ECE 385

Spring 2020

Experiment 8

SOC with USB and VGA Interface in SystemVerilog

Michael Faitz and Zohair Ahmed

Section ABJ: Friday 2:00-4:50

Yuming Wu and Lian Yu

Introduction

The operation of the USB/VGA interface in this lab was to teach us how to implement new forms of I/O in our programs. In this lab, we wrote C code that processed key presses from a keyboard and displayed it on the FPGA, as well as SystemVerilog code that allowed us to change the behavior of a bouncing ball with those presses. This behavior was displayed on a monitor using the VGA output capabilities.

Written Description of Lab 8 System

a. Written Description of Entire Lab 8 System

i. The NIOS processor has access to the addresses of the CY7 chip, allowing it read/write capabilities. From this, we can read the inputs from the USB chip, which we process with functions in IO_handler and usb files. By using a similar set of modules in platform designer to lab 7, we already set up clocks that ensure correct data transfers will occur. We additionally created modules for the otg_hpi variables we used in our C code, to ensure proper handling of interfacing with the CY7. The VGA output required that we had the VGA_controller module, which produced the timing signals needed to ensure proper synchronization with the DE2's VGA output. These timing signals ran at a frequency that, when used with the 50 MHz clock of the FPGA, allowed us a 60Hz framerate for the VGA output. We also used the Color_Mapper module to handle object rendering and coloring, such that we could see the ball and the rest of the screen.

b. Written Description of the USB Protocol

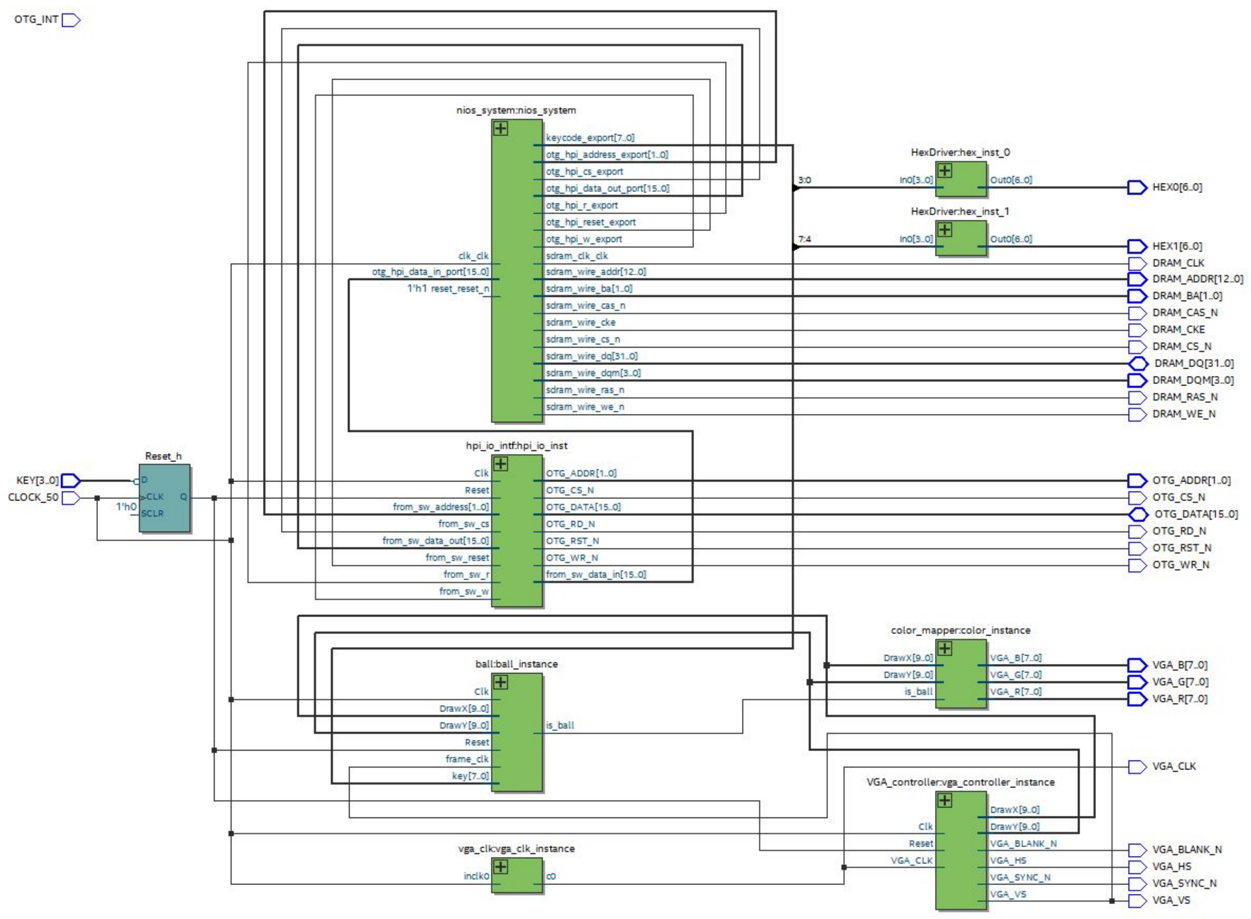
IO_Write Purpose:
Writes data from the Address and Data inputs to the address and data pointers to the USB Controller registers.

