ECE385

Fall 2020

Experiment 8

SOC with USB and VGA Interface in SystemVerlog

Name: Zhou Qinren
Zhang Yichi
Lab Section: LA3
TA's Name: Yu Yuqi

Nov. 24th 2020

1 Introduction

In this lab, we will build an interface to connect a keyboard with universal serial bus (USB), a monitor with VGA, the NIOS-II processor and our DE2-115 board. With this interface, we create a game which we control a ball to move around with ”wasd” on the keyboard.

The data collected by keypresses will be stored in the Cypress EZ-OTG (CY7C67200) USB controller, then be transferred to the FPGA through HPI controlled by NIOS-II. After the keycodes are converted to the motion of the ball, it updates the ball’s status. Meanwhile, the VGA is constantly refreshing the monitor display pixel by pixel based on its own clock to show the ball moving around.

2 Written Description

2.1 Entire System

Lab 8 system consists of a monitor, a keyboard and a DE2 board. The monitor is connected to the board through VGA port, and the keyboard through the USB port. The NIOS is similar to the one in Lab 7. It serves as the processor, which perform some operations and read instructions. We use on-chip memory and SDRAM as memory storage. The sdram_pll is in charge of dealing with sdram. Jtag_uart module allows us to use the terminal of the host computer to communicate with the NOIS using print and scan statements in C. We also added some other PIOs to specify the ways EZ-OTG and FPGA interact.

The keyboard is connected through a USB port. The EZ-OTG handles the USB protocol, it will fetch the keycode we input. We also write some instructions in C code, which send some signals to HPI, which is responsible of transferring data between EZ-OTG and FPGA.

The VGA controller is used to display on the monitor. We have a color_mapper to paint the color of the screen, and a ball module to indicate whether the current pixel is within the ball.

2.2 How the NIOS Interacts with Both the CY7C67200 USB Chip and the VGA Components

The NIOS handles the USB chip with its software. We added some PIOs to output some signals which is in charge of HPI interface. Then we write some C code to output the signals. These signals will be connected to the USB chip, and HPI_ADDR indicates the address we want to read or write. HPI_DATA is the register which holds the data we want.

The NIOS-II does not interact with VGA directly. VGA operations are written in SV code and is handled by FPGA itself.

2.3 CY7 to Host Protocol (HPI)

HPI is a built-in module on the CY7. We can use HPI to fetch or write data. We mainly deal with HPI Data and HPI Address registers. If we want to write data, we first set the Chip Select and Write to active, then write to the address register the desired address, then we write the data to the Data register. When we want to read, we set CS and Read to active, then write the desired address to the Address register, and then we read the data from the Data register.

2.4 USBRead, USBWrite, IO_read and IO_Write

IO_write and IO_read write to or read from a specific register with a given address. USBWrite and USBRead make use of the two IO functions. USBWrite writes the address to the HPI ADDR, and the data to HPI Data. USBRead writes the address to HPI ADDR, and read data from HPI Data. To sum up, USBRead and USBWrite are the IO interface between the Nios and the USB chip, IO_read and IO_write are two helper functions.

