ECE 385

Fall 2021

Experiment #8

Lab 8

SOC with USB and VGA Interface in SystemVerilog

Guanshujie Fu 3190110666

Xiaomin Qiu 3190110717

Lab Section D231 18:00 – 21:00

TA: Yuchuan Zhu

1. Introduction

i. *Briefly summarize the operation of the USB/VGA interface*

The operation of the USB/VGA interface in this lab could control the movement of a ball on a screen in X and Y direction with keyboard inputs WASD. We connected the monitor to the VGA port and the keyboard to the USB port, then we could process key presses from keyboard and display it on the FPGA with C# and change the movement of the ball with the key presses accordingly with SV.

2. Written Description of Lab 8 System

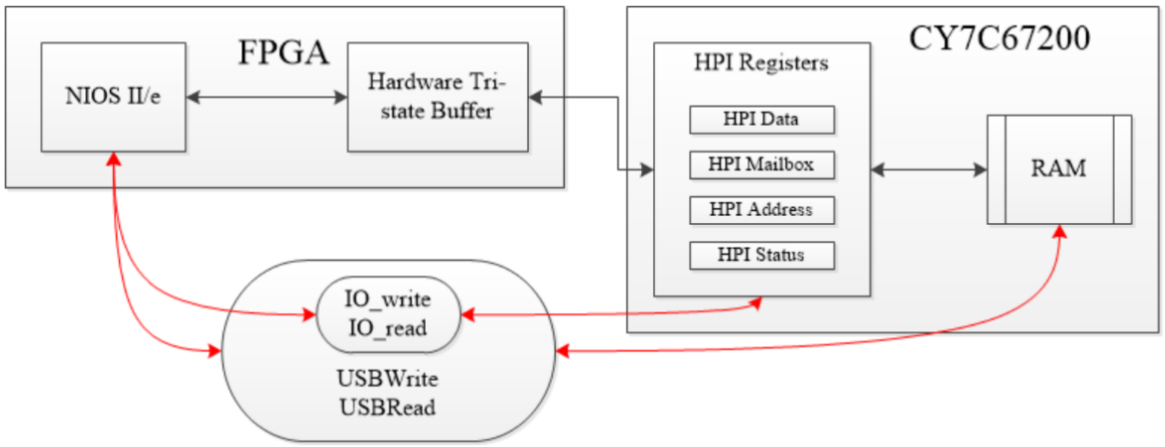
i. *Written Description of the entire Lab 8 system*

The goal of this system is to make a small ball move on the VGA monitor screen. The entire system contains three main parts: NIOS II processor, USB keyboard, and VGA monitor. The NIOS system fetch user's inputs from CY7 USB chip, and then transfer the signal to ball routing and VGA controller to control the VGA monitor to display the corresponding position of the ball. The USB keyboard needs to communicate with the NIOS II through CY7 chip and HPI as the interface. The VGA monitor receives signals generated in the FPGA and use them to display each pixel on the monitor.

a. *Describe in words how the NIOS interacts with both the CY7C67200 USB chip and the VGA components*

In this lab, we first connected a USB keyboard and a VGA monitor to a DE2 board and implemented a USB driver that could write and read from the USB buffer. The top-level module links all the related modules together. Then the DE2 board would first write the specific address to the CY7C67200 USB chip to get data back from it. With the help of the functions in IO_handler as well as USB modules and based on the addressed wrote before, the NIOS processor could read and write data from the USB chip. Meanwhile, variable-update modules for otg_hpi variables have been implemented to ensure correct interfacing with the USB chip and clocks have been set to ensure correct data transfers.

As for the VGA driver, we used the VGA_controller module to produce the timing signals needed to ensure proper synchronization with the DE2's VGA output. The DE2 board with 50 MHz clock generates a special 25 MHz VGA clock to achieve around 60Hz refresh rate for the VGA output which drives the horizontal sync word properly. We also used the Color_Mapper module and ball module to handle object's movement and color on the screen.



ii. *Written description of the USB protocol*

a. *Describe the purpose of the USBRead, USBWrite, IO_read and IO_write functions in the C code*

USBRead Purpose:

It is used to allow reading data from the device such as registers in the CY7C67200 USB Controller. After writing the needed address to the address register, it could read from the data register with the data from device already available and return the data found in that address via IO_Read.

USBWrite Purpose:

It is used to write data from the NIOS processor to the OTG chip. After writing the needed address to the address register with two uses of IO_Write per function, it could write the desired data values to the internal data register of the USB controller and send the data to the host.