IO_Read Purpose:
Returns the 16 bit data held in otg_hpi_data after updating otg_hpi_address to a given 8 bit input address.

USBWrite Purpose:
Instantiates two uses of IO_Write per function in order to write the address and data values to the internal registers of the USB controller.

USBRead Purpose:
Reads data from the registers in the CY7C67200 USB Controller. Instantiates one use of IO_Write for the address and returns the data found in that address via IO_Read.

c. Block Diagram



d. Module Descriptions

```
hpi_io_intf( input        Clk, Reset,
             input [1:0]  from_sw_address,
             output[15:0] from_sw_data_in,
             input [15:0] from_sw_data_out,
             input        from_sw_r, from_sw_w, from_sw_cs, from_sw_reset, // Active low
             inout [15:0] OTG_DATA,
             output[1:0]  OTG_ADDR,
             output       OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N // Active low
            );
```

Description: The variables containing sw are updated by outside modules, and the otg variables are updated to the sw variables as needed. otg variables represent the USB chip inputs.
Purpose: Variables in here need to be updated to ensure correct data reads and writes. The module updates the otg signals such that the C code can correctly read and write from the USB keyboard, or be reset.

```
VGA_controller (input          Clk,        // 50 MHz clock
                               Reset,      // Active-high reset signal
```

```
        output logic       VGA_HS,      // Horizontal sync pulse.  Active low
                           VGA_VS,      // Vertical sync pulse.  Active low
        input              VGA_CLK,     // 25 MHz VGA clock input
        output logic       VGA_BLANK_N, // Blanking interval indicator.  Active low.
                           VGA_SYNC_N,  // Composite Sync signal.  Active low.  We don't use
it in this lab,
                                        // but the video DAC on the DE2 board requires an input for
it.
        output logic [9:0] DrawX,       // horizontal coordinate
                           DrawY        // vertical coordinate
        );
```

Description: Takes the data of the ball's current position on screen and updates it to the VGA display with the variables DrawX and DrawY.
Purpose: Serves as a compatibility file for the VGA display on the board to the available VGA display. Also serves as an available reset for said display with the reset button of the FPGA.

```
lab8( input              CLOCK_50,
      input        [3:0]  KEY,          //bit 0 is set up as Reset
      output logic [6:0]  HEX0, HEX1,
      // VGA Interface
      output logic [7:0]  VGA_R,        //VGA Red
                          VGA_G,        //VGA Green
                          VGA_B,        //VGA Blue
      output logic        VGA_CLK,      //VGA Clock
                          VGA_SYNC_N,   //VGA Sync signal
                          VGA_BLANK_N,  //VGA Blank signal
                          VGA_VS,       //VGA vertical sync signal
                          VGA_HS,       //VGA horizontal sync signal
      // CY7C67200 Interface
      inout  wire  [15:0] OTG_DATA,     //CY7C67200 Data bus 16 Bits
      output logic [1:0]  OTG_ADDR,     //CY7C67200 Address 2 Bits
      output logic        OTG_CS_N,     //CY7C67200 Chip Select
                          OTG_RD_N,     //CY7C67200 Write
                          OTG_WR_N,     //CY7C67200 Read
                          OTG_RST_N,    //CY7C67200 Reset
      input               OTG_INT,      //CY7C67200 Interrupt
      // SDRAM Interface for Nios II Software
      output logic [12:0] DRAM_ADDR,    //SDRAM Address 13 Bits
      inout  wire  [31:0] DRAM_DQ,      //SDRAM Data 32 Bits
      output logic [1:0]  DRAM_BA,      //SDRAM Bank Address 2 Bits
      output logic [3:0]  DRAM_DQM,     //SDRAM Data Mast 4 Bits
      output logic        DRAM_RAS_N,   //SDRAM Row Address Strobe
```

```
                    DRAM_CAS_N,   //SDRAM Column Address Strobe
                    DRAM_CKE,     //SDRAM Clock Enable
                    DRAM_WE_N,    //SDRAM Write Enable
                    DRAM_CS_N,    //SDRAM Chip Select
                    DRAM_CLK      //SDRAM Clock
          );
```

