**ECE385**

**Spring 2019**

**Experiment #8**

# SOC with USB and VGA Interface
# In SystemVerlog

Austing Lee (sal3)

Zuling Chen (zulingc2)

Tuesday 3PM ABE

Patrick Wang, Gene Shiue

**Introduction**

This week's lab was meant to teach us how to write code and work with Quartus to display data from the FPGA board onto a monitor through VGA and how to connect a USB keyboard input to the FPGA. The design was supposed to be display the ASCII value of the key that was pressed on the keyboard onto the hex displays and if one of those keys were either W,A,S or D, make the bouncing ball move in that direction. The ball was also to bounce off the edges of the screen. USB (Universal Studio Bus) is a used norm that defines methods of communication between devices through a certain port. On the FPGA, there is a USB controller that handles data that goes through the port. There is also software inside the NIOSII chip that takes the keycodes from the USB keyboard input. VGA on the otherhand is the norm that defines the display as a matrix of pixels. One line at a time, a beam will print each individual pixel.

**Written Description of Lab 8 System**

For this lab, we started with a bouncing ball in the middle of the screen. Our task was to design a system where keypresses and their ASCII values would be displayed on a HEX display and keypresses corresponding to W,A,S,D would change the ball's direction. The ball had boundaries at each side of the screen. We wanted the ball to bounce off whenever it hit an edge and we did not want the ball to have any diagonal movement.

NIOS interacts with the keyboard USB device through the USB EZ-OTG standard IO handler. The EZ-OTG is constantly polling for data and whenever it receives an interrupt signal is when it would read data from the keyboard. For the actual interface, we defined a few signals, include data from the keypress, chip select, memory address, and a few read/write enables. Data from the keyboard would then be sent to our chip in the form of an ASCII value, which we then took and interpreted to perform another function. The VGA controller generated a new image every time the ball moved to a new spot or changed direction due to one of these key presses.

**USB Protocol**

The USB read function reads the data that was stored inside the USB controller's internal registers. The USB write function allowed us to write data into the internal registers. IO_read handled the delay and timing needed so that values from the internal registers could be read and be used to update the ball's movement as needed. IO_write handled the delay and timing needed so that data could be written from the internal registers into the FPGA.

**Modules**

Module: lab8
Inputs: CLOCK_50, [3:0] KEY, OTG_INT
Outputs: [6:0] HEX0, [6:0] HEX1, [7:0] VGA_R, [7:0] VGA_G, [7:0] VGA_B, VGA_CLK, VGA_SYNC_N, VGA_BLANK_N, VGA_VS, VGA_HS, [1:0] OTG_ADDR, OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N, [12:0] DRAM_ADDR, [1:0] DRAM_BA, [3:0] DRAM_DQM, DRAM_RAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK
Inout: [15:0] OTG_DATA, [31:0] DRAM_DQ,
Description: Top-level module for lab8.
Purpose: Instantiates all other required modules and connects relevant input/outputs

Module: ball.sv
Inputs: Clk, Reset, frame_clk, [7:0] keycode, [9:0] DrawX, [9:0] DrawY
Outputs: is_ball
Inout:
Description: Contains information and instructions regarding the ball's movement
Purpose: Detects keyboard input and changes ball's movements. Also details boundary/edge-cases and computes pixel color for background/ball

Module: HexDriver.sv
Inputs: [3:0] In0
Outputs: [6:0] Out0
Inout:
Description: Changes the 4 bit input in their corresponding HEX values
Purpose: This is used by the FPGA to display hex values for the input from the keyboard.

Module: Color_Mapper.sv
Inputs: [9:0] DrawX, [9:0] DrawY, is_ball
Outputs: [7:0] VGA_R, VGA_G, VGA_B
Inout:
Description: Assigns color of the pixel depending on whether of not the pixel is supposed to be the ball.
Purpose: This is used to display the actual moving ball and background onto the FPGA monitor.

Module: hpi_io_intf.sv
Inputs: from_sw_r, from_sw_w, from_sw_cs, OTG_INT, Clk, Reset, [1:0] from_sw_address, [15:0] from_sw_data_out,

Outputs: [15:0] from_sw_data_in, [1:0] OTG_ADDR, OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N
Inout:[15:0] OTG_DATA
Description:The module corresponding to the EZ-OTG chip and determines which way data goes.
Purpose: This module was used to connect USB input-output data and sends the right data depending on select signals.