IO_Read Purpose:

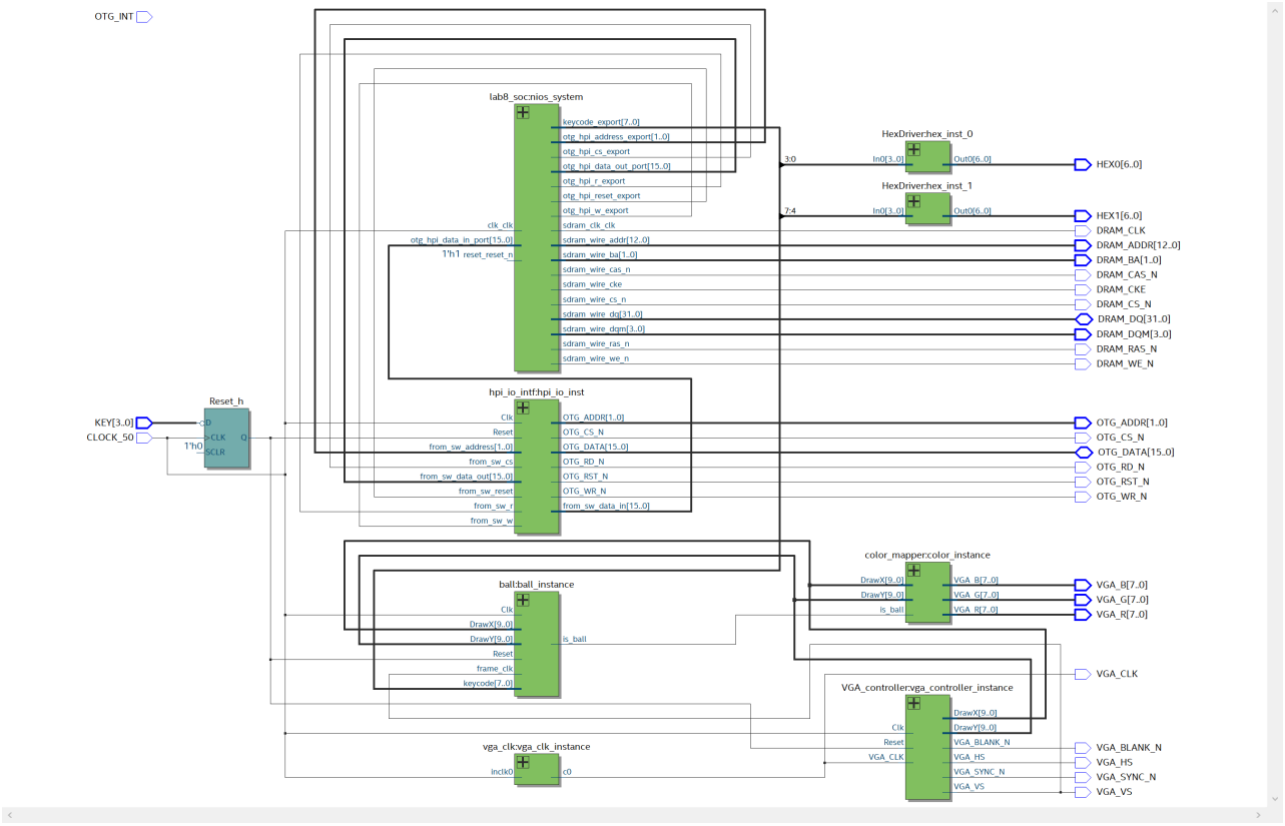
It is used to read the data available from the 8-bits input address we provided it and returs the 16 bit data held in otg_hpi_data after updating otg_hpi_address.

IO_Write Purpose:

It is used to writes data from the required Address and Data inputs to the destination address and data pointers of the USB Controller registers.

iii. *Block diagram*

a. *This diagram should represent the placement of all your modules in the top level. Please only include the top level diagram and not the RTL view of every module. The Qsys view of the NIOS processor is not necessary for this portion.*



iv. *Module descriptions*

a. *A guide on how to do this was shown in the Lab 5 report outline. Do not forget to describe the Qsys generated file for your Nios system! When describing the generated file, you should describe the PIO blocks added beyond those just needed to make the NIOS system run (i.e. the ones needed to communicate with the USB chip).*