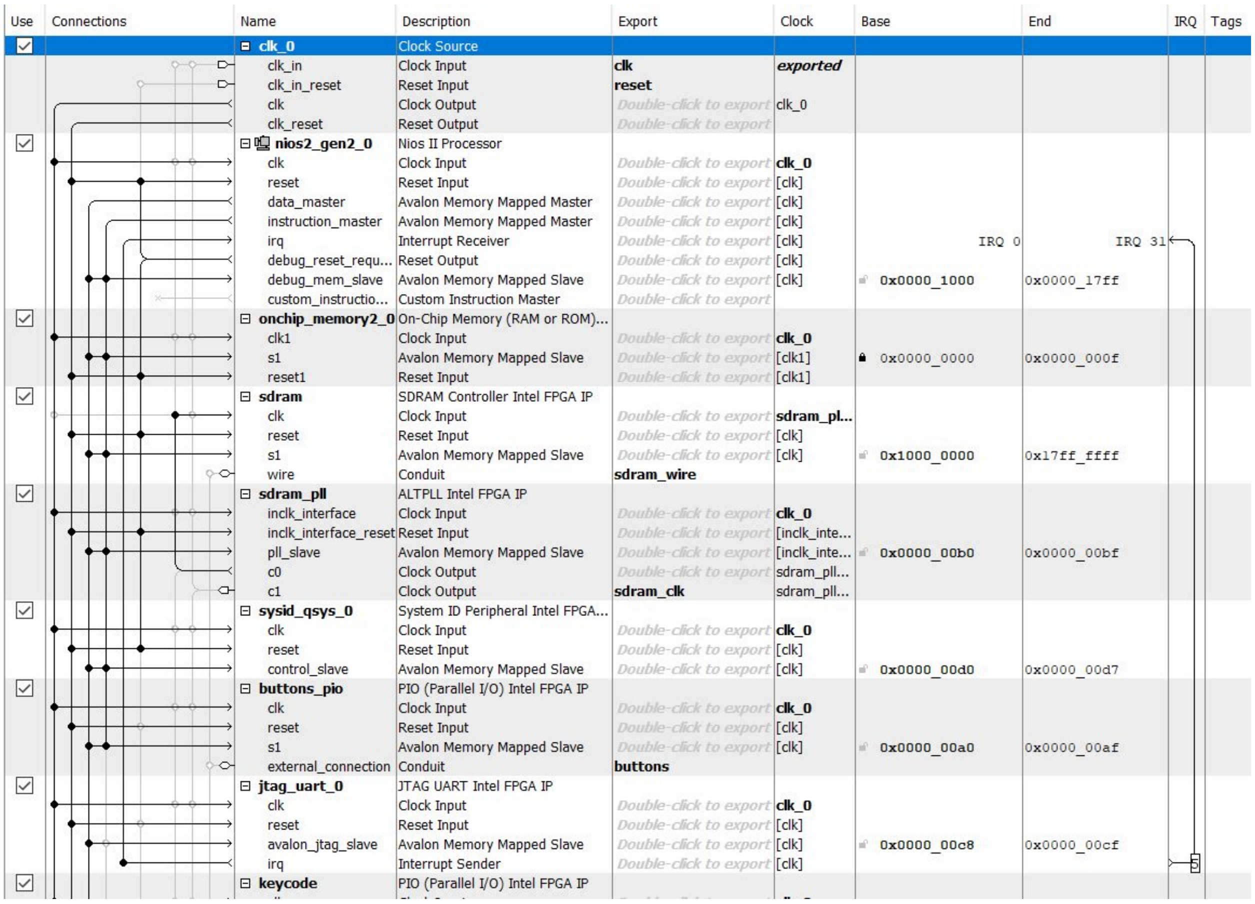
Description: Serves as the top level module and provides the output data for the C code to assign to the board and the usb cable.
Purpose: Instantiates all the other System Verilog modules and links together the entire program with the help of Qsys.

```
color_mapper ( input            is_ball,          // Whether current pixel belongs to ball
                                                  //   or background (computed in ball.sv)
               input        [9:0] DrawX, DrawY,       // Current pixel coordinates
               output logic [7:0] VGA_R, VGA_G, VGA_B // VGA RGB output
              );
```

