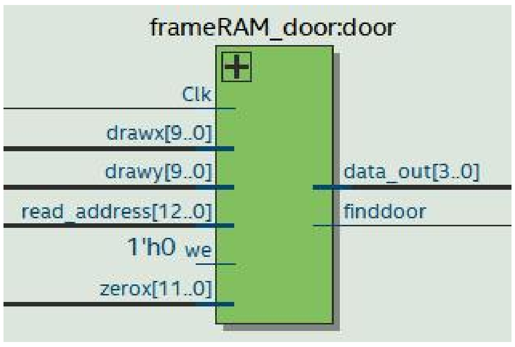
Inputs: [3:0] data_in, [12:0] write_address, read_address, [11:0] zerox, [9:0] drawx, drawy, we,Clk

Outputs: [3:0] data_out, finddoor

Description: In this module, we create a memory stored the image information of the door. Also we use the zero point of the screen and the size of the door to determine whether a pixel is in the area of door or not.

Purpose: This module stores the image of the door and determine the area of the door.

RTL Diagram:



leftmoverectangle.sv

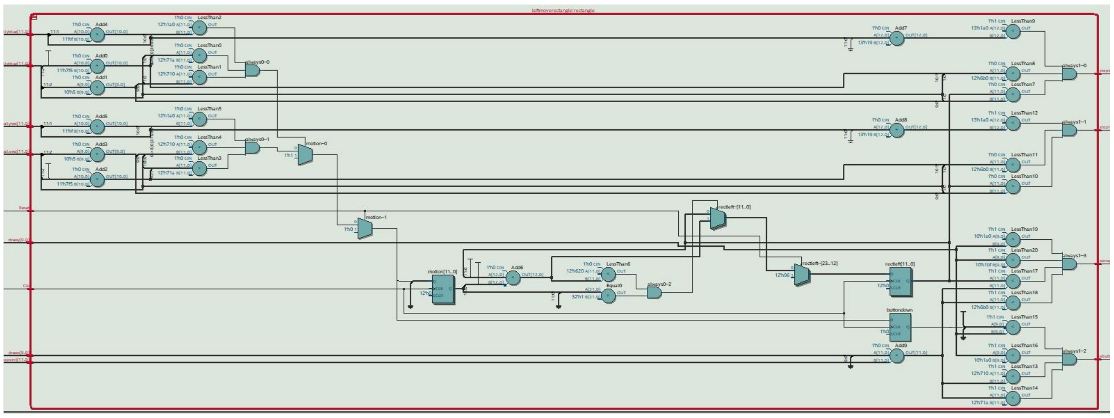
Inputs: [11:0] ballxblue, ballxred, ballyblue, ballyred, Clk, Reset, [9:0] drawx, drawy, [11:0] zeropoint

Outputs: isinrectangle, isbutton, stopblue_y, stopred_y

Description: This module contains the information of the first obstacles in the game, the moving left platform. After pressing the red button in the specific position and set the isbutton signal to 1, the platform will move left. We set the start position and stop position of the rectangle in this module.

Purpose: This module contains the information of the left moving platform and control the behavior of the obstacles.

RTL Diagram:



platform.sv

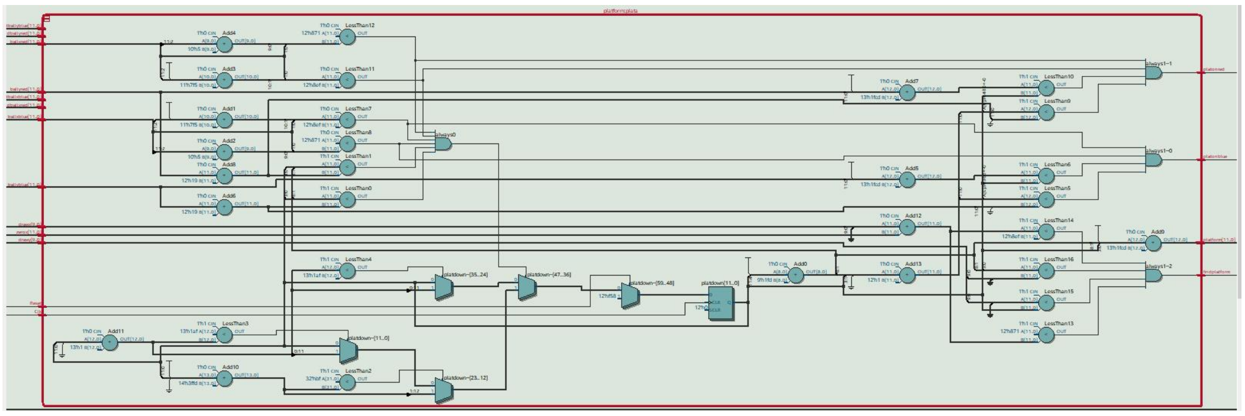
Inputs: Clk, Reset, [11:0] ballxblue, ballyblue, ballxred, ballyred, zerox, oldballxblue, oldballyblue, oldballxred, oldballyred, [9:0]drawx, drawy,

Outputs: findplatform, platonblue, platonred, [11:0] platform

Description: In this module, we control the movement of the platform going up when two characters standing on the platform. The platform can carry the two characters to the dais. We also use the signal findplatform to find the area of the platform and draw the position of the lift on the screen.