Modules	**Lab8.sv**
Inputs	input CLOCK_50, input[3:0] KEY, input OTG_INT,
Outputs	output logic [6:0] HEX0, HEX1, output logic [7:0] VGA_R, VGA_G, VGA_B, output logic VGA_CLK, VGA_SYNC_N, output logic VGA_BLANK_N, VGA_VS, VGA_HS, inout wire [15:0] OTG_DATA, output logic[1:0] OTG_ADDR, output logic OTG_CS_N, OTG_RD_N,

	output logic OTG_WR_N, OTG_RST_N,
Description	It is the top-level module which connects NIOS 2, OTG chip connection and the VGA controller modules.
Purpose	It links all the modules together and instantiates all the System Verilog modules.

Modules	**hpi_io_intf.sv**
Inputs	input Clk, Reset, input [1:0] from_sw_address, input [15:0] from_sw_data_out, input from_sw_r, from_sw_w, from_sw_cs, input from_sw_reset, inout [15:0] OTG_DATA
Outputs	output[15:0] from_sw_data_in, output[1:0] OTG_ADDR, output OTG_RD_N, OTG_WR_N, OTG_CS_N, output OTG_RST_N
Description	This module is used to interfaces with OTG chip which controls the USB and determines the outputted signals. The otg-variables are the inputs of the USB chip, and they are updated to the sw-variables as required.
Purpose	This module controls and updates otg signals like read, write, address, databuffer signals to the OTG chip which ensure the correct data reads, writes and reset from USB keyboard for C#. It allows us to control the input output bus.

Modules	**VGA_controller.sv**
Inputs	input Clk, Reset, input VGA_CLK,
Outputs	output logic VGA_HS, VGA_VS, output logic VGA_BLANK_N, VGA_SYNC_N, output logic [9:0] DrawX, DrawY
Description	This module controls the VGA signals such as the vertical and horizontal syncs. It 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	This module is used to determine the edge of the screen

	and when to output the vertical or horizontal sync signals. It also records the position of the pixel we are working on .

Modules	**ball.sv**
Inputs	input Clk, Reset, frame_clk, input[9:0] DrawX, DrawY,
Outputs	output logic is_ball
Description	This module creates a ball and combines the keyboard with the motion. It takes in current pixel coordinates and keypress to determine behavior.
Purpose	It is used to control the movement of the ball in X and Y directions by changing the pixels the ball located, depending on the keyboard inputs. It also controls the way the ball bounces off the wall. Through VGA output, the corresponding output will be displayed on the screen.