**Block diagram**

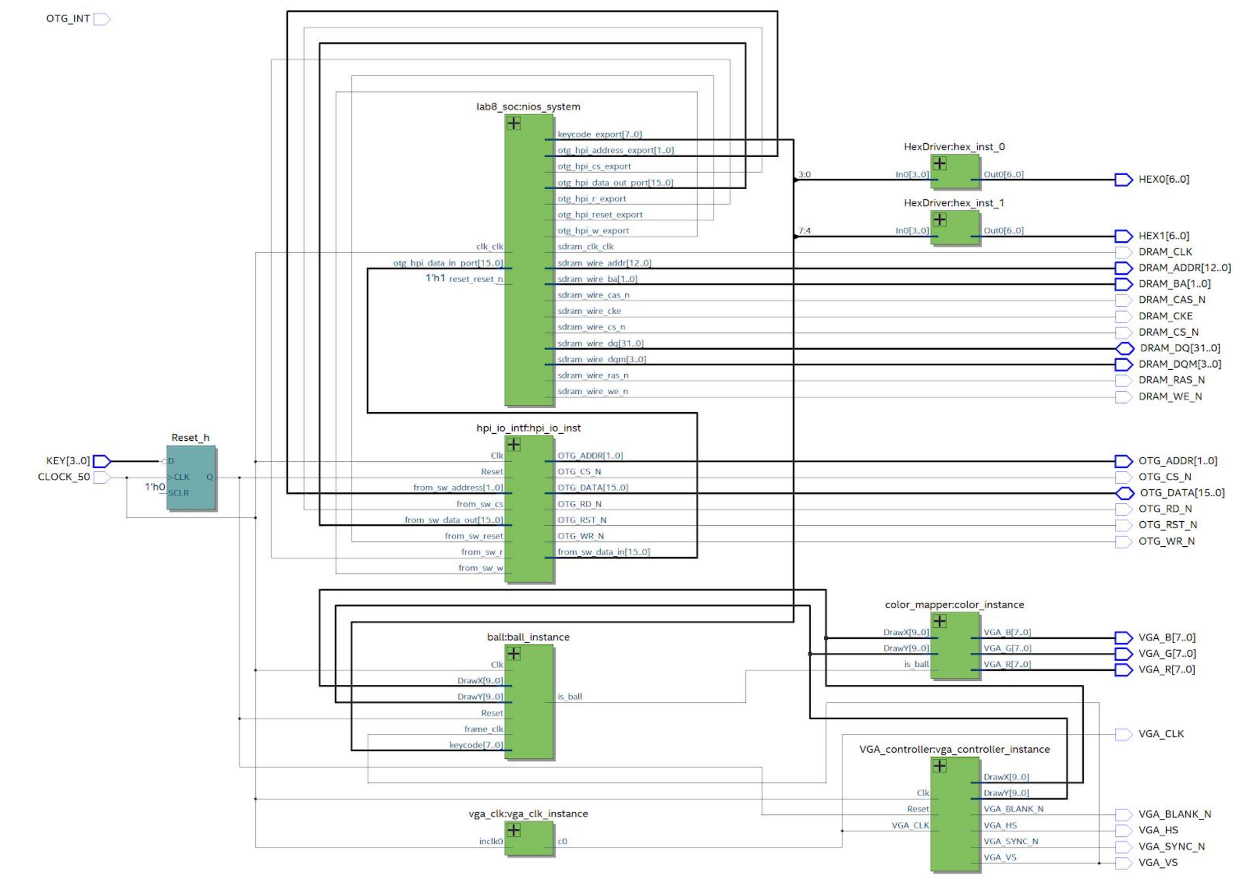

Figure 1: Lab 8 Top Level block diagram

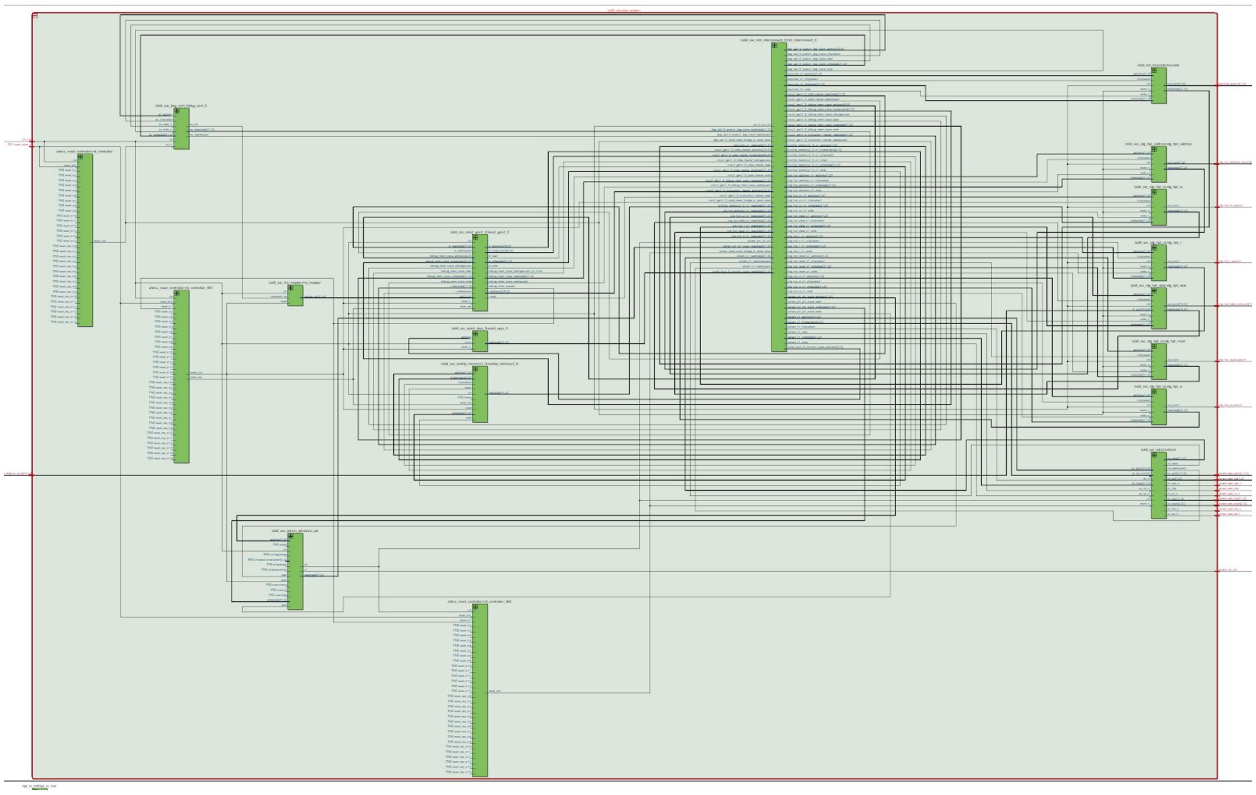
Figure 2: NIOS system

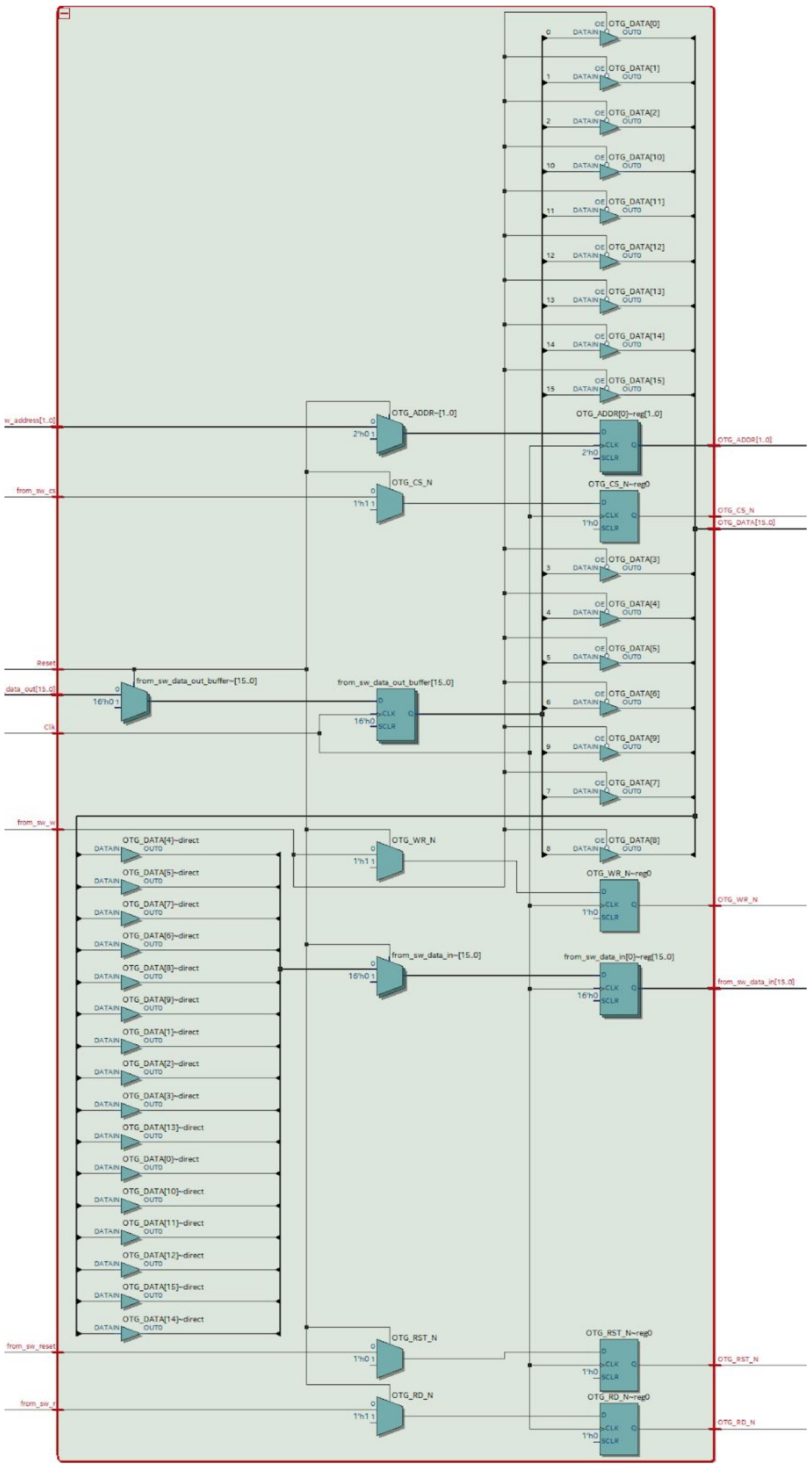
Figure 3: hpi_io interface

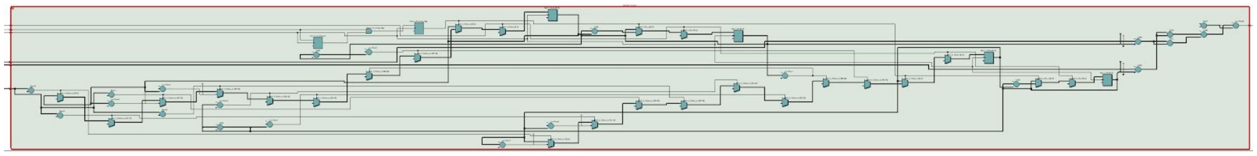
Figure 4: Ball

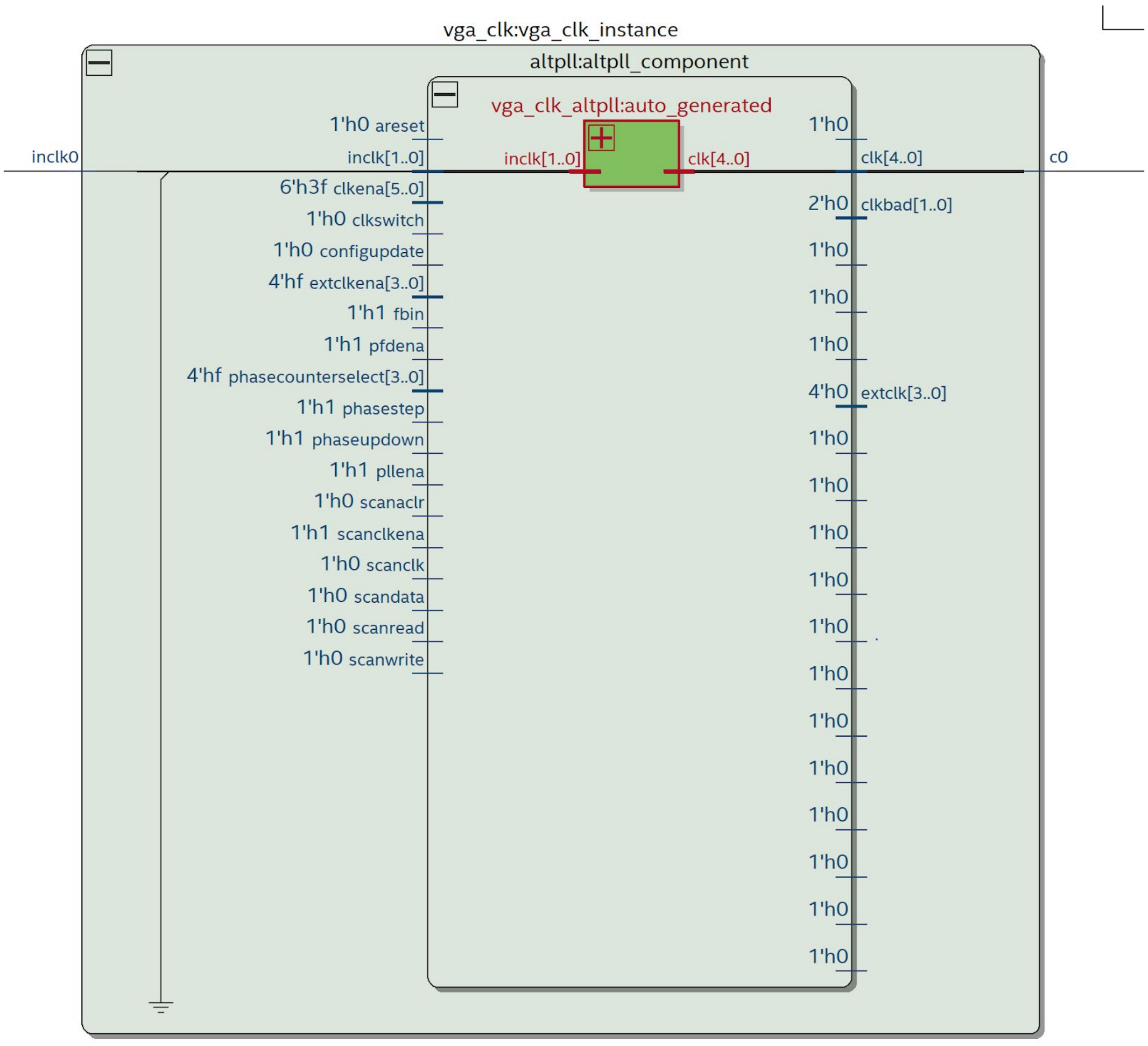

Figure 5: VGA Clock