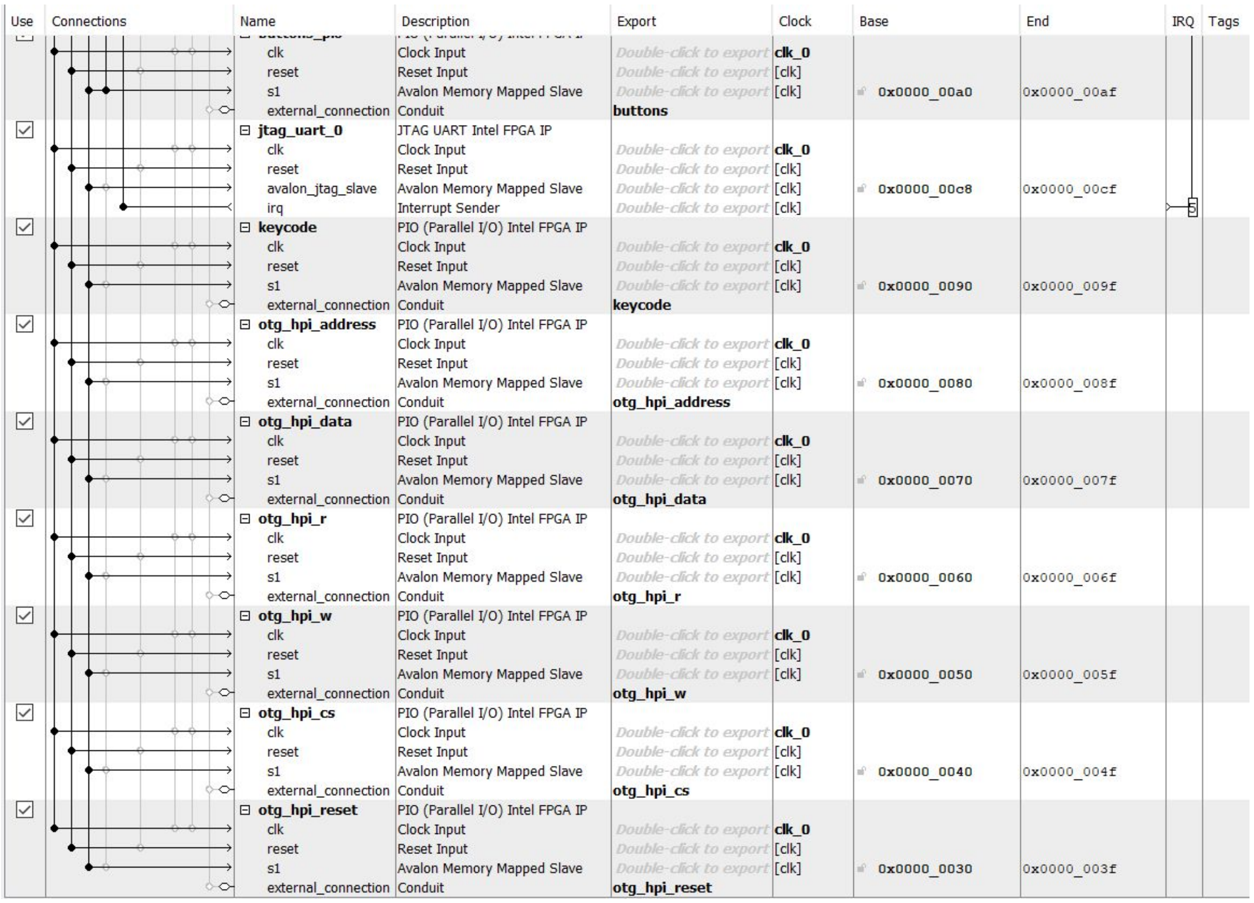
Description:
Purpose:

```
ball ( input        Clk,              // 50 MHz clock
                      Reset,              // Active-high reset signal
                      frame_clk,          // The clock indicating a new frame (~60Hz)
          input [9:0]   DrawX, DrawY,           // Current pixel coordinates
                                             input [7:0]   key,
          output logic  is_ball            // Whether current pixel belongs to ball or background
         );
```

Description: Module containing ball object, takes in current pixel coordinates as well as keypress to determine behavior
Purpose: Using coordinates and is_ball, determines which pixels draw the ball. Input key handles changes in movement direction. Also updates coordinates to be displayed through VGA output.