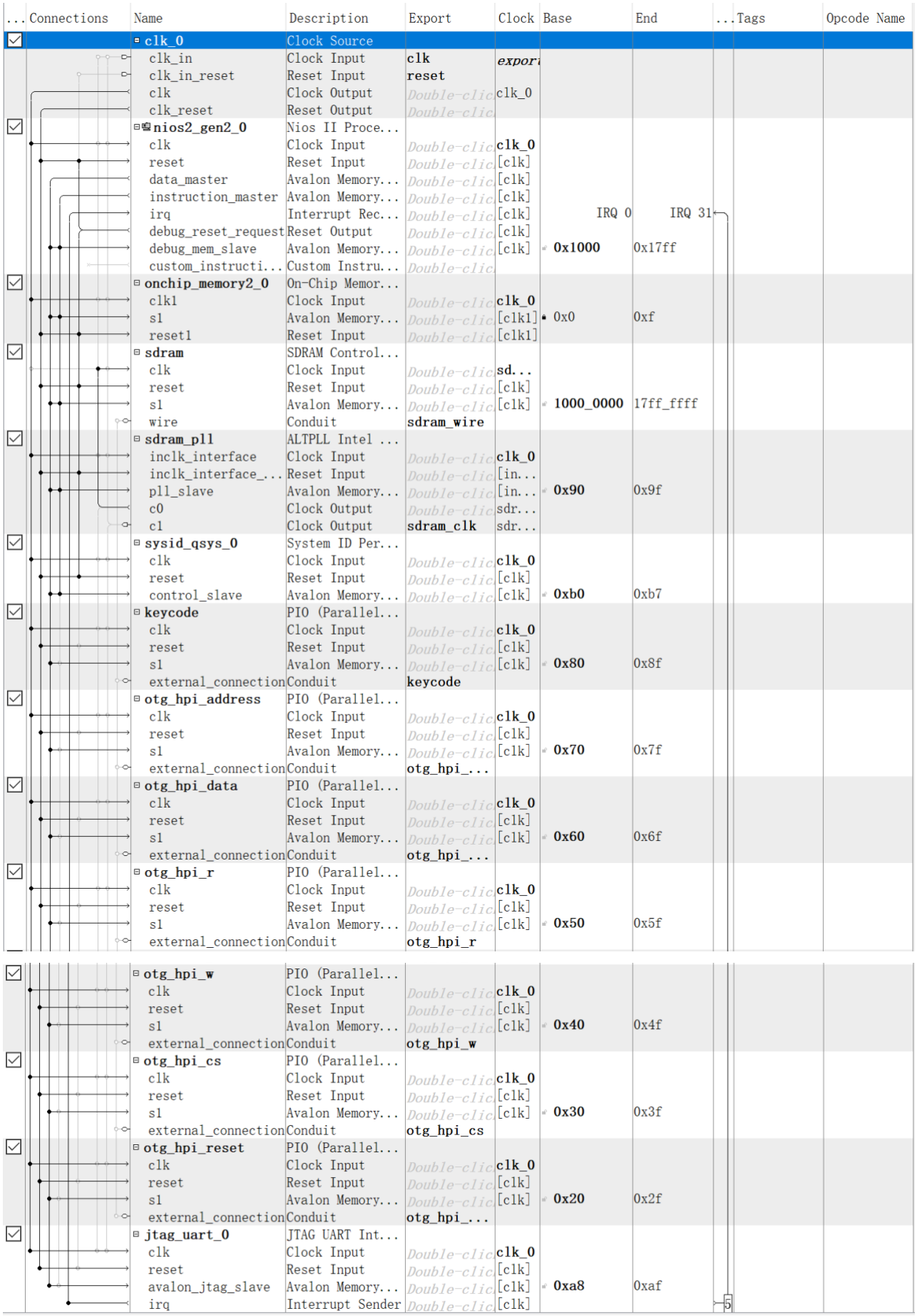
Modules	**color_mapper.sv**
Inputs	input is_ball, input[9:0] DrawX, DrawY,
Outputs	output logic [7:0] VGA_R, VGA_G, VGA_B
Description	This module determines the color that should be displayed on the monitor.
Purpose	It is used to decide which color to be output to VGA for each pixel. The input of this module determines if the current pixel is a ball. The pixel will become white if is_ball, and stay in the background color if! is_ball.

Modules	**HexDriver.sv**
Inputs	input logic [3:0] In0
Outputs	output logic [6:0] Out0
Description	This is a hex driver that transfers a 4-bit signal to represent a hex display on the FPGA. The 4 bits in are needed to have designations for hex values 0 to F.
Purpose	It helps to present the output hex-value on the board.

Modules	**Vga_clk.sv**
Inputs	input inclk0

Outputs	output c0
Description	This module generates the clock that controls the VGA controller.
Purpose	It is used to generate 25MHz clock for the VGA controller to refresh the screen at clock to 60Hz.

Modules	**Lab8_soc.v**
Inputs	input wire clk_clk, input wire [15:0] otg_hpi_data_in_port, input wire reset_reset_n,
Outputs	output wire [7:0] keycode_export, output wire [1:0] otg_hpi_address_export, output wire otg_hpi_cs_export, output wire [15:0] otg_hpi_data_out_port, output wire otg_hpi_r_export, output wire otg_hpi_reset_export, output wire otg_hpi_w_export, output wire sdram_clk_clk, output wire [12:0] sdram_wire_addr, output wire [1:0] sdram_wire_ba, output wire sdram_wire_cas_n, output wire sdram_wire_cke, output wire sdram_wire_cs_n, inout wire [31:0] sdram_wire_dq, output wire [3:0] sdram_wire_dqm, output wire sdram_wire_ras_n, output wire sdram_wire_we_n
Description	This module connects all the hardware together to build the NIOS II system including all PIOs, JTAG UART, memory controller, NIOS II processor.
Purpose	It provides all the hardware components together to communicate with the USB chip and we can program in C with the mapped I/Os.