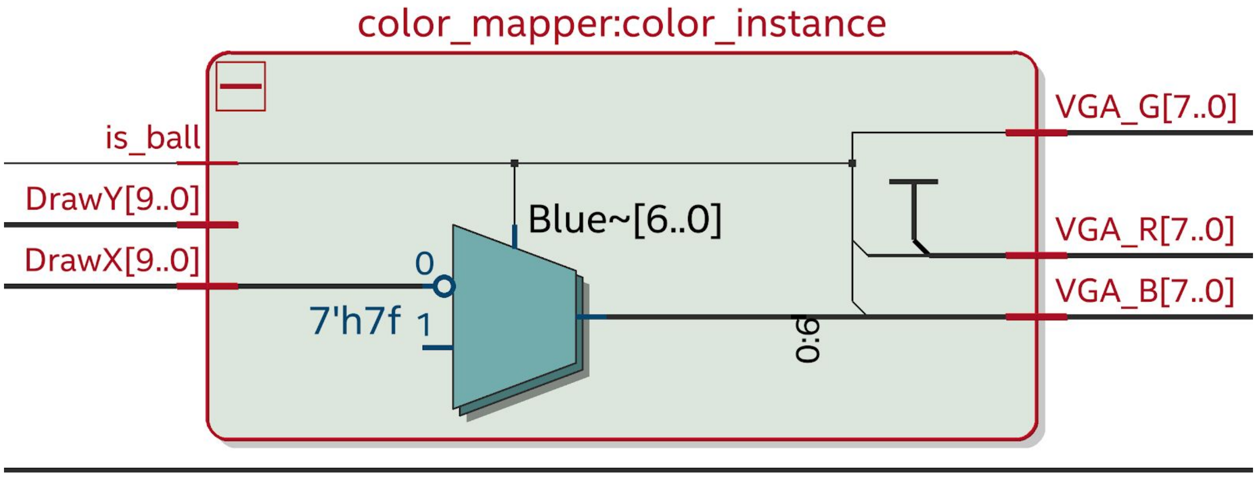

Figure 6: Color mapper

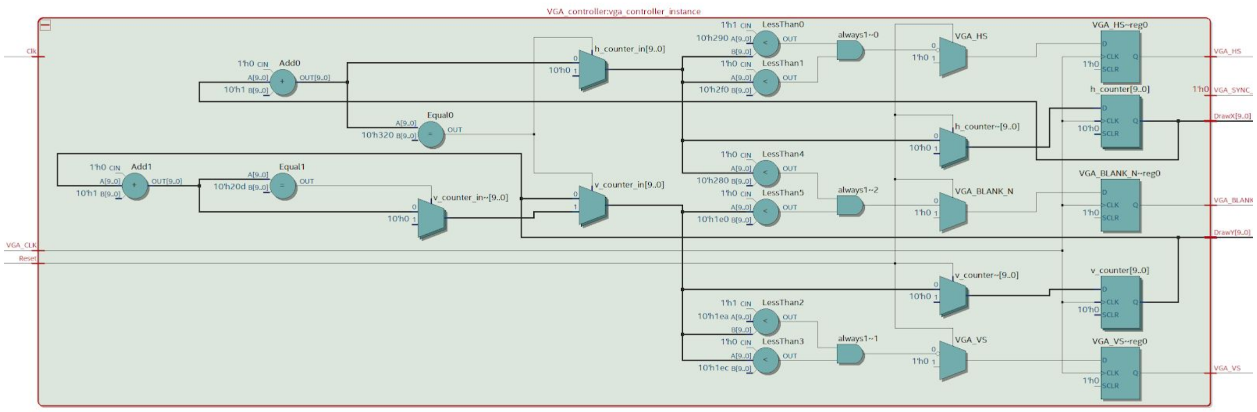

Figure 7: VGA Controller

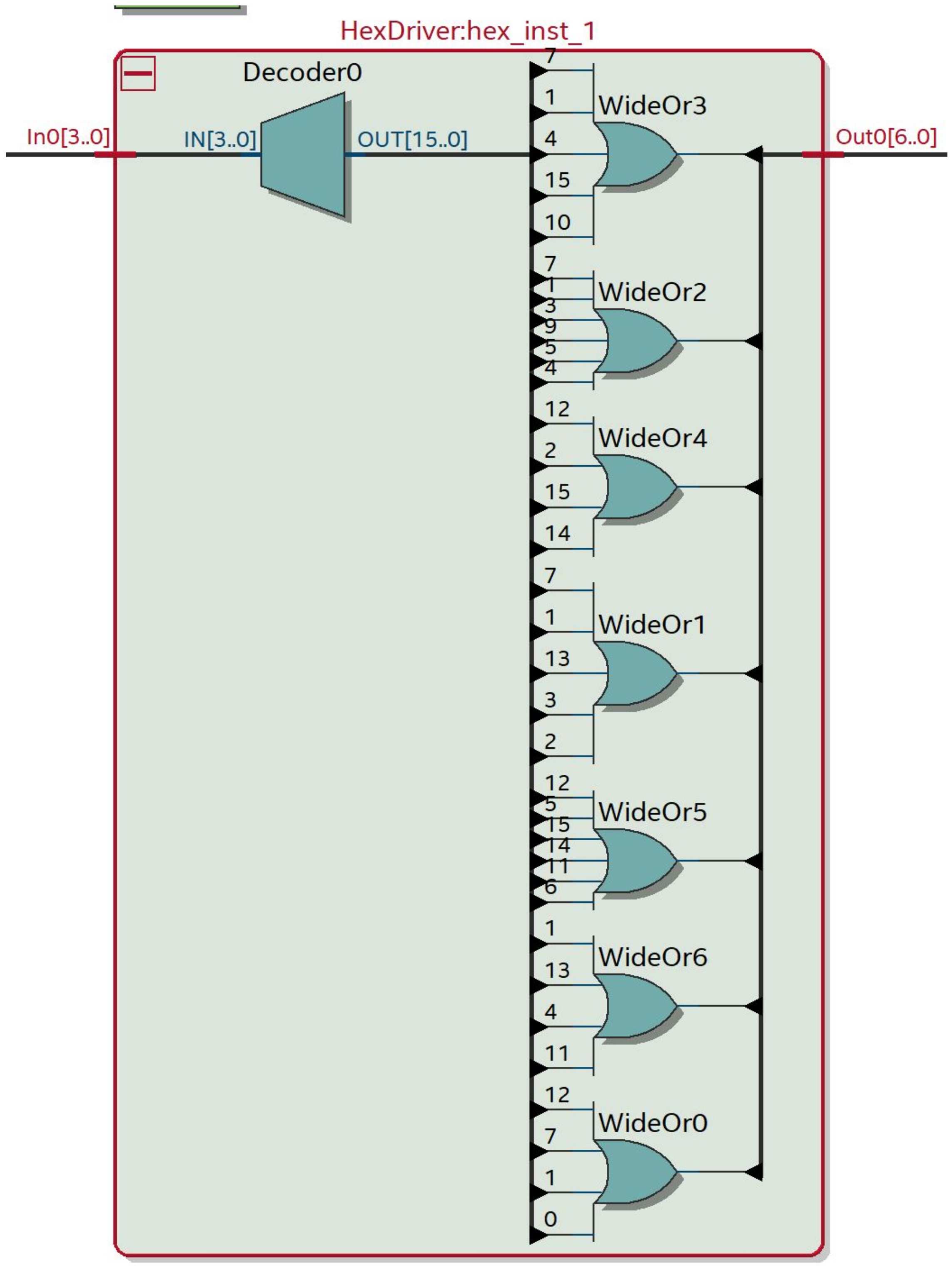

Figure 8: Hex Drivers

**Hidden Questions**

1. By using parallel assignments, we change the value of Ball_Y_Motion to the old value rather than the newer one. If this issue occurred at the very edge, then for one frame, the