Purpose: This module achieve the goal to set the second obstacles of the game-- the elevator. It can carry the characters to the dais.

RTL Diagram:



colorselect.sv

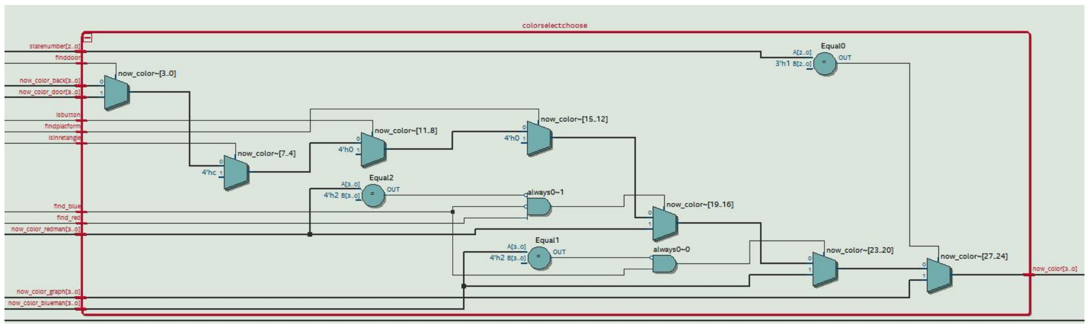
Inputs: logic find_blue, find_red, isinretangle, isbutton, findplatform, finddoor, [3:0] now_color_back, now_color_blueman, now_color_redman, now_color_door, now_color_graph, [2:0] statenumber,

Outputs: [3:0] now_color

Description: This module collect the area information from different modules and finally determine which module to print on the screen.

Purpose: This module determine which module to print on the screen.

RTL Diagram:



color_finder.sv

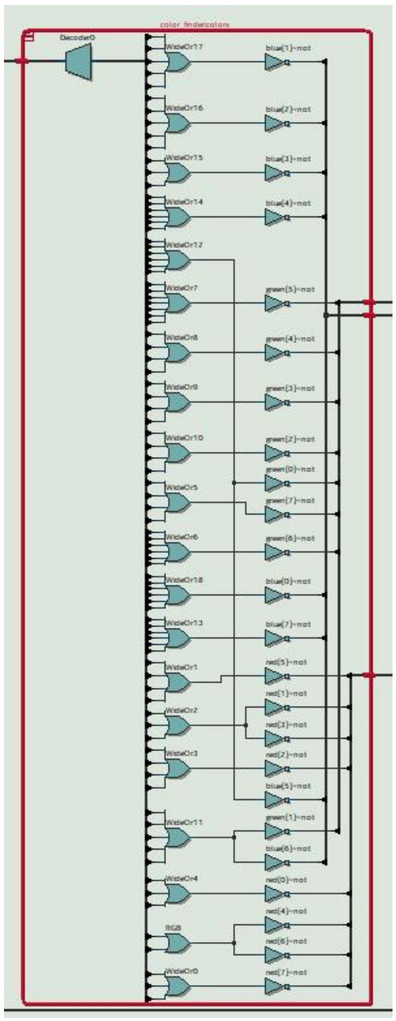
Inputs: [3:0] color

Outputs: [7:0]red, green, blue

Description: The information of the images we store in the memories are the index of the colors. We use the codes provided from the course website https://github.com/Atrifex/ECE385-HelperTools

Purpose: This module transfer the index of the color in the related RGB values.