...	Connections	Name	Description	Export	Clock	Base	End	...	Tags	Opcode Name
☑		clk_0	Clock Source							
		clk_in	Clock Input	clk	export					
		clk_in_reset	Reset Input	reset						
		clk	Clock Output	Double-click	clk_0					
		clk_reset	Reset Output	Double-click						
☑		nios2_gen2_0	Nios II Proce...							
		clk	Clock Input	Double-click	clk_0					
		reset	Reset Input	Double-click	[clk]					
		data_master	Avalon Memory...	Double-click	[clk]					
		instruction_master	Avalon Memory...	Double-click	[clk]					
		irq	Interrupt Rec...	Double-click	[clk]	IRQ 0	IRQ 31			
		debug_reset_request	Reset Output	Double-click	[clk]					
		debug_mem_slave	Avalon Memory...	Double-click	[clk]	0x1000	0x17ff			
		custom_instructi...	Custom Instru...	Double-click						
☑		onchip_memory2_0	On-Chip Memor...							
		clk1	Clock Input	Double-click	clk_0					
		s1	Avalon Memory...	Double-click	[clk1]	0x0	0xf			
		reset1	Reset Input	Double-click	[clk1]					
☑		sdram	SDRAM Control...							
		clk	Clock Input	Double-click	sd...					
		reset	Reset Input	Double-click	[clk]					
		s1	Avalon Memory...	Double-click	[clk]	1000_0000	17ff_ffff			
		wire	Conduit	sdram_wire						
☑		sdram_pll	ALTPLL Intel ...							
		inclk_interface	Clock Input	Double-click	clk_0					
		inclk_interface_...	Reset Input	Double-click	[in...					
		pll_slave	Avalon Memory...	Double-click	[in...	0x90	0x9f			
		c0	Clock Output	Double-click	sdr...					
		c1	Clock Output	sdram_clk	sdr...					
☑		sysid_qsys_0	System ID Per...							
		clk	Clock Input	Double-click	clk_0					
		reset	Reset Input	Double-click	[clk]					
		control_slave	Avalon Memory...	Double-click	[clk]	0xb0	0xb7			
☑		keycode	PIO (Parallel...							
		clk	Clock Input	Double-click	clk_0					
		reset	Reset Input	Double-click	[clk]					
		s1	Avalon Memory...	Double-click	[clk]	0x80	0x8f			
		external_connection	Conduit	keycode						
☑		otg_hpi_address	PIO (Parallel...							
		clk	Clock Input	Double-click	clk_0					
		reset	Reset Input	Double-click	[clk]					
		s1	Avalon Memory...	Double-click	[clk]	0x70	0x7f			
		external_connection	Conduit	otg_hpi_...						
☑		otg_hpi_data	PIO (Parallel...							
		clk	Clock Input	Double-click	clk_0					
		reset	Reset Input	Double-click	[clk]					
		s1	Avalon Memory...	Double-click	[clk]	0x60	0x6f			
		external_connection	Conduit	otg_hpi_...						
☑		otg_hpi_r	PIO (Parallel...							
		clk	Clock Input	Double-click	clk_0					
		reset	Reset Input	Double-click	[clk]					
		s1	Avalon Memory...	Double-click	[clk]	0x50	0x5f			
		external_connection	Conduit	otg_hpi_r						
☑		otg_hpi_w	PIO (Parallel...							
		clk	Clock Input	Double-click	clk_0					
		reset	Reset Input	Double-click	[clk]					
		s1	Avalon Memory...	Double-click	[clk]	0x40	0x4f			
		external_connection	Conduit	otg_hpi_w						
☑		otg_hpi_cs	PIO (Parallel...							
		clk	Clock Input	Double-click	clk_0					
		reset	Reset Input	Double-click	[clk]					
		s1	Avalon Memory...	Double-click	[clk]	0x30	0x3f			
		external_connection	Conduit	otg_hpi_cs						
☑		otg_hpi_reset	PIO (Parallel...							
		clk	Clock Input	Double-click	clk_0					
		reset	Reset Input	Double-click	[clk]					
		s1	Avalon Memory...	Double-click	[clk]	0x20	0x2f			
		external_connection	Conduit	otg_hpi_...						
☑		jtag_uart_0	JTAG UART Int...							
		clk	Clock Input	Double-click	clk_0					
		reset	Reset Input	Double-click	[clk]					
		avalon_jtag_slave	Avalon Memory...	Double-click	[clk]	0xa8	0xaf			
		irq	Interrupt Sender	Double-click	[clk]			5		

QSYs Modules

Clk_0: It is the functional clock for the entire system and can be served as a reference for the other modules.

nios2_gen2_0 Block: It handles the conversion of C# to SV which could be executed by the FPGA board.

