ECE 385

Spring 2023

Final Lab: Poke-maze

Romin Patel & Diqing Zuo

Rominmp2, Diqingz2

Lab Section: AL1

05/09/2023

TA: Marwan Eladl

Introduction:

In this lab, we were assigned to design and construct our own project with the utilization of the FPGA. The lab consisted of four-weeks in which it consisted of brainstorming project ideas, critical thinking for the construction of the hardware and software components that must be incorporated, and additional features to add onto the project in order to make the final result of the project more appealing. For our final project, we decided on creating a Pokemon version of a maze game. The maze is composed of the creation of sprites that will be shown on the VGA screen using the help of coloring pixels onto the screen. There are a few purposes of the creation of sprites for our game as it allows a sprite to be controlled by the user's input and allows enemies to create difficulty to pass through the mazes. The objective of the game is to move a sprite character with the help of the WASD keys (from a wired keyboard interface) through different mazes, however, the obstacles of the game such as enemies, narrower gaps throughout the maze, and the collision of the walls of the maze create difficulty in the mazes. There are a total of 3 mazes with increased difficulty every incremented level to complete in order to successfully finish the game. Overall, the game was designed with the help of prior knowledge obtained from the class and the completion of the previous labs to complete this project.

Written Description of the Hardware and Software Components of the Game:

In this portion we will explain the hardware and software components that were a part of the process in order to design and function the maze game. To begin, the NIOS II processor is utilized to connect and interact with both the MAX3421E chip and the VGA components. The MAX3421E chip was basically already set up for us as we just had to transfer the lab 6 software component and the PIO blocks in order to create the necessary SPI communication in order for the MAX3421E chip to communicate with a wired keyboard. The setup of the keyboard allowed the sprite that is controlled by the user to be able to have movement. The software component is concluded there as that is the only necessary software construction we had to make for the project. From lab 6, we used the keycode inputs to move a coded ball, however, we had to move a sprite that is designed with colored pixels. Therefore, the file that controls the movement of the sprite consists of a case statement that depicts the direction of the sprite depending on the keycode pressed by the user. This information is then used by the VGA file in order to correctly display the sprite onto the VGA screen through the VGA host port that is a part of the FPGA. Overall, the NIOS II processor with the help of the Platform Designer tool allows the necessary data input to be obtained to the SystemVerilog code in order to correctly output the necessary displays to the VGA screen.

Now, we will focus on the hardware aspect of the lab, which is the more crucial component that needs addressing. As previously mentioned, we have explained how the software component comes into play in the description above. In addition, several more modules were created beyond the recycled lab 6 modules, to incorporate additional features and sprites. The initial consideration is that for all modules requiring a clock signal, a 50 MHz clock was utilized. This is because that seemed to be sufficient for the game we developed since we mainly have to move sprites along the screen. Therefore, the frequency used in lab 6 remained the same for us. If we later wanted to increase the sprite's speed, then we could have increased the frequency for the clock to do so, however, we felt the 50 MHz clock is appropriate for our game.

The VGA controller module generates timing signals known as horizontal and vertical sync, which are essential for our VGA monitor to function. The code defines a DrawX and DrawY coordinate, used by the electron beam, that allows the coloring tools in our design to color each pixel required without modifications for future designs. The VGA host port on the FPGA board connects the VGA to the board. The combination of these two components and the VGA_controller module's code function demonstrates their relationship. The Color Mapper module colors each pixel in the sprite design and maps it to the VGA_controller module. Similar to the ball.sv file from lab 6, the pokeball.sv module handles sprite movement with respect to the vertical sync in our design. Recognizing the start of the next frame when the signal shifts from low to high, we update the sprite's movement based on the current key pressed during the timeframe.

In the lab, we had to develop several new hardware modules in order to add features to the game we designed. To begin, we constructed a module that keeps track of the amount of times the user collides with a wall. Furthermore, whenever the count equals to ten then the game must come to an end. The game will know when the user collides with a wall by another module that detects when the sprite has interacted with a wall of the maze. The wall bouncer module does the necessary math in order to obtain the corners of the sprite and checks the condition of whether one of the corners interacts with the bitmap design of the maze. The bitmap for each of the mazes is designed in another module, in which there are different bitmaps for each of the different levels in the game. The bitmap module constructs the format of the mazes by using row order form, in which each row is filled with 0's or 1's depending on the path that was constructed. The different mazes that were made were chosen, based on the counter module. The counter module is a counter that outputs a 3 bit value in order to indicate what maze needs to be displayed by being an input value to the bitmap module. Along with the collision of the walls, there is a module that is required to detect collision with enemies. To begin, for our enemy we used a pokeball as a sprite and we used another ball.sv type of module in order to have fixed horizontal movement for the pokeball along the maze. In the module, we assigned the horizontal movement to a specific

horizontal coordinate in which it would move left to right independently from any user input. Moving to the collision detector module, it allowed the correct output signal of when the pokeball would interact with the user's sprite (the user's sprite is Charmander, which is a pokemon). The collision detector is set to high in the module, depending on the distance between the two sprites. Furthermore, if the distance of the sprites were within a close distance, then the output signal of the collision would be set to high.

Besides the new modules that were constructed, there were modifications to the color mapper module we needed to make in order to display the sprites to the screen. We first had to choose the sprites we wanted to use for our game and be able to store the image to the M9K on-chip memory. We were capable of doing this by using Ian's Helper Tool in order to automatically generate the necessary files in order to store the images to the RAM. We used the new files generated from the Ian's Helper Tool in order to display the correct sprites that we wanted to display in the color mapper module.

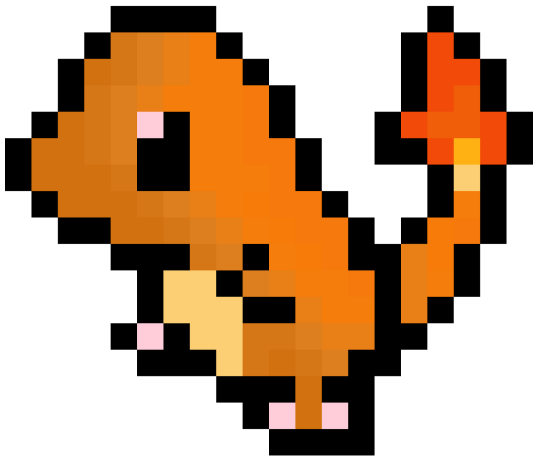

Image 1: Sprite used for User Movement

Image 2: Sprite Used for Enemy Collision

Lastly, the main module that controls the signals and states of the game is the state machine module. The state machine allows when the game should be ended whether that be with the collisions of walls or enemies, incrementing to the next maze, when to turn on different sprites, when the collision with the wall counter value should be incremented, and when to show the start and end screen of the game. The state machine starts with the start state, in which the start screen signal is set to high and ultimately allows the start screen image to be displayed onto the VGA screen. The next state is the level state, in which it is moved to this state if the space bar is pressed in the start state. In the level state, all of the moving sprites are turned on. In the level state, it checks many different conditions such as whether the collision with the wall has occurred 10 times, if all three mazes have been passed, if the collision with the pokeball occurred, and if each individual maze has been passed. Besides the collision with the wall and the pass of an individual maze, the next state for the other conditions is to move to the end state. For the condition of when the collision with the wall occurs, it moves to the score state in which it has a total of 4 wait states in order to update the score on the screen. For the condition of when the individual maze has been completed it moves to the increment state, in which it increments the counter and ultimately shows the next maze to complete. Lastly, the end state sets the signal of the end screen in which it shows the “game over” screen to the VGA display.