3 Block Diagram

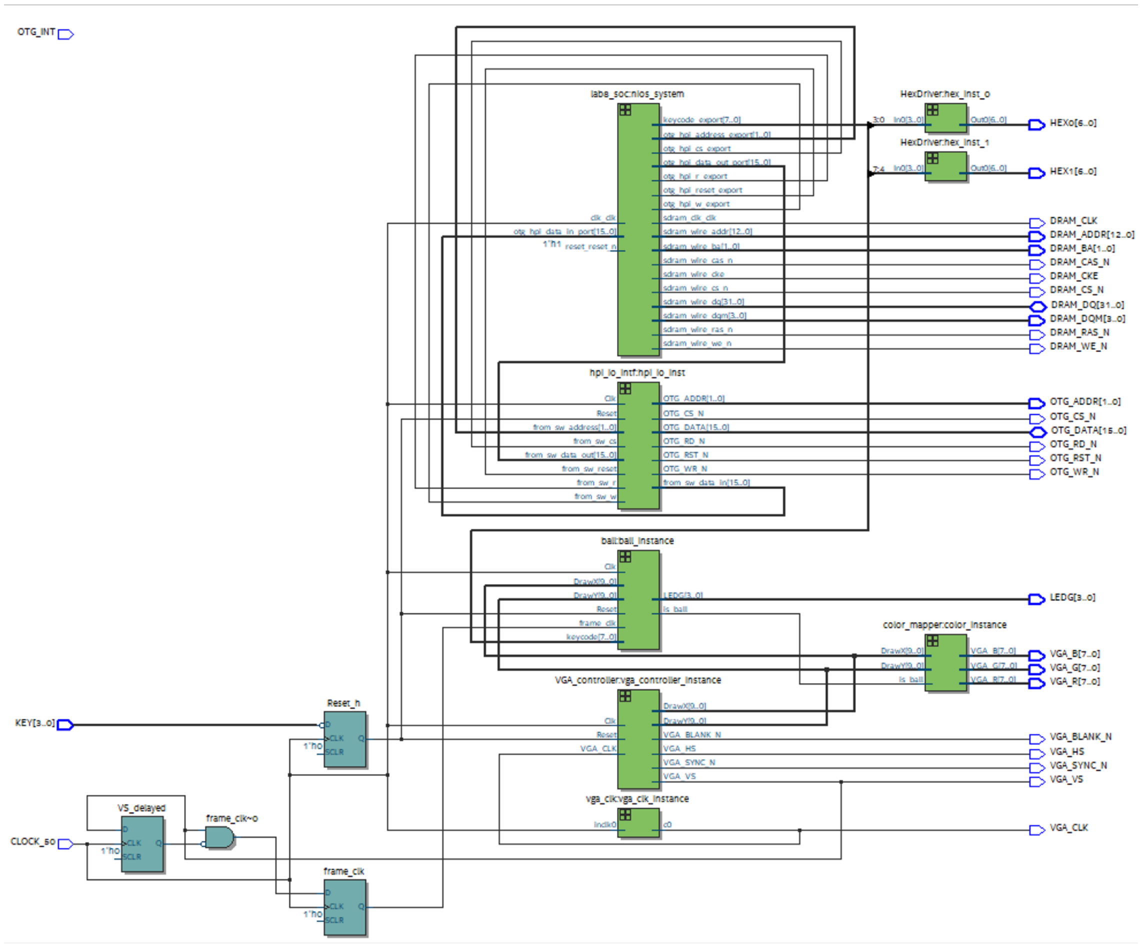


Figure 1: Top level block diagram.

4 Module Description

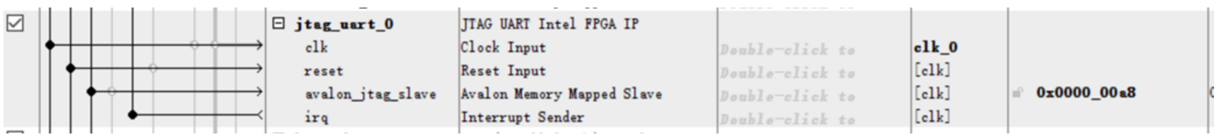
Module: lab8.sv
Inputs: CLOCK_50, KEY, OTG_INT
Outputs: HEX0, HEX1, VGA_R, VGA_G, VGA_B, VGA_CLK, VGA_SYNC_N, VGA_BLANK_N, VGA_VS, VGA_HS, [1:0] OTG_ADDR, OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N, OTG_INT, [12:0] DRAM_ADDR, [1:0] DRAM_BA, [3:0] DRAM_DQM, DRAM_RAS_N, DRAM_CAS_N DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK, [3:0] LEDG
Inouts: [15:0] OTG_DATA, [31:0] DRAM_DQ
Description: This is the top-level module.
Purpose: This module instantiates the low-level modules and connect them properly.

Module: VGA_controller.sv
Inputs: Clk, Reset, VGA_CLK
Outputs: VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, [9:0] DrawX, [9:0] DrawY
Description: This module controls the VGA signals (horizontal and vertical).
Purpose: This module keeps tracks of the pixel that is currently under test, so that we can draw that pixel correctly.

Module: hpi_io_intf.sv
Inputs: Clk, Reset, [1:0] from_sw_address, [15:0] from_sw_data_out, from_sw_r, from_sw_w, from_sw_cs, from_sw_reset
Outputs: [15:0] from_sw_data_in, [1:0] OTG_ADDR, OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N
Inputs: [15:0] OTG_DATA
Description: This module sends some control signals to the OTG chip.
Purpose: This module is the hpi interface that communicates with OTG chip, it sends some control signals to the chip. These signals indicate whether to perform a read or write operation, and the address is specified.

Module: Color_Mapper.sv
Inputs: is_ball, [9:0] DrawX, [9:0] DrawY
Outputs: [7:0] VGA_R, [7:0] VGA_G, VGA_B
Description: This module determined whether the current pixel should display color.
Purpose: This module is used to draw the ball, if the current pixel is part of the ball, then it will be colored, and if not, it will remain the background color.

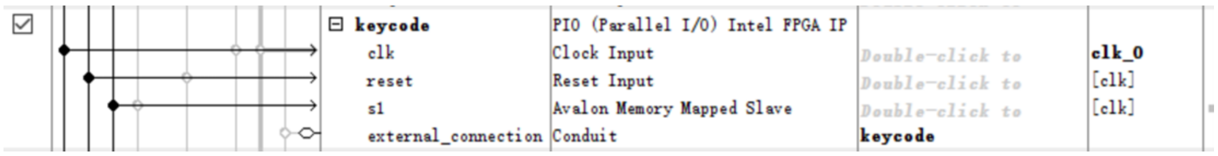
Module: ball.sv
Inputs: Clk, Reset, frame_clk, [9:0] DrawX, [9:0] DrawY, [7:0] keycode
Outputs: is_ball, [3:0] LEDG
Description: This module determines whether the current pixel is part of the ball, and it is also in charge of changing the motion direction according to the given keycode.
Purpose: This module indicates whether the current pixel is part of the ball. It also memorizes the motion direction of the ball, and it will change direction according to the keycode. Moreover, it handles the case when the ball is bouncing back.



⊟ jtag_uart_0	JTAG UART Intel FPGA IP				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_00a8	
irq	Interrupt Sender	Double-click to	[clk]		

Figure 2: JTAG_UART.