Onchip_memory2_0: It is the memory block and a compliment to the nios2 processor which could get us available hardware memory for variables needed to execute the C code onto the board.

sdram: It allows us to get available SDRAM on the FPGA board which can be processed and updated quickly. We need SDRAM because the on-chip memory is too small for the program to be stored and operated successfully.

sdram_pll: It is the separate clock of the system which accounts for the small delays within the transfer of the SDRAM's data in and data out.

sysid_qsys_0: It is used for verification of the correctness of the format transmission between the software code C# and hardware code SV.

buttons_pio: It is the PIO which allows for bidirectional data transfer from the FPGA to software and the inputs are the press of buttons on the FPGA.

jtag_uart_0: It is used for debugging the software as it allows for terminal access. keycode: Reads data, updates the USB chip and then sends it to the software for logic instruction.

otg_hpi_address: It is the PIO that makes the needed address that is sent by the software to the FPGA and stored in memory of the SOC available.

otg_hpi_data: It is the 16-bits PIO that sizable data transfer between the two and makes the cross transfer of data from the FPGA to software possible.

otg_hpi_r: It is the PIO that makes the enable bit to read from the SOC memory which is sent from the software to the FPGA.

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

otg_hpi_cs: It is the PIO that makes the enable bit to turn on and off the memory of the SOC that is sent

from the software to the FPGA.

otg_hpi_reset: It is the PIO that makes the reset of the SOC memory sent from the software to the FPGA possible.

3. Answers to both hidden questions

i. *What are the advantages and/or disadvantages of using a USB interface over PS/2 interface to connect to the keyboard? List any two. Give an answer in your Post-Lab.*

Disadvantages:

a. PS/2 has better compatibility. It can match more devices.

b. PS/2 has lower latency. It has a shorter delay compared with USB.

Advantages:

a. USB is easy to be installed or uninstalled. We just need to plug or unplug it. Yet the PS/2 keyboard needs to be plugged in when booting up the system.

b. USB has higher bandwidth utilization; it can support both synchronous and asynchronous transmission.

ii. *Notice that Ball_Y_Pos is updated using Ball_Y_Motion.*

a. *Will the new value of Ball_Y_Motion be used when Ball_Y_Pos is updated, or the old?*

b. *What is the difference between writing "Ball_Y_Pos_in = Ball_Y_Pos + Ball_Y_Motion;" and "Ball_Y_Pos_in = Ball_Y_Pos + Ball_Y_Motion_in;"?*

c. *How will this impact behavior of the ball during a bounce,*

d. *how might that interact with a response to a keypress? Give an answer in your Post-Lab.*

a. The old value of Ball_Y_Motion be used when Ball_Y_Pos is updated. 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.

b. The difference is that the first one means that the Y position of the next clock cycle is determined by current Y position and current motion in the Y direction, which will not be influenced by the current pressed key or whether the ball is at the boundary. Whereas the second one means that Y position of the next clock cycle is determined by current Y position, current key-press action and the boundary status.

c. During the bounce, the ball will continue the original direction for one more cycle and go outside of the boundary.

d. With a response to a keypress, this may cause delay with response to a key press or a bounce.

4. Answer to Post Lab Questions

i. *What is the difference between VGA_clk and Clk?*

The VGA_clk runs at 25MHz frequency to ensure VGA port and the monitor operate at 60Hz while the Clk runs at 50MHz frequency to run other programs.

ii. *In the file io_handler.h, why is it that the otg_hpi_data is defined as an integer pointer while the otg_hpi_r is defined as a char pointer?*

otg_hpi_data is defined as an integer pointer as it represents an inout variable, which handles data above one byte. Thus we use an integer pointer which requires 4 bytes.

otg_hpi_r is defined as a char pointer as it handles the reading of one character, which points to a register that holds the value of the key pressed and requires only one byte. Thus, we have no need for integers of 4 bytes, the 1-byte char is enough, and it could save memory.

iii. *Document the Design Resources and Statistics in following table.*

LUT	2688
DSP	10
Memory (BRAM)	11392
Flip-Flop	2235
Frequency	115.15MHz
Static Power	105.15mW
Dynamic Power	0.74mW
Total Power	174.83mW

5. Conclusion

i. *Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it*

Our design worked well, the ball could move and bounce in either the X or Y direction on the monitor screen as required and we learned how to interface with HID and HPI protocol. There is nothing ambiguous, incorrect, or unnecessarily difficult in the lab manual.