RTL Diagram:



statemachine.sv

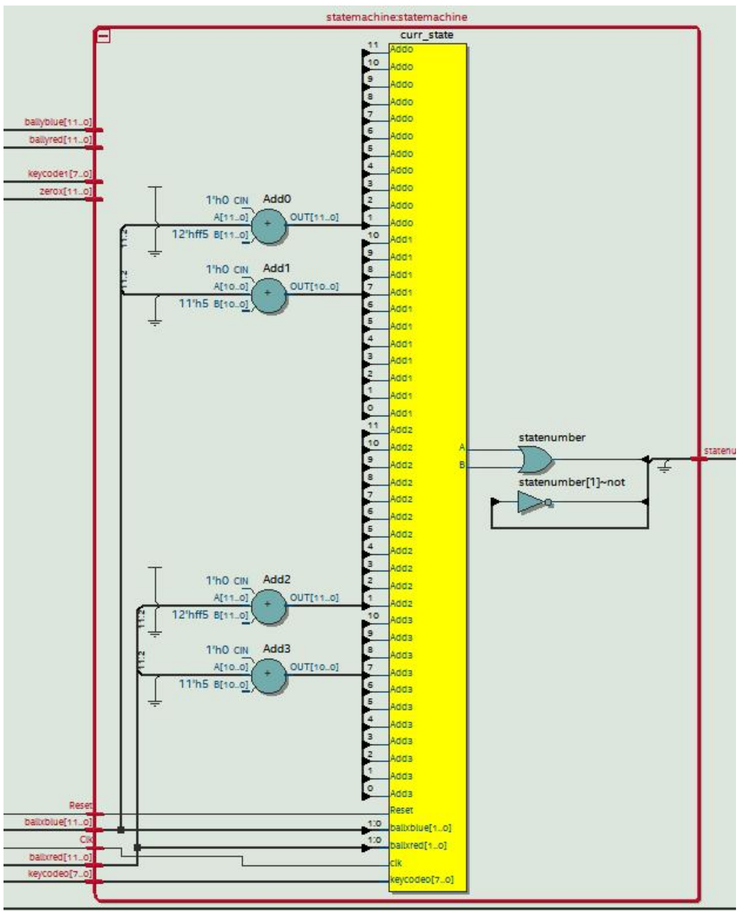
Inputs: Clk, Reset, [7:0] keycode0, keycode1, [11:0] ballxblue, ballyblue, ballxred, ballyred, zerox

Outputs: [2:0]statenumber

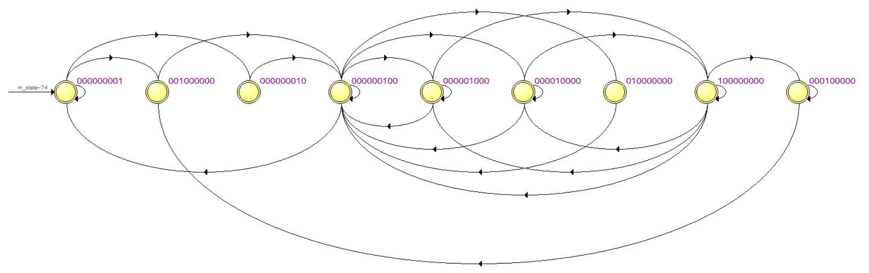
Description: We design a state machine to control the switch between interfaces.

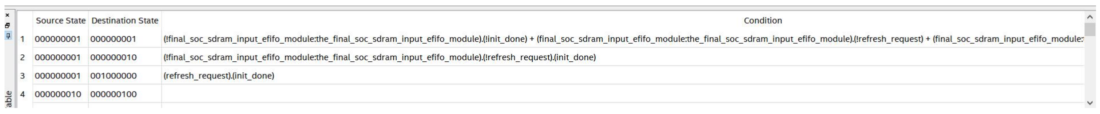
Purpose: We use this state machine to switch the starting page to the game and end the game.

RTL Diagram:



State Diagram:





	Source State	Destination State	Condition
1	000000001	000000001	(!final_soc_sdram_input_efifo_module:the_final_soc_sdram_input_efifo_module).(!init_done) + (final_soc_sdram_input_efifo_module:the_final_soc_sdram_input_efifo_module).(!refresh_request) + (final_soc_sdram_input_efifo_module:
2	000000001	000000010	(!final_soc_sdram_input_efifo_module:the_final_soc_sdram_input_efifo_module).(!refresh_request).(init_done)
3	000000001	001000000	(refresh_request).(init_done)
4	000000010	000000100	