State Diagram

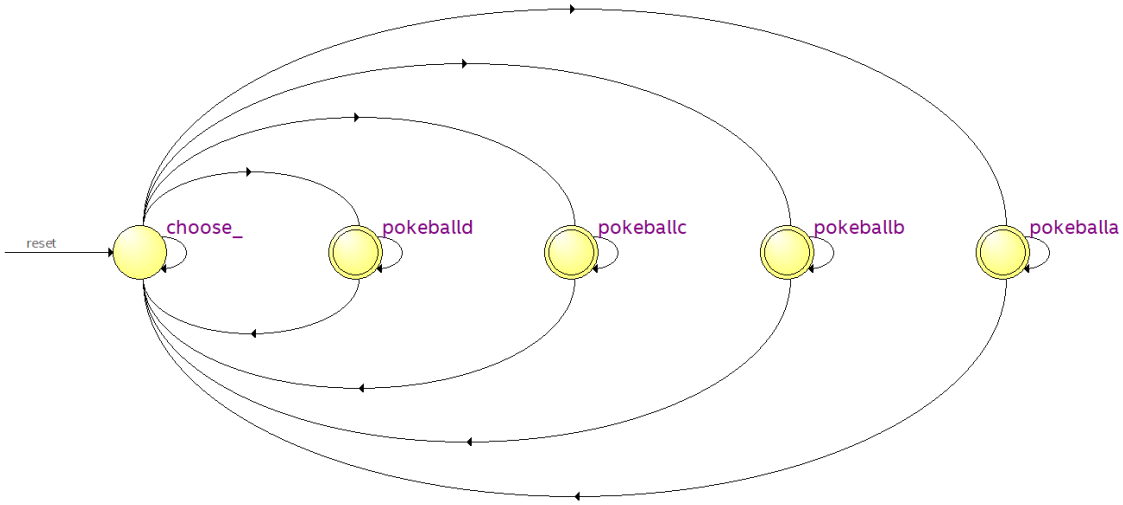

Image 2: State diagram for Pokemon Chooser

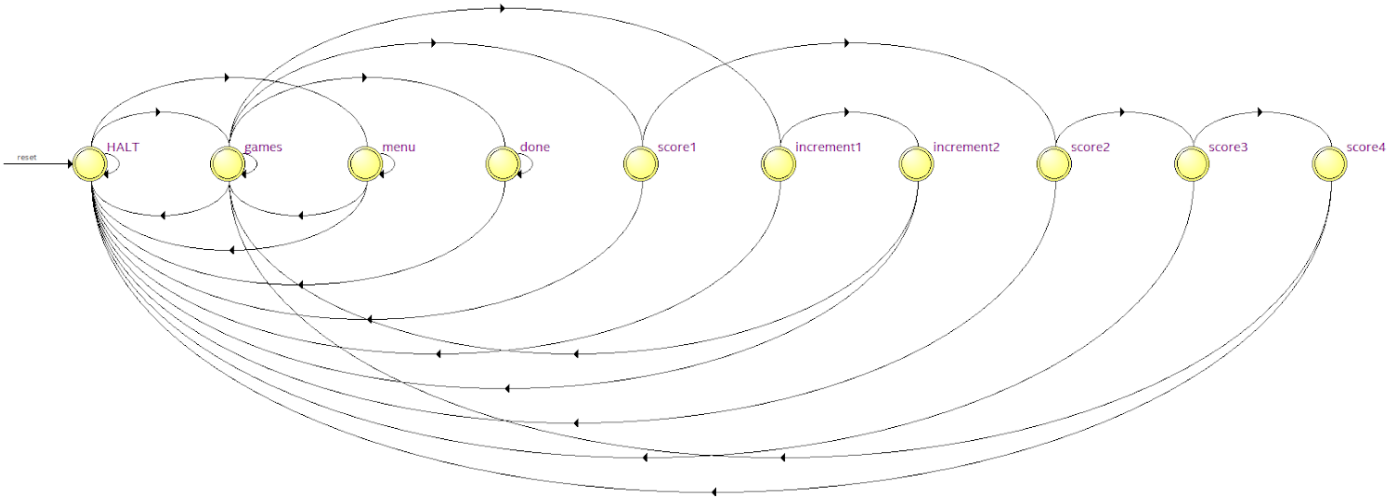

Image 3: Game Level Chooser

Overall Level Block Diagram:

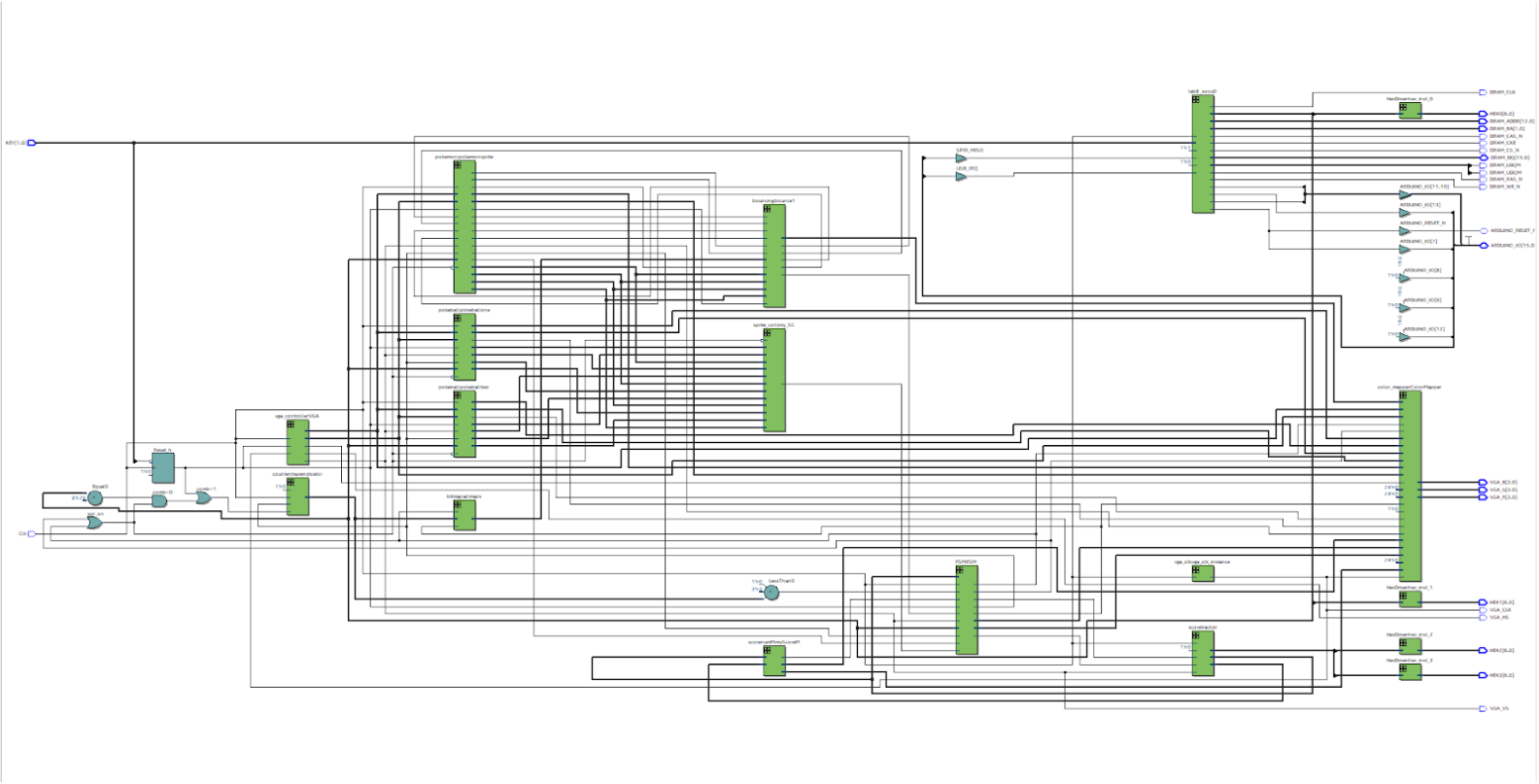

Image 4: Overall Level Block Diagram

System Level Block Diagram:

Pokemon sprite:

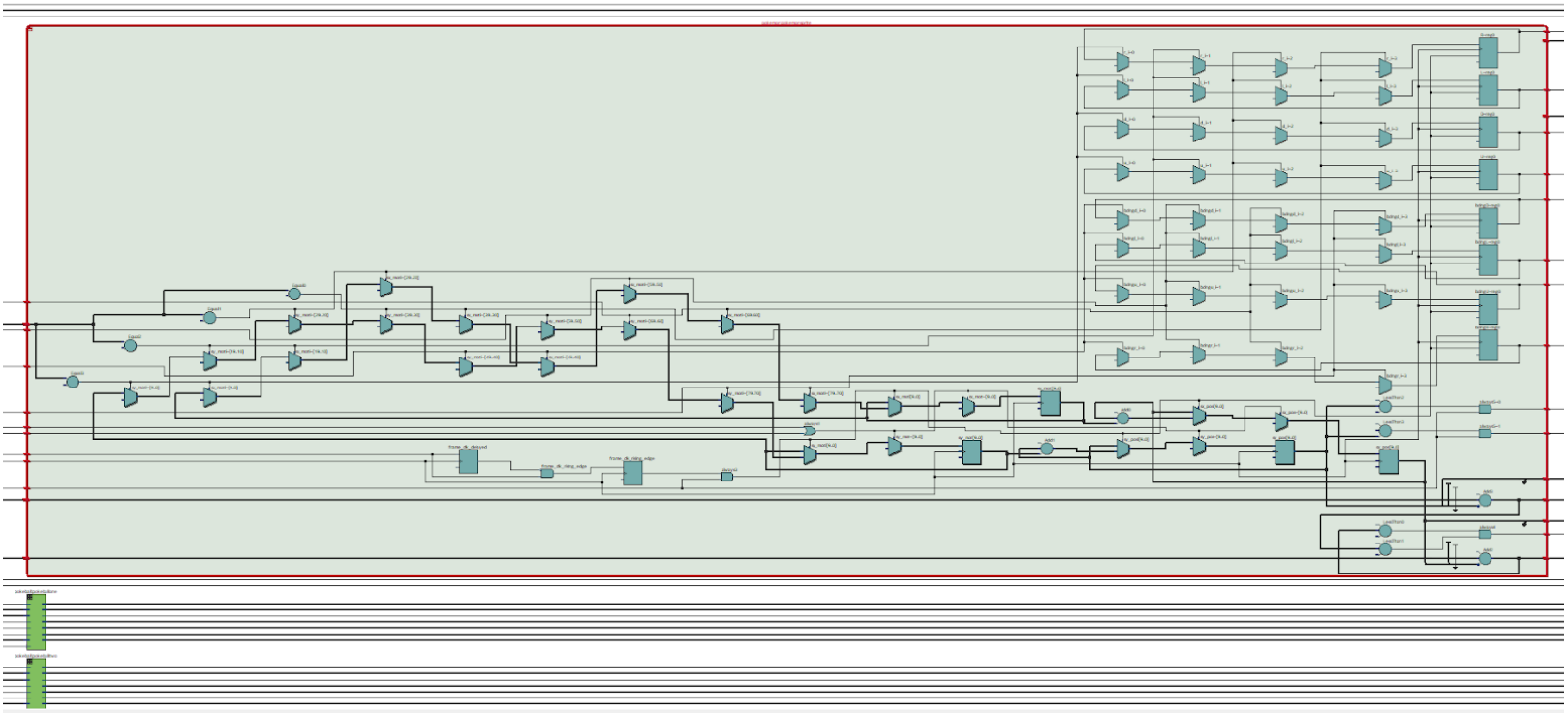

Image 5: Pokemon Sprite Block Diagram

Counter:

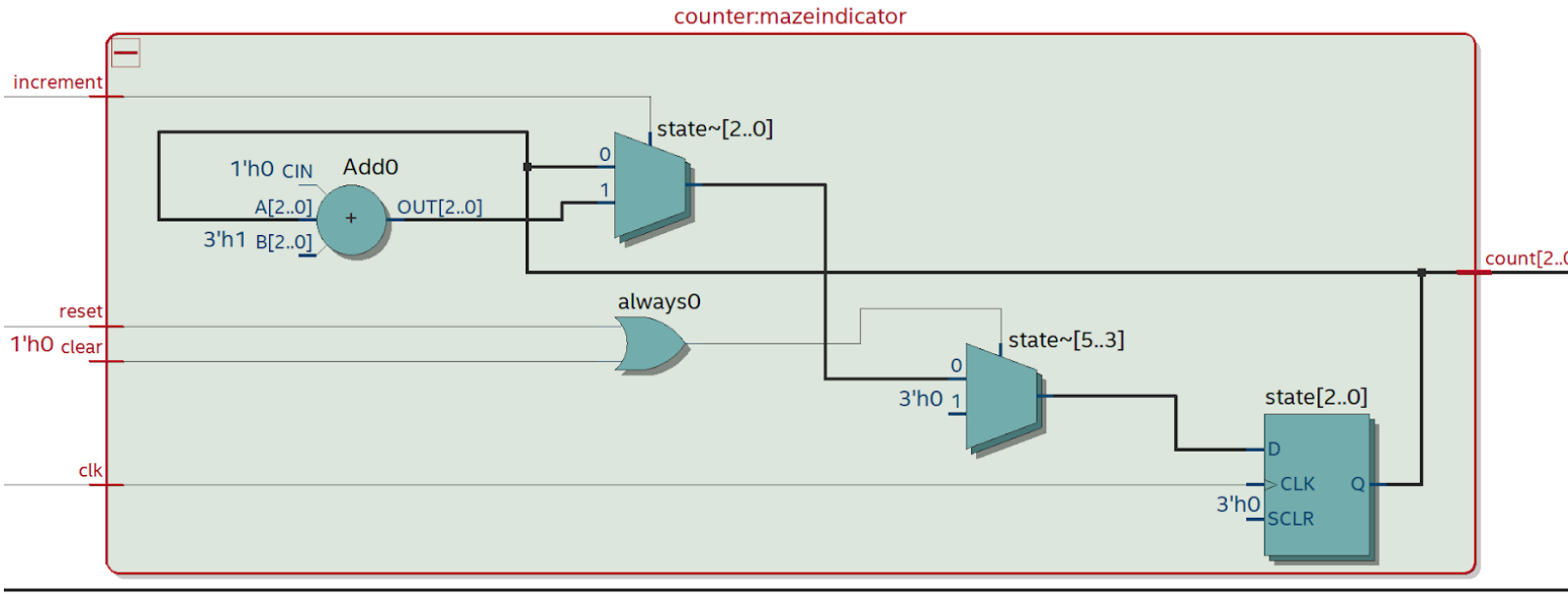

Image 6: Counter Block Diagram

Bitmap:

Image 7: Bitmap Block Diagram

Bouncing:

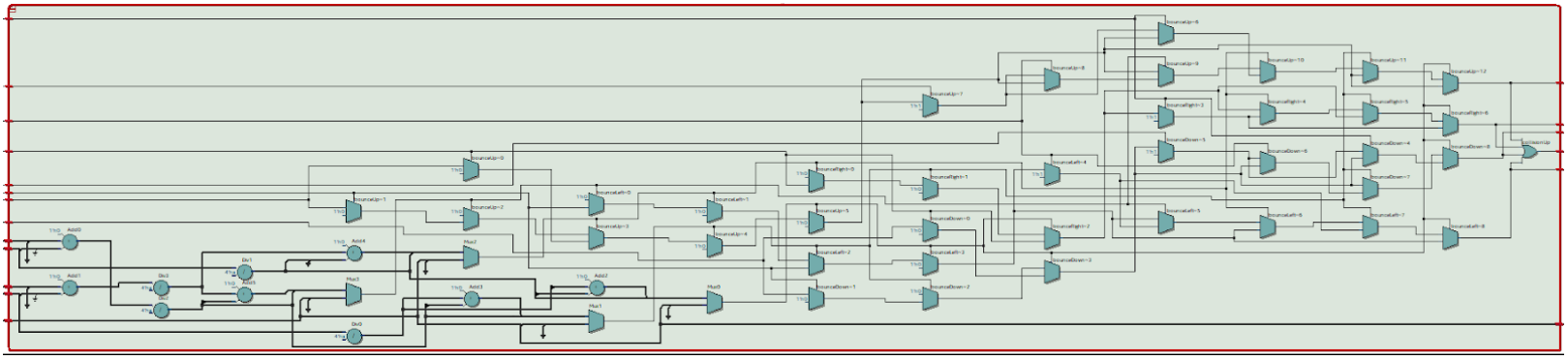

Image 8: Bouncing Block Diagram

Sprite_collision:

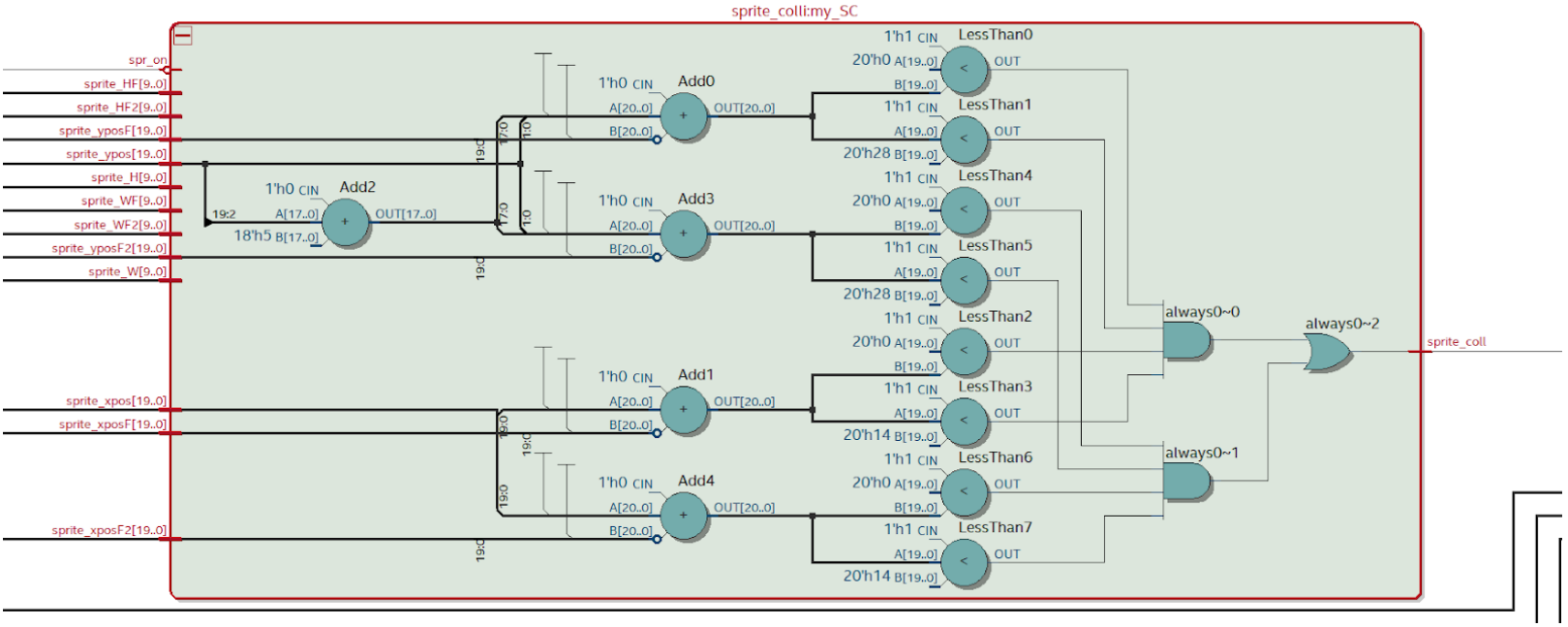

Image 9: Sprite_collision

FSM:

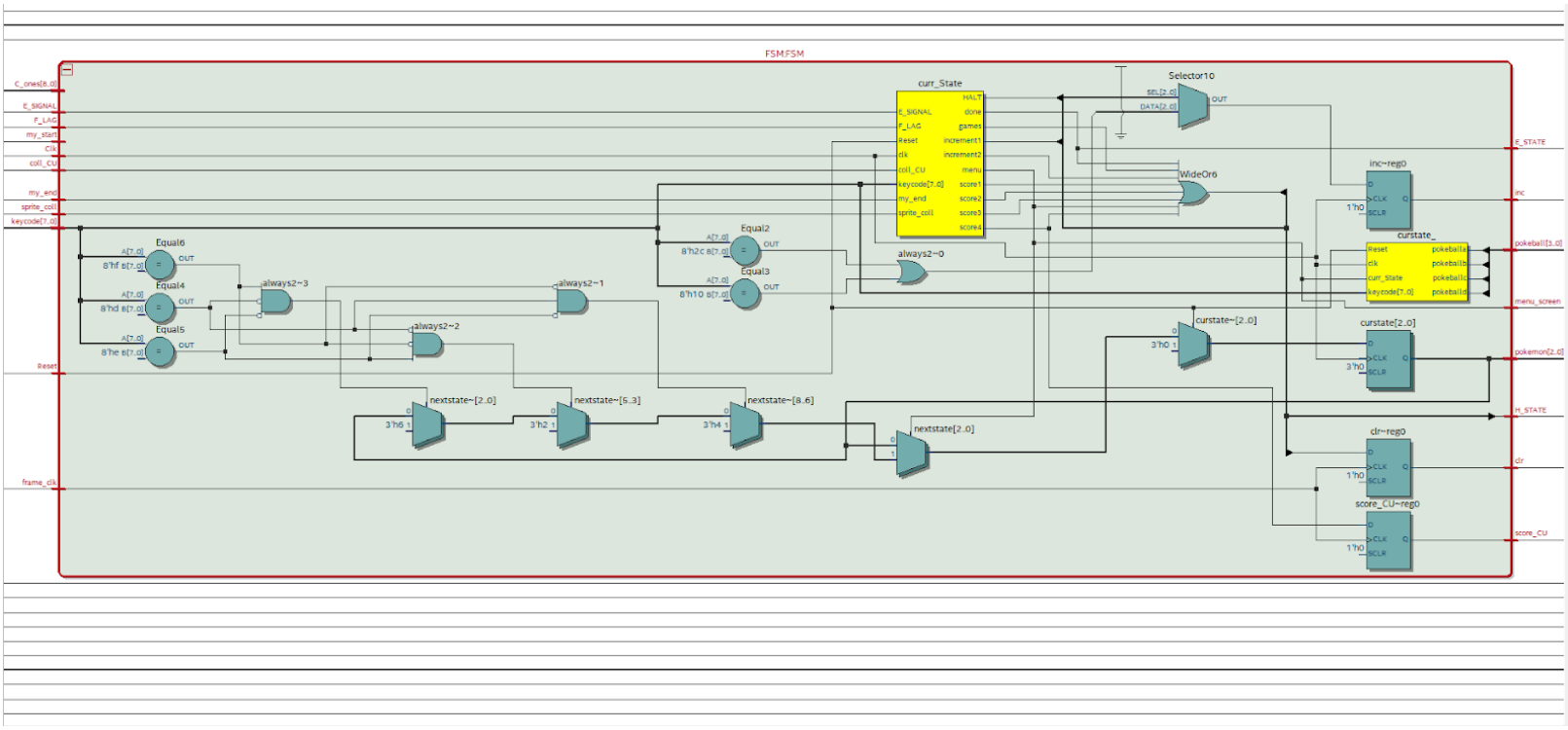

Image 10: FMS Block Diagram

Top:

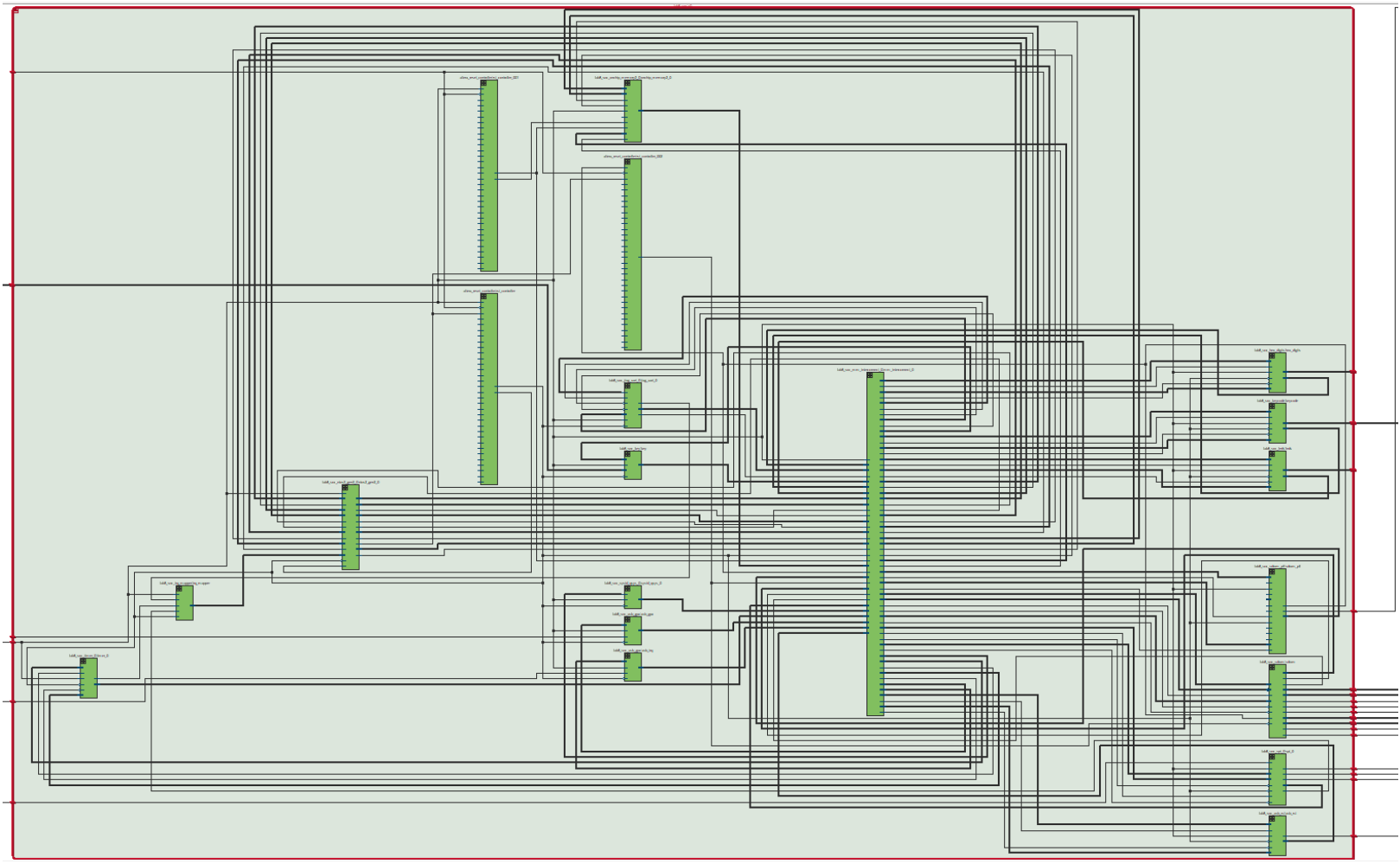

Image 11: Top Level Block Diagram

Scoretracker:

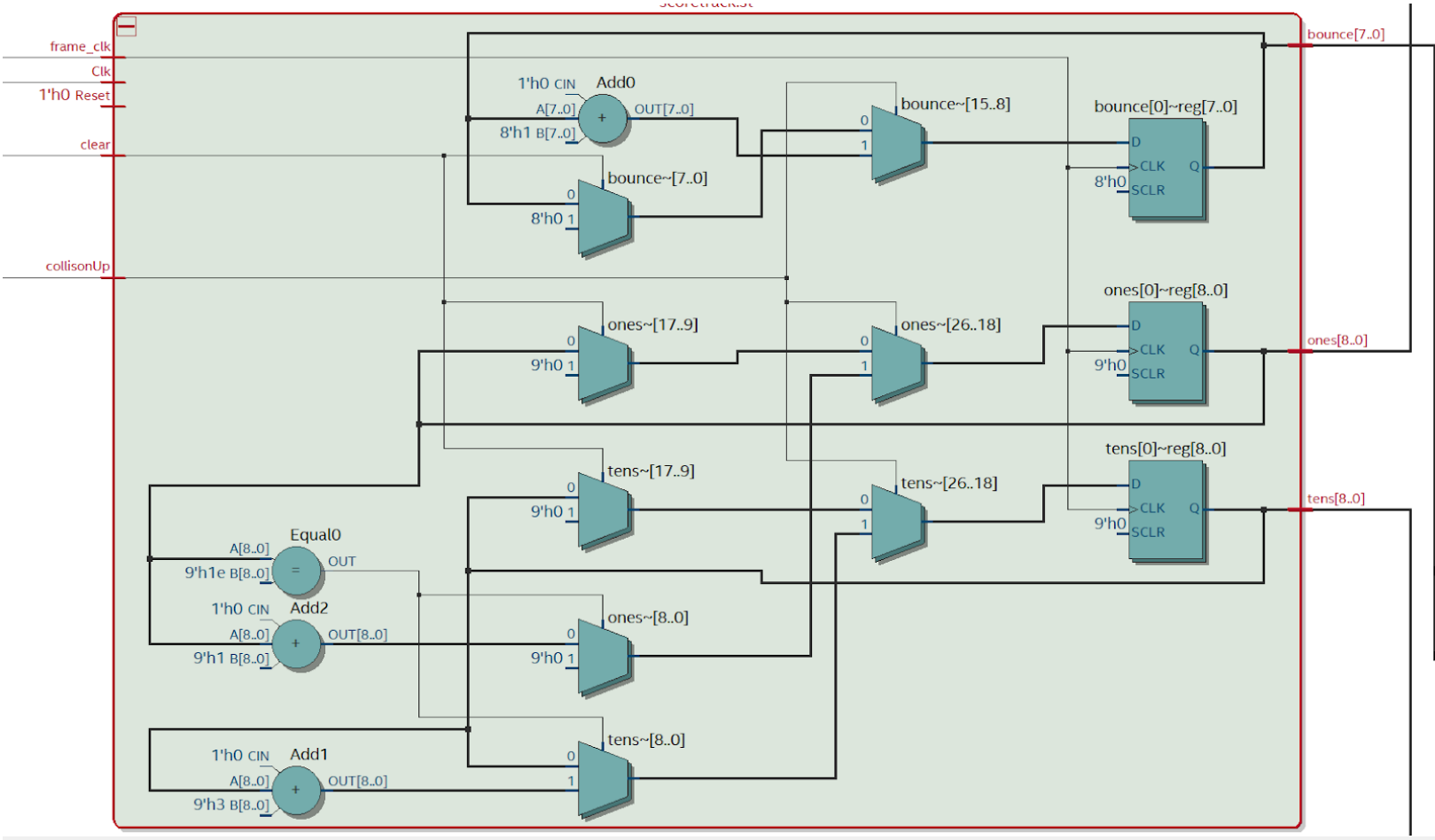

Image 12: Score Tracker Block Diagram

ColorMapper:

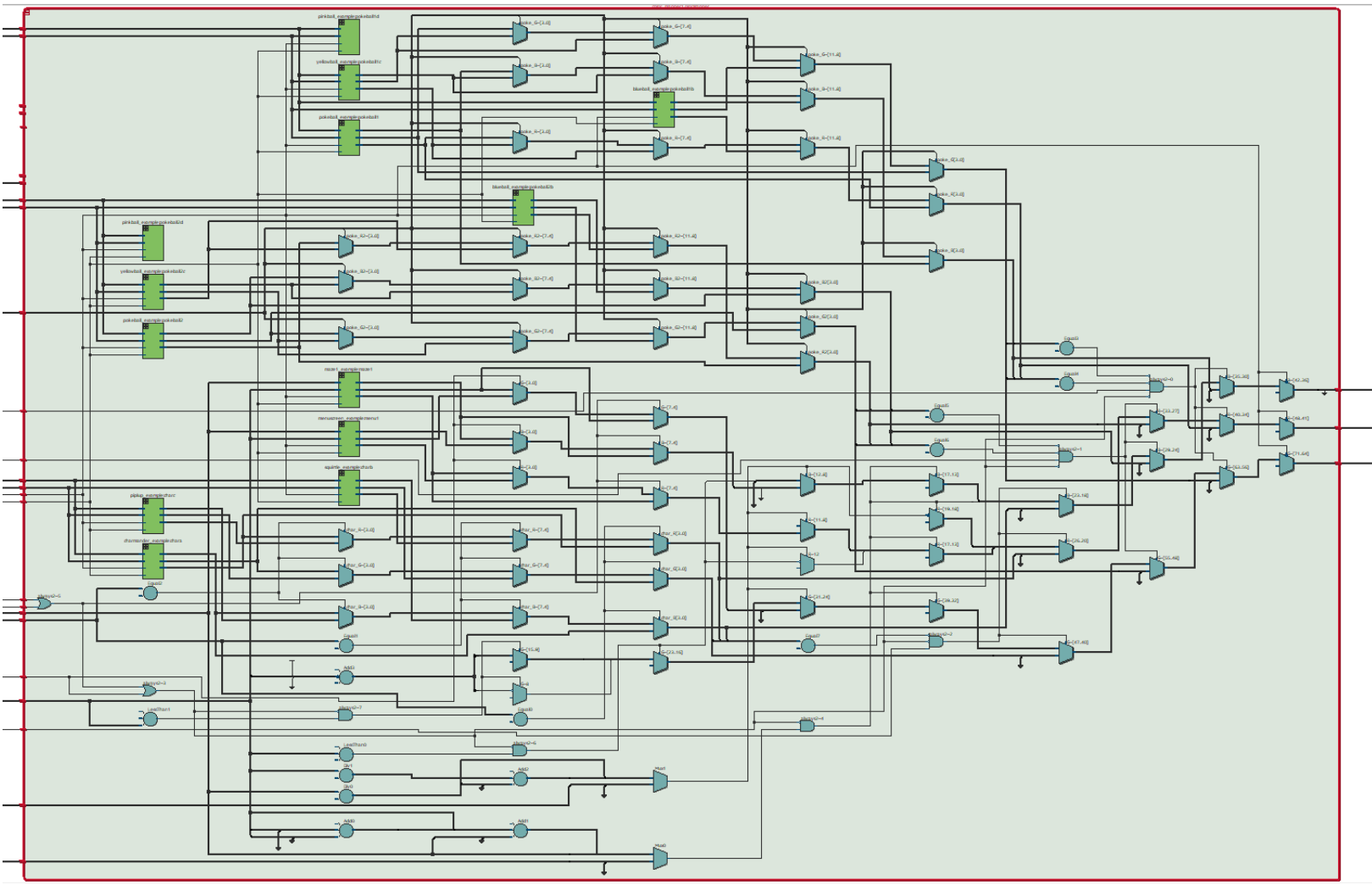

Image 13: ColorMapper

Module Description:

Module: Pokeball.sv
Inputs: Clk, Reset, frame_clk, [9:0] DrawX, [9:0]DrawY, [7:0] keycode, bnceL, bnceR, bnceU, bnceD, spr_on, inc.

Outputs: [19:0] Sprite_dx, [19:0] Sprite_dy, [9:0] sprite_W, [9:0] sprite_H, [19:0] sprite_xpos, [19:0] sprite_ypos, L, R, U, D, bcingL, bcingR, bcingU, bcingD, my_Sprite, my_start, my_end.

Description: The module assigned the movement of the pokeball, which is an enemy projectile in the game. The module assigns a fixed X and Y coordinates to the pokeball, and also sets up the speed horizontally for the pokeball to move from left to right.

Purpose: The purpose of the module is to construct an enemy for the maze to become difficult for the person to traverse through the game. The module will also be required for the collision module that detects if the charmander interacts with the pokeball.

Module: bitmap
Inputs: [2:0] count, E_STATE, menu_screen

Outputs: [3071:0] C_map

Description: The module decides the map, start screen, or the end screen that needs to be displayed based on the counter input signal. There are a total of 5 different bit-maps in the module. 1 for the start screen, 1 for the end screen, and 3 different bit-maps for the mazes.

Purpose: In order to display the correct game screen to the VGA screen and also make all the necessary screens, this module was required. The input of the signal is obtained from the counter module and is incremented based on the state machine constructed.

Module: Score display
Inputs: [8:0] C_ones, [8:0] C_tens

Outputs: [0:4799] ones, [0:4799] tens, F_LAG

Description: This module takes two input digits, C_ones and C_tens, and converts them into bitmaps for displaying on a VGA screen as a score. Each pattern is 80 bits wide and 60 bits high. Two patterns are provided for each digit, one for an on segment, represented by “1”s, and one for an off segment, represented by “0”s.

Purpose:The purpose of this module is to create a score display for a VGA screen based on two score digits to be displayed. The module provides a bitmap representation of each digit, which is then combined to create the final score display.

Module: sprite_colli

Inputs:
[9:0] sprite_WF: width of the first sprite
[9:0] sprite_HF: height of the first sprite
[19:0] sprite_xposF: x-position of the top-left corner of the first sprite

[19:0] sprite_yposF: y-position of the top-left corner of the first sprite
[9:0] sprite_WF2: width of the second sprite
[9:0] sprite_HF2: height of the second sprite
[19:0] sprite_xposF2: x-position of the top-left corner of the second sprite
[19:0] sprite_yposF2: y-position of the top-left corner of the second sprite
[19:0] sprite_xpos: x-position of the sprite being checked for collision
[19:0] sprite_ypos: y-position of the sprite being checked for collision
[9:0] sprite_W: width of the sprite being checked for collision
[9:0] sprite_H: height of the sprite being checked for collision
spr_on: signal to enable the sprite collision detection
Outputs:
sprite_coll: signal indicating whether a collision has occurred between the sprite being checked and any of the other two sprites (pokemon and pokeball)

Description:
The module sprite_colli is responsible for detecting collisions between two sprites. It takes as input the positions and dimensions of two sprites and also the position and dimensions of a third sprite that is being checked for collision with the other two. If the spr_on signal is high, indicating that sprite collision detection is enabled, the module checks whether the third sprite overlaps with either of the other two sprites. If a collision is detected, the sprite_coll signal is set high, otherwise it is set low.

The module first calculates the vertical and horizontal distances between the third sprite and each of the other two sprites using the following equations:

D_Y = (sprite_ypos + 10'd20) - sprite_yposF;
D_X = sprite_xpos - sprite_xposF;
D_Y2 = (sprite_ypos + 10'd20) - sprite_yposF2;
D_X2 = sprite_xpos - sprite_xposF2;
Then, it checks whether there is any overlap between the third sprite and the other two sprites by checking whether the vertical and horizontal distances are within the dimensions of the other sprites:

If (((D_Y >= 1'b0 && D_Y <= 10'd40) && (D_X >= 1'b0 && D_X <= 10'd20)) || ((D_Y2 >= 1'b0 && D_Y2 <= 10'd40) && (D_X2 >= 1'b0 && D_X2 <= 10'd20))), the sprite_coll signal is set high indicating a collision.
Otherwise, the sprite_coll signal is set low indicating no collision.

Purpose:

The module sprite_colli allows for the detection of collisions between sprites, which would be part of one of our main baseline functions, as it would help us decide if the pokemons are caught by any of the pokeballs

Module: counter

Inputs:
Clk, reset, clear, increment

Outputs:
Count

Description:
The counter module is responsible for incrementing and updating the counter value, which is used to determine which maze should be displayed on the screen. The counter value is incremented by one when the 'increment' signal is asserted, which is obtained from the FSM. The counter uses a synchronous always_ff block to update the counter state based on the clock signal.

Purpose:
The purpose of this module is to provide a counter to keep track of the maze that is currently being displayed on the screen. This is necessary for the VGA controller to know which bitmap to display. The counter is updated based on the 'increment' signal, which is obtained from the FSM.

Module: FSM

Inputs:
Clk, frame_clk, keycode, E_SIGNAL, Reset, my_start, my_end, coll_CU, C_ones, sprite_coll, F_LAG

Outputs:
H_STATE, E_STATE, inc, menu_screen, pokemon, pokeball, score_CU, clr

Description:
The module uses three state variables curr_State, curstate, and curstate_. It also defines two more state variables: HALT and menu. The HALT state stops the game, and the menu state displays the game menu.