ball would move into the wall. If we didn't account for this by pressing the key as soon as it hits the edge, then the ball's motion would go into the wall. A quick fix to this could be to use an if-else block for the assignment so that the position and movement are updated separately instead of at the same time.
2. PS/2 is cheaper to build, make and implement, as well as being faster at data transfer. USB only transfers data when the host requests it.

**Postlab Questions**

1. VGA_clk ran at 25MHz and updated the frames on the monitor while Clk ran at 50 MHz and drove the FPGA board operations.
2. Otg_hpi_r is a signal that is read that determines whether or not the OTG interface is ready to be read, so it only need be a char. Otg_hpi_address passes along an address so it would require an int rather than a char.
3.

| | |
|---|---|
| LUT | 2,380 |
| DSP | 0 |
| Memory(BRAM) | 11392 |
| Flip-Flop | 2140 |
| Frequency | 56.8 MHz |
| Static Power | 102.01 mW |
| Dynamic Power | 0.81 mW |
| Total Power | 161.31 mW |

**Conclusion**

For our lab, we ran into some syntax issues that were resolved rather quickly. It was then that we couldn't get our ball to bounce off the sides of the walls, even with the given code. We tried using cases to deal with key presses but we could not get it to work properly. After switching to if-else statements instead of using cases, we finally got the ball moving in the right direction. At the end, we were able to get a fully functional project.