edge_control.sv

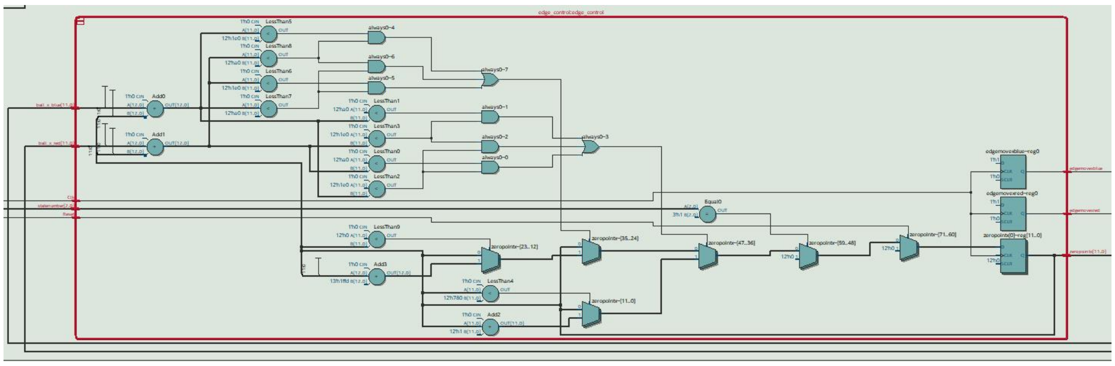
Inputs: [11:0]ball_x_blue, ball_x_red, Reset, Clk, [2:0] statenumber,

Outputs: [11:0] zeropointx, edgemovexblue, edgemovexred

Description: We use the positions of the characters and boundry conditions to calculate the proper position of the background. As the characters move, the background would follow them. The output value zeropointx refers to the x coordinate of the left up point of the background.

Purpose: This module calculate and control the scrolling of the background as the characters move.

RTL Diagram:



ball.sv

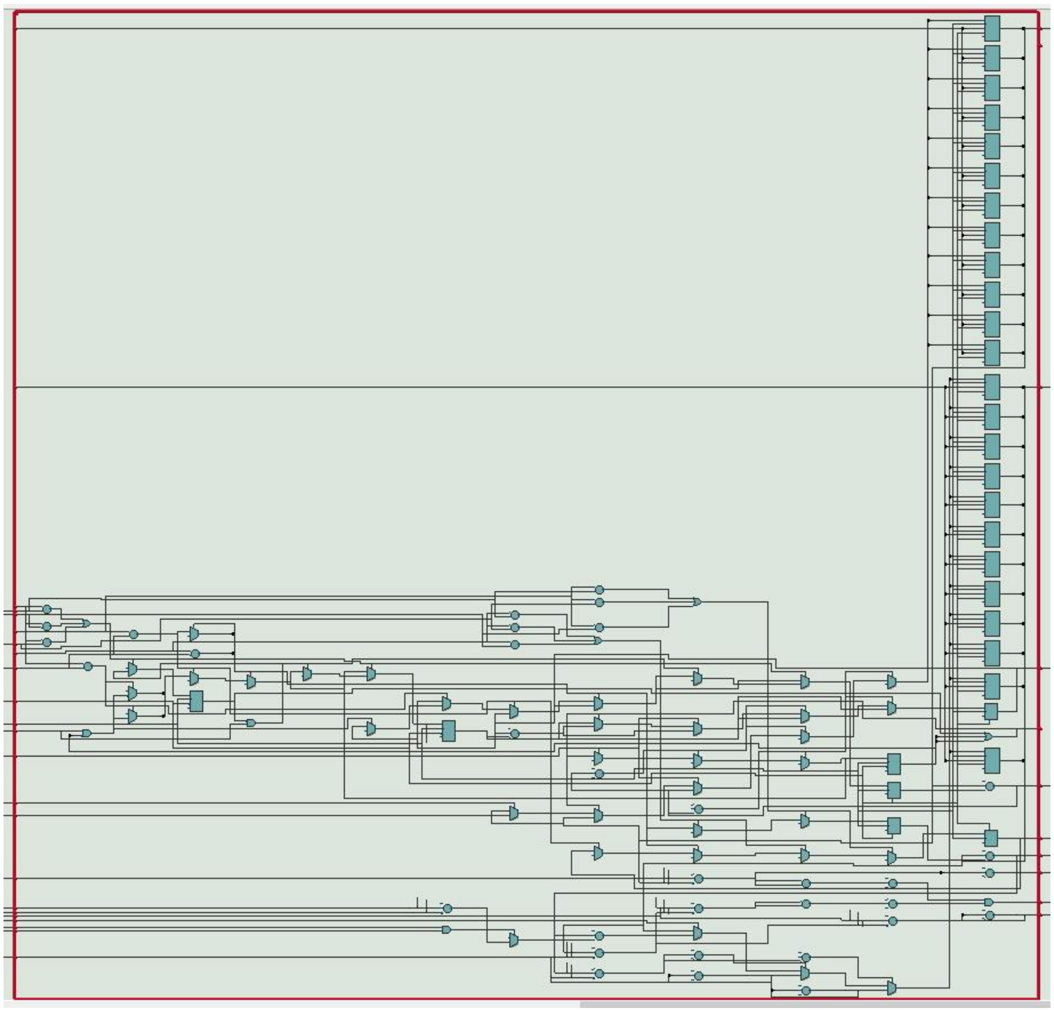
Inputs: Reset, frame_clk, [7:0] keycode0, keycode1, keycode2, keycode3, keycodeleft, keycoderight, keycodejump, [9:0] drawX, drawY, movex, movey, testjump, [11:0] Ball_X_Center, Ball_Y_Center, [11:0] zerox, [11:0] otherx, othertestx, istop, othermovex, [11:0] platform, platon