Description: The FSM module is a finite state machine that controls the flow of the whole game. The module receives inputs such as the clock signals (Clk, frame_clk), keyboard input (keycode), game states (E_SIGNAL, my_start, my_end, coll_CU, sprite_coll, F_LAG), and other variables (C_ones) and generates outputs (H_STATE, E_STATE, inc, menu_screen, pokemon, pokeball, score_CU, clr) to control the game. With pokemon and pokeball as the states that would eventually tell us what the player chooses for their pokemon and pokeball characters, as shown on our menu screen.

Purpose: The purpose of the FSM module is to control the game flow, which includes selecting the current game state and updating the corresponding outputs to display on the screen. The module has nine possible states, including HALT, menu, games, increment1, increment2, score1, score2, score3, and score4. The module transitions between these states based on the inputs received. The module also controls the display of various game elements, such as the menu screen, pokemon and pokeball, and the score display. The module generates signals to update these elements based on the current game state and other inputs.

Module: ScoreCounter

Inputs:
Clk, frame_clk, clear, Reset, collisonUp

Outputs:
[7:0] bounce: score count for each collision
[8:0] tens: tens digit of the overall score
[8:0] ones: ones digit of the overall score

Description:
The ScoreCounter module is responsible for keeping track of the score in a game. It increments the score by 1 for each collision that occurs, as indicated by the collisonUp input signal. The overall score is displayed on the top right corner of the VGA screen, where the tens digit is output to the tens signal and the ones digit is output to the ones signal. The module also supports clearing of the score through the clear signal, and resetting of the score to zero through the Reset signal.

Purpose:
This module is essential for keeping track of the score in a game and displaying it to the user. It allows the user to see how well they are performing on the game page.

Module: top.sv

Inputs:
Clk
KEY[1:0]: 2-bit input for key presses
ARDUINO_IO[15:0
ARDUINO_RESET_N

Outputs:
HEX0, HEX1, HEX2, HEX3: 7-segment display output signals (for testing and preventing quartus from ignoring our score outputs for some weird reason)
VGA_R, VGA_G, VGA_B: 4-bit output signals for red, green, and blue colors used in VGA
VGA_CLK: output signal for VGA clock
VGA_VS: output signal for VGA vertical sync
VGA_HS: output signal for VGA horizontal sync
DRAM_ADDR: [12:0]
DRAM_DQ: [15:0]
DRAM_BA: [2:0]
DRAM_LDQM, DRAM_UDQM, DRAM_RAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK

Description:
The module "top" is the top-level module of a SoC design. The module includes various input and output signals for different components of the SoC. It also includes various wire and register signals used for interfacing with other modules and components, such as SPI and USB interfaces, and helps us connects all instantiated modules together.

Purpose:
The "top" module serves as the main interface for the SoC design, providing input and output signals for various components.

Module: vga_controller

Inputs:
Clk, Reset, pixel_clk

Outputs:
hs: Horizontal sync signal

vs: Vertical sync signal
blank: Blank signal
sync: Sync signal
DrawX: X-coordinate of the pixel to be drawn on the VGA screen
DrawY: Y-coordinate of the pixel to be drawn on the VGA screen

Description: This module is responsible for generating the necessary sync signals and pixel coordinates required to display the image on the VGA screen. The module generates the horizontal sync (hs), vertical sync (vs), blank, and sync signals as outputs. It also generates the pixel coordinates (DrawX and DrawY) for the pixel to be drawn on the VGA screen at the current clock signal.

Purpose: This module generates the necessary signals to synchronize and display the image on the VGA screen. It takes the clock and pixel clock signals as inputs and generates the necessary signals required for the VGA monitor to display the image correctly.

Module: color_mapper

Inputs:
myBall: the location of the ball in the game
blank: a blanking signal used to reset the VGA display
vga_clk: the clock signal used to synchronize the VGA display
DrawX, DrawY: the X and Y coordinates of the pixel currently being drawn on the VGA display
Sprite_dx, Sprite_dy: the change in X and Y position of the sprite in the game
Sprite_DXF, Sprite_DYF, Sprite_DXF2, Sprite_DYF2: the X and Y coordinates of the first and second pokeball sprites in the game
pokemon: a signal indicating which Pokemon sprite is being used in the game
pokeball: a signal indicating which type of pokeball is being used in the game
my_Sprite: the location of the sprite in the game
my_SpriteFire: the location of the fire animation in the game
my_SpriteFireTWO: the location of the second fire animation in the game
H_STATE: a signal indicating the current state of the horizontal movement of the game
E_STATE: a signal indicating the current state of the game
menu_screen: a signal indicating if the menu screen is currently being displayed
ones, tens: signals indicating the ones and tens digit of the score in the game
C_pic: the bit-map that needs to be displayed on the VGA display

Outputs:

VGA_R, VGA_G, VGA_B: the red, green, and blue color values of the current pixel being drawn on the VGA display

Description:
The color_mapper module is responsible for mapping the different signals from the game into corresponding VGA color values for each pixel being drawn on the VGA display. The module includes several sub-modules to handle different tasks, including drawing the maze, displaying the menu screen, and displaying the pokeball and Pokemon sprites. The input signals are used to determine which sub-module should be active and what color values should be assigned to each pixel.

Purpose:
The purpose of the color_mapper module is to convert the various game signals into corresponding VGA color values so that the game can be displayed on the VGA display. It also includes instantiations of sub-modules for displaying different game elements, such as the maze, Pokemon, and pokeball sprites, allowing the colors at current positions at a certain clock signal to be displayed correctly.

Module: pokeball

Inputs: Clk, Reset, frame_clk, spr_on, inc, DrawX, DrawY, keycode

Outputs: Sprite_dx, Sprite_dy, sprite_W, sprite_H, sprite_xpos, sprite_ypos, my_Sprite

Description: The module controls the position and motion of a sprite, which in this case is a pokeball, on the screen. It takes inputs from the VGA controller, including Clk, Reset, frame_clk, DrawX, DrawY, and keycode, and generates outputs to control the sprite's motion and position, including Sprite_dx, Sprite_dy, sprite_W, sprite_H, sprite_xpos, sprite_ypos, and my_Sprite.
The sprite's position is controlled by the module's internal logic, which calculates the new position of the sprite based on the frame_clk input and the sprite's current motion. The sprite's motion is set to a default value of 0 when the module is reset or when the input inc is received. The module then calculates the new position of the sprite by adding the sprite's current motion to its current position. It then determines whether the sprite should be displayed on the screen based on its position and dimensions. If the sprite is within the bounds of the screen, my_Sprite is set to 1; otherwise, my_Sprite is set to 0.

Purpose: The pokeball module is responsible for controlling the sprite's position and motion and determining whether the sprite should be displayed on the screen. The

module takes inputs from the VGA controller and generates outputs that control the sprite's motion and position.

Platform Designer Module Description:

Blocks we included for platform designer:

- Clk_0: block for clocks
- Nios2_gen2_0: Nios II Processor, as a 32 bit CPU controller, executes code stored in the memory.
- SDRAM: Memory block for storing code and data
- Spi_0: serial peripheral interface that allows the processor to communicate with external devices using a synchronous serial protocol.
- Hex_digits_pio, key: input/output devices that allow the user to interact with the system using buttons and displays.
- Timer_0: a timer peripheral that can generate interrupts to the processor at specific time intervals.
- Key: button on FPGA
- jtag_uart_0: a UART peripheral that allows the processor to communicate with external devices, in this case helps us with debugging our code in eclipse.
- VGA_text_mode_controller_0: designed to generate VGA signals for displaying text characters on a VGA monitor.
- PIOs: keycode, usb_irq, usb_gpx,usb_rst (reset controller)