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
✓		**clk_0**	Clock Source						
		clk_in	Clock Input	**clk**	***exported***				
		clk_in_reset	Reset Input	**reset**					
		clk	Clock Output	*Double-click to export*	clk_0				
		clk_reset	Reset Output	*Double-click to export*					
✓		**nios2_gen2_0**	Nios II Processor						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]				
		instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]				
		irq	Interrupt Receiver	*Double-click to export*	[clk]	IRQ 0	IRQ 31		
		debug_reset_requ...	Reset Output	*Double-click to export*	[clk]				
		debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0000_1000**	0x0000_17ff		
		custom_instructio...	Custom Instruction Master	*Double-click to export*					
✓		**onchip_memory2_0**	On-Chip Memory (RAM or ROM)...						
		clk1	Clock Input	*Double-click to export*	**clk_0**				
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	0x0000_0000	0x0000_000f		
		reset1	Reset Input	*Double-click to export*	[clk1]				
✓		**sdram**	SDRAM Controller Intel FPGA IP						
		clk	Clock Input	*Double-click to export*	**sdram_pl...**				
		reset	Reset Input	*Double-click to export*	[clk]				
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x1000_0000**	0x17ff_ffff		
		wire	Conduit	**sdram_wire**					
✓		**sdram_pll**	ALTPLL Intel FPGA IP						
		inclk_interface	Clock Input	*Double-click to export*	**clk_0**				
		inclk_interface_reset	Reset Input	*Double-click to export*	[inclk_inte...				
		pll_slave	Avalon Memory Mapped Slave	*Double-click to export*	[inclk_inte...	**0x0000_00b0**	0x0000_00bf		
		c0	Clock Output	*Double-click to export*	sdram_pll...				
		c1	Clock Output	**sdram_clk**	sdram_pll...				
✓		**sysid_qsys_0**	System ID Peripheral Intel FPGA...						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		control_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0000_00d0**	0x0000_00d7		
✓		**buttons_pio**	PIO (Parallel I/O) Intel FPGA IP						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0000_00a0**	0x0000_00af		
		external_connection	Conduit	**buttons**					
✓		**jtag_uart_0**	JTAG UART Intel FPGA IP						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0000_00c8**	0x0000_00cf		
		irq	Interrupt Sender	*Double-click to export*	[clk]			5	
✓		**keycode**	PIO (Parallel I/O) Intel FPGA IP						



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_00a0	0x0000_00af		
		external_connection	Conduit	buttons					
✓		⊟ jtag_uart_0	JTAG UART Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_00c8	0x0000_00cf		
		irq	Interrupt Sender	Double-click to export	[clk]			5	
✓		⊟ keycode	PIO (Parallel I/O) Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0090	0x0000_009f		
		external_connection	Conduit	keycode					
✓		⊟ otg_hpi_address	PIO (Parallel I/O) Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0080	0x0000_008f		
		external_connection	Conduit	otg_hpi_address					
✓		⊟ otg_hpi_data	PIO (Parallel I/O) Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0070	0x0000_007f		
		external_connection	Conduit	otg_hpi_data					
✓		⊟ otg_hpi_r	PIO (Parallel I/O) Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0060	0x0000_006f		
		external_connection	Conduit	otg_hpi_r					
✓		⊟ otg_hpi_w	PIO (Parallel I/O) Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0050	0x0000_005f		
		external_connection	Conduit	otg_hpi_w					
✓		⊟ otg_hpi_cs	PIO (Parallel I/O) Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0040	0x0000_004f		
		external_connection	Conduit	otg_hpi_cs					
✓		⊟ otg_hpi_reset	PIO (Parallel I/O) Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0030	0x0000_003f		
		external_connection	Conduit	otg_hpi_reset					

Qsys Modules:

nios2_gen2_0 Block: The nios2 block handles the C code conversion over to system verilog which then is able to be executed within the hardware FPGA board.

Onchip_memory2_0: The memory block acts as available hardware memory for variables needed to execute the C code onto the board. This effectively acts as a compliment to the nios2 processor.