Outputs: [11:0] BallX, BallY, BallS, find, [11:0] testballx, testbally, Ball_v, [9:0] findballx, findbally, x_direction, jump

Description: This module can provide the current position and the nect position of the character. In other words, it can determine the movement of the character under different situations. It can also control the ball to jump and move on the background and across the barriers.

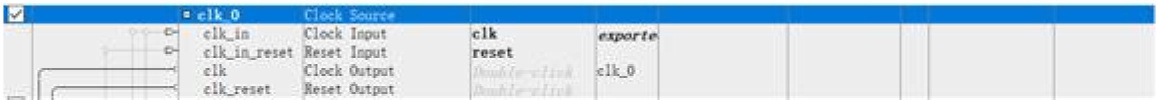
Purpose: This module is the control unit of the character. More detailed information is provided in section2.

RTL Diagram:



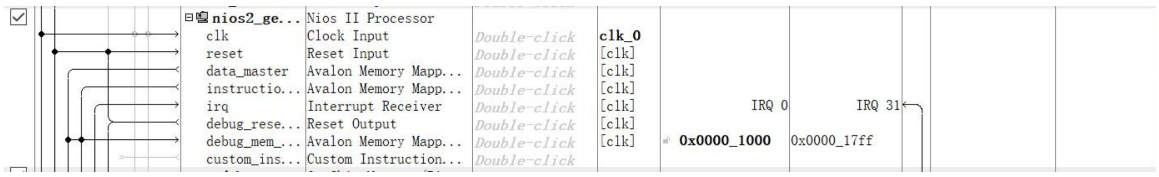
6. System Level Block Diagram

In the part of platform designer, we mainly refer to the hardware of lab6. There is little difference between the main part and lab6, but we add more keycode reading ports, so that we can control the complex actions of two figures at the same time.



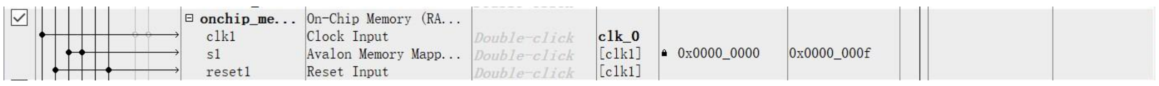
clk_0	Clock Source		
clk_in	Clock Input	clk	exporte
clk_in_reset	Reset Input	reset	
clk	Clock Output	Double-click	clk_0
clk_reset	Reset Output	Double-click	

This is the total clock of the whole system, used to control the operation of each part, and its output clk and reset are used by the components of other parts as clocks and reset



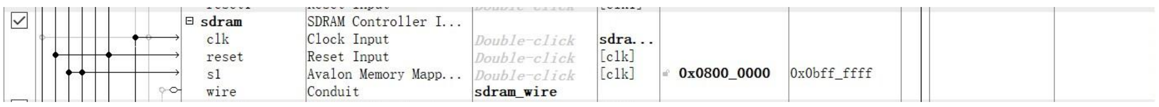
nios2_ge...	Nios II Processor				
clk	Clock Input	Double-click	clk_0		
reset	Reset Input	Double-click	[clk]		
data_master	Avalon Memory Mapp...	Double-click	[clk]		
instructio...	Avalon Memory Mapp...	Double-click	[clk]		
irq	Interrupt Receiver	Double-click	[clk]	IRQ 0	IRQ 31
debug_rese...	Reset Output	Double-click	[clk]		
debug_mem_...	Avalon Memory Mapp...	Double-click	[clk]	0x0000_1000	0x0000_17ff
custom_ins...	Custom Instruction...	Double-click			

This is the nios2 core CPU processor, and its output has two Avalon branches, data-Master and instruction master, which are used to transmit data and instructions, while all operations and processing are done in NIOS2



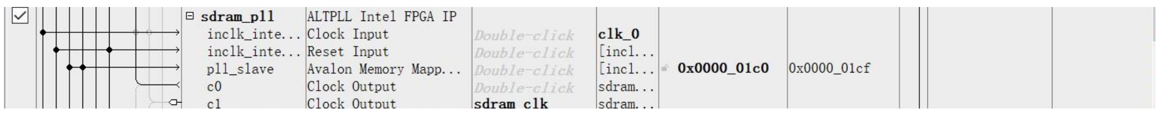
onchip_me...	On-Chip Memory (RA...				
clk1	Clock Input	Double-click	clk_0		
s1	Avalon Memory Mapp...	Double-click	[clk1]	0x0000_0000	0x0000_000f
reset1	Reset Input	Double-click	[clk1]		

This is part of the memory on the chip. It doesn't have much memory space, but it receives signals from the processor more efficiently.



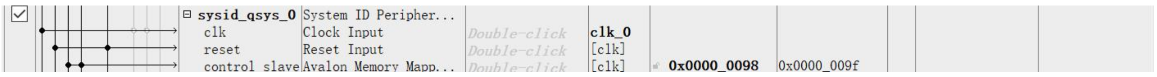
sdram	SDRAM Controller I...				
clk	Clock Input	Double-click	sdra...		
reset	Reset Input	Double-click	[clk]		
s1	Avalon Memory Mapp...	Double-click	[clk]	0x0800_0000	0x0bff_ffff
wire	Conduit	sdram_wire			

The function of this part of SDRAM is to communicate with external storage. Using the memory of our FPGA, its large capacity is convenient for us to store a large amount of data, such as software information.



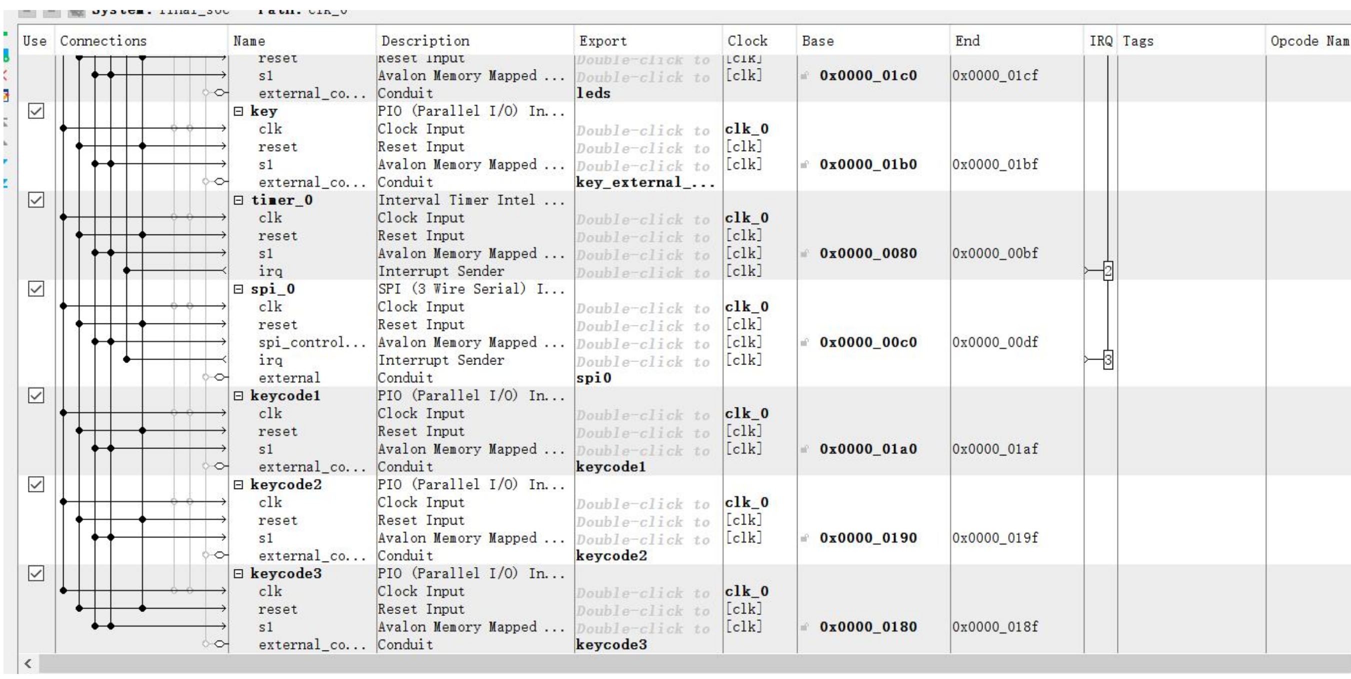
sdram_pll	ALTPLL Intel FPGA IP				
inclk_inte...	Clock Input	Double-click	clk_0		
inclk_inte...	Reset Input	Double-click	[incl...		
pll_slave	Avalon Memory Mapp...	Double-click	[incl...	0x0000_01c0	0x0000_01cf
c0	Clock Output	Double-click	sdram...		
c1	Clock Output	sdram_clk	sdram...		

The pll module plays a control role on the sdram module, and it will output a clock with delay, which can be used as the sdram clock, so as to effectively avoid the delay caused by reading data on the FPGA board.



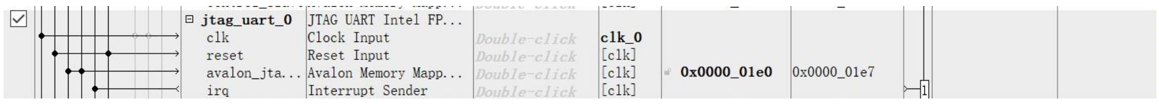
Use	Name	Description	Export	Clock	Base	End
✓	⊟ sysid_qsys_0	System ID Peripher...				
	clk	Clock Input	Double-click	clk_0		
	reset	Reset Input	Double-click	[clk]		
	control_slave	Avalon Memory Mapp...	Double-click	[clk]	0x0000_0098	0x0000_009f

This is a system id checker, which gives us a series of numbers, and then the software loader checks the software when starting the software. In this way, we can avoid finding that the software is not compatible with the processor or the software corresponding to the old hardware when we add the software.



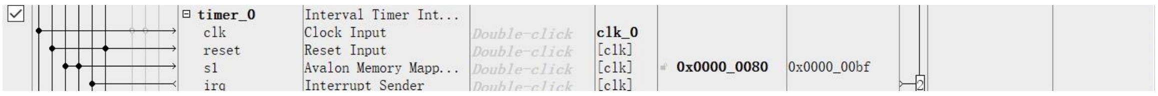
Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Nam
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped ...	Double-click to	[clk]	0x0000_01c0	0x0000_01cf			
		external_co...	Conduit	leds						
✓		⊟ key	PIO (Parallel I/O) In...							
		clk	Clock Input	Double-click to	clk_0					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped ...	Double-click to	[clk]	0x0000_01b0	0x0000_01bf			
		external_co...	Conduit	key_external_...						
✓		⊟ timer_0	Interval Timer Intel ...							
		clk	Clock Input	Double-click to	clk_0					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped ...	Double-click to	[clk]	0x0000_0080	0x0000_00bf			
		irq	Interrupt Sender	Double-click to	[clk]			2		
✓		⊟ spi_0	SPI (3 Wire Serial) I...							
		clk	Clock Input	Double-click to	clk_0					
		reset	Reset Input	Double-click to	[clk]					
		spi_control...	Avalon Memory Mapped ...	Double-click to	[clk]	0x0000_00c0	0x0000_00df			
		irq	Interrupt Sender	Double-click to	[clk]			3		
		external	Conduit	spi0						
✓		⊟ keycode1	PIO (Parallel I/O) In...							
		clk	Clock Input	Double-click to	clk_0					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped ...	Double-click to	[clk]	0x0000_01a0	0x0000_01af			
		external_co...	Conduit	keycode1						
✓		⊟ keycode2	PIO (Parallel I/O) In...							
		clk	Clock Input	Double-click to	clk_0					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped ...	Double-click to	[clk]	0x0000_0190	0x0000_019f			
		external_co...	Conduit	keycode2						
✓		⊟ keycode3	PIO (Parallel I/O) In...							
		clk	Clock Input	Double-click to	clk_0					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped ...	Double-click to	[clk]	0x0000_0180	0x0000_018f			
		external_co...	Conduit	keycode3						

PIO part: This is used to connect all the input and output ports of the experiment. It includes switch, button, led, usb and hex display. The content of this part is slightly different in lab6.1 and lab6.2, but it is more comprehensive in 6.2, so only the structure of part 6.2 is shown here.



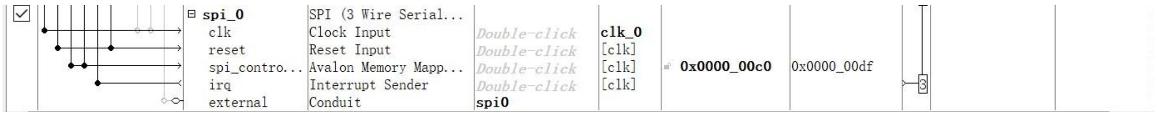
Use	Name	Description	Export	Clock	Base	End	IRQ
✓	⊟ jtag_uart_0	JTAG UART Intel FP...					
	clk	Clock Input	Double-click	clk_0			
	reset	Reset Input	Double-click	[clk]			
	avalon_jta...	Avalon Memory Mapp...	Double-click	[clk]	0x0000_01e0	0x0000_01e7	
	irq	Interrupt Sender	Double-click	[clk]			1

This module is convenient for us to carry out printf operation during debug.



```
timer_0     Interval Timer Int...
  clk       Clock Input            Double-click  clk_0
  reset     Reset Input            Double-click  [clk]
  s1        Avalon Memory Mapp...  Double-click  [clk]   0x0000_0080   0x0000_00bf
  irq       Interrupt Sender       Double-click  [clk]
```

Timer is a tool that we use to process delays and count the underlying delays. It can count up the cycles when it is required.

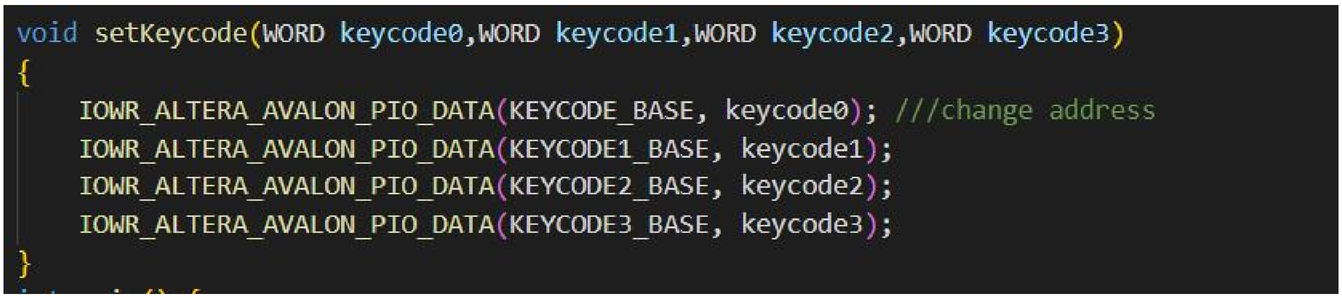


```
spi_0           SPI (3 Wire Serial...
  clk           Clock Input            Double-click  clk_0
  reset         Reset Input            Double-click  [clk]
  spi_contro... Avalon Memory Mapp...  Double-click  [clk]   0x0000_00c0   0x0000_00df
  irq           Interrupt Sender       Double-click  [clk]
  external      Conduit                spi0
```

Spi module is a new component we added in 6.2. Through this part, we can write the data read by USB into our central processing unit (CPU) to realize the interaction between USB and CPU.

7. Software Component of the lab

In the software part, we use lab6 to provide usb_kb software code, through which to read keycode information of keyboard. Different from lab6, we modify setkeycode function, so that multiple keycodes can be read at the same time. This allows you to handle the simultaneous pressing of multiple keyboard keys.



```
void setKeycode(WORD keycode0,WORD keycode1,WORD keycode2,WORD keycode3)
{
    IOWR_ALTERA_AVALON_PIO_DATA(KEYCODE_BASE, keycode0); ///change address
    IOWR_ALTERA_AVALON_PIO_DATA(KEYCODE1_BASE, keycode1);
    IOWR_ALTERA_AVALON_PIO_DATA(KEYCODE2_BASE, keycode2);
    IOWR_ALTERA_AVALON_PIO_DATA(KEYCODE3_BASE, keycode3);
}
```

8. Design Statistics Table

LUT	9,366 / 49,760 (19 %)
DSP	20
Memory(BRAM)	460,800 / 1,677,312 (27 %)
Flip-Flop	2,743 / 51,509 (5 %)
Frequency	132.21 MHz
Static Power	96.18mW
Dynamic Power	0.68mW
Total Power	106.17mW

9. Conclusion

9.1 Problems encountered in the design and solutions

When we finished the final project, we also encountered some problems. The first problem was the reading of colors. At the beginning, the screen could not show the colors we needed well, but showed a very dark color. After the research, we found that we need to add the sync signal control. In addition, we had a situation where the character would jump too high all at once, and we stabilized the character's jump by limiting the triggering of the jump.

9.2 Conclusion

In this final project, we built a two-person cooperative game through the hardware design with NIOS as the cpu core and the software part interacting with the keyboard, and through vga signal output. Here we can use the keyboard to control the two characters through various obstacles and platforms and finally reach the end door. In the experimental design, we have a further understanding of the systemverilog language, and also learn the internal operating principle of FPGA board, as well as vga signal imaging principle, keyboard signal reading principle. In the process of correcting the code, we are constantly improving our ability. In the process of correcting errors, we also found the wrong understanding in the lab in the past. This final project has provided great help to our personal ability improvement.