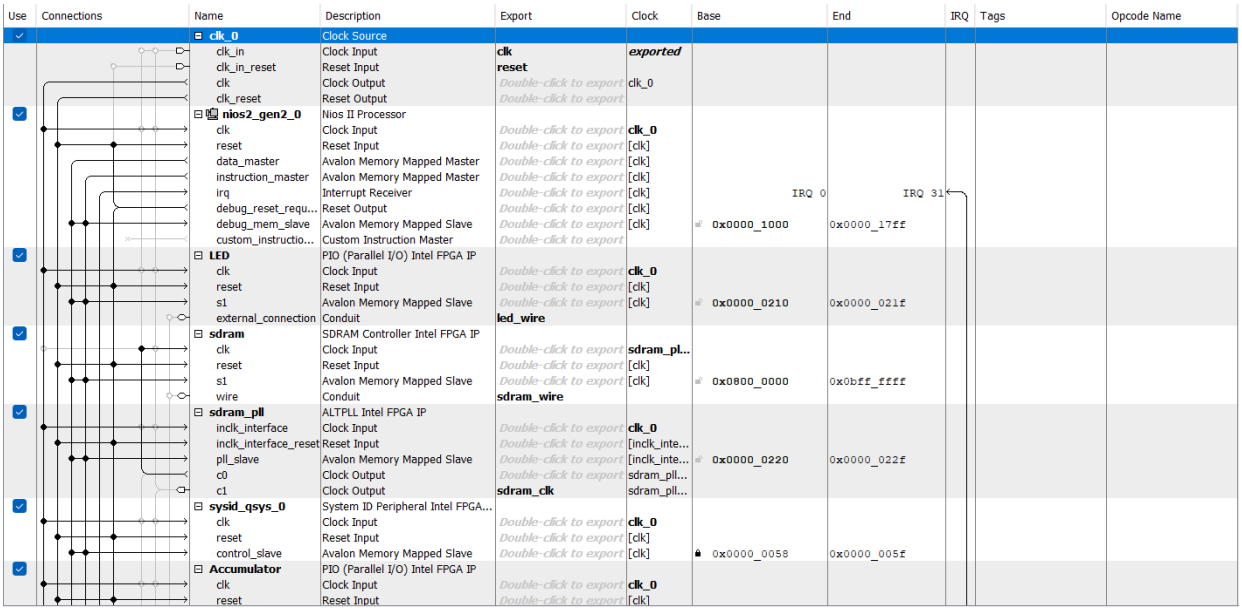

Image 14: Platform designer-1

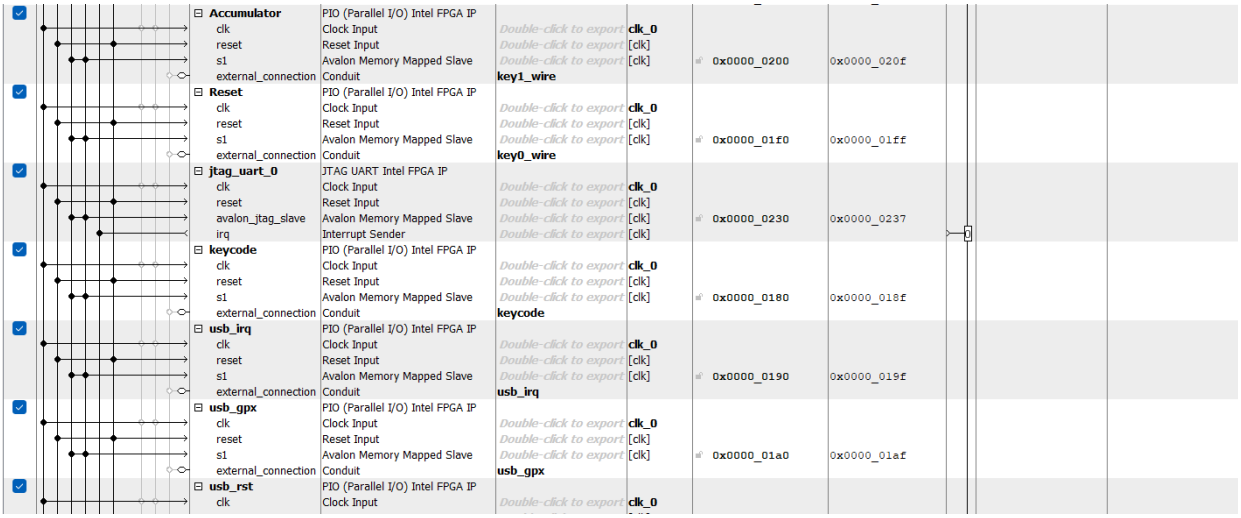

Image 15: Platform designer-2

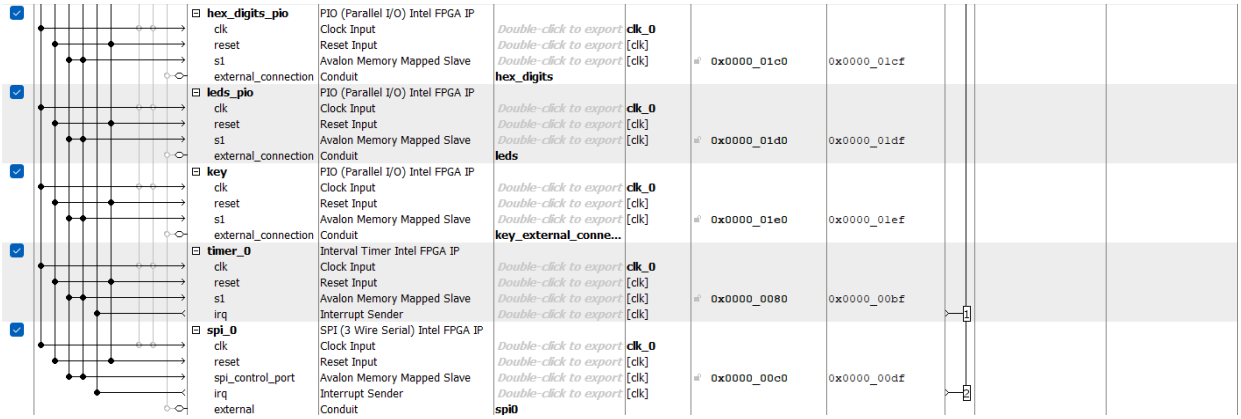

Image 16: Platform designer-3

Design Resources and Statistics:

LUT	2713
DSP	0
Memory(BRAM)	671,952 bits
Flipflop	14,146
Frequency	108.62 MHz
Static Power	102.42 mW
Dynamic Power	0.71 mW
Total Power	112.27 mW

Table: Final Table

Conclusion:

In conclusion, we chose to create and finish making a Pokemon styled maze game for the final project. The main objective of the game was to navigate the Charmander character to the endpoint of the maze using the wired keyboard, allowing movement in the all necessary direction. The goal was to guide the sprite from the starting point to the finish line in each of the three mazes while ensuring that the Charmander does not collide with the walls more than 10 times or collide with the pokeball, as doing so would result in a game over.

The lab provided a valuable experience where we could showcase the skills learned since the beginning of the course, while also acquiring new skills such as utilizing images into the code and implementing features like collision detection and score counters to enhance the game experience. At the end, we were able to make the game work successfully and demonstrate to the TA the different features of the game we constructed. Although we were finished with the game a few days prior to the demonstration to the TA, we had issues when trying to add a certain feature to the game. Furthermore, we wanted to add a menu screen that would allow the user to choose between 3 different pokemon for the user to use to move around the mazes and choose between 4 different pokeballs for the enemy in the game. Although we successfully implemented the different sprites and added another state in the game state machine and created a mux for the different pokeballs and pokemon to choose from for the color mapper module, we were unable to fully add it to the game. Furthermore, after successfully choosing the pokemon and pokeball of the users choosing and leaving the menu screen, the game goes to the first maze but is frozen. We did not have enough time to resolve this issue. Other features we would have love to add to the game if we had more time was the ability to play multiplayer to compete in a race style, add sound, and add a timer for the amount of time it takes the user to complete the maze.

Image 14: Menu Screen we Attempted to Function with our Project

For the next semester, we would like to recommend another small required/optional lab assigned to teach students the ability to create sprites and understand how to align the sprites onto the VGA display in a correct manner. This would be beneficial as it took quite the amount of time to truly understand how to implement a sprite into the code and also being able to display it onto the screen. Besides that, we believe the UAs were truly helpful throughout our completion of the lab and helped us everytime we were confused about how to implement certain things to our project.