Clk_0 (not shown in screenshot): Serves as the functional clock for the entire system and acts as a reference for the other modules.

sdram: This block allows us to get available SDRAM on the FPGA board since on-chip memory is too small for the available program to be stored and updated successfully. SDRAM is useful due to its quick processing time and low update and output delay.

sdram_pll: This block is the method to account for the small delays within the transfer of the data in and out of the SDRAM and acts as a separate clock for the system.

sysid_qsys_0: This block verifies the correct transfer of the software and hardware by looking back and forth between the C code and SystemVerilog to assure the data transfer is done in the correct format.

buttons_pio: The inputs are the buttons on the FPGA device. This PIO allows for bidirectional data transfer from the FPGA to software and visa versa.

jtag_uart_0: Allows for terminal access for use in debugging the software.

keycode: Reads data, updates the USB chip and then sends it to the software for logic instruction.

otg_hpi_address: PIO that allows the desired address in memory of the SoC to be found that is sent from the software to the FPGA.

otg_hpi_data: PIO that allows for cross transfer of data from the FPGA to software and the other way around. This PIO has a width of 16 bits which allows for sizable data transfer between the two.

otg_hpi_r: PIO that allows the enable bit to read from the memory of the SoC that is sent from the software to the FPGA.

otg_hpi_w: PIO that allows the enable bit to write to the memory of the SoC that is sent from the software to the FPGA.

otg_hpi_cs: PIO that allows the enable bit turn on and off the memory of the SoC that is sent from the software to the FPGA.

otg_hpi_reset: PIO that allows for the reset of the memory of the SoC sent from the software to the FPGA.

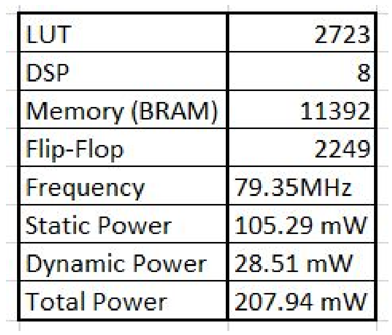
Answers to Both Hidden Questions

a. PS/2 is more secure, as it cannot have files sent through it unlike a USB interface. Therefore, buying a PS/2 keyboard is guaranteed to be safe, unlike USB where the behavior cannot be expected. It also has lower latency, allowing for less lag after keypresses. However, a PS/2 keyboard needs to be plugged in when booting up the system, unlike a USB which can be unplugged and plugged back in with ease.
b. The new value of Ball_Y_Motion will use the old value of Ball_Y_Pos due to parallel assignments. However, the parallel assignments between our always_ff and always_comb blocks limit the time that the new value of Ball_Y_Motion will be outdated such that we do not notice. The difference between those two statements is that one takes advantage of the parallel assignments, whereas the other will result in data corruption due to consecutive writing to the same variable. If we update Ball_Y_Pos_in

with Ball_Y_Motion_in, we will do this. While the ordering and choices of these variables updating when they do, the motion lags slightly. The ball will still avoid going through the boundary so long as our boundary conditions are correct. Regarding keypresses, we simply processed those before our boundary conditions to avoid boundary breaking.

Answer to Post Lab Questions

a. The VGA_clk runs at a lower frequency, 25MHz, and is used only to update the monitor and its elements. The difference in frequency is to ensure that the monitor output operates at 60Hz. The Clk is used to run the rest of the program, such as gathering the input of key presses and processing them.
b. otg_hpi_data is an integer and represents an inout variable, which handles data above one byte. otg_hpi_r handles the reading of one character, which requires one byte, and points to a register that holds the value of the key pressed. A character requires one byte of storage, while an integer requires 4. We set otg_hpi_r to be a char pointer as we have no need for the extra allocated memory, and it is the smallest primitive datatype we can use.



LUT	2723
DSP	8
Memory (BRAM)	11392
Flip-Flop	2249
Frequency	79.35MHz
Static Power	105.29 mW
Dynamic Power	28.51 mW
Total Power	207.94 mW

Conclusion

Our design functioned almost perfectly and was able to demonstrate all the available demo points, however, the compilation of the software onto the board would sometimes crash. When the software compilation did not crash, however, the program would work fine. Possible fixes for this issue include looking over the provided code and commenting out some of the printf lines to help reduce the strain on our processors. The hardware faults and limitations that we ran into doubled the amount of time we needed to finish this lab. I understand that this issue was significant as it was due to the virus, but hopefully more details can be given about what keyboards will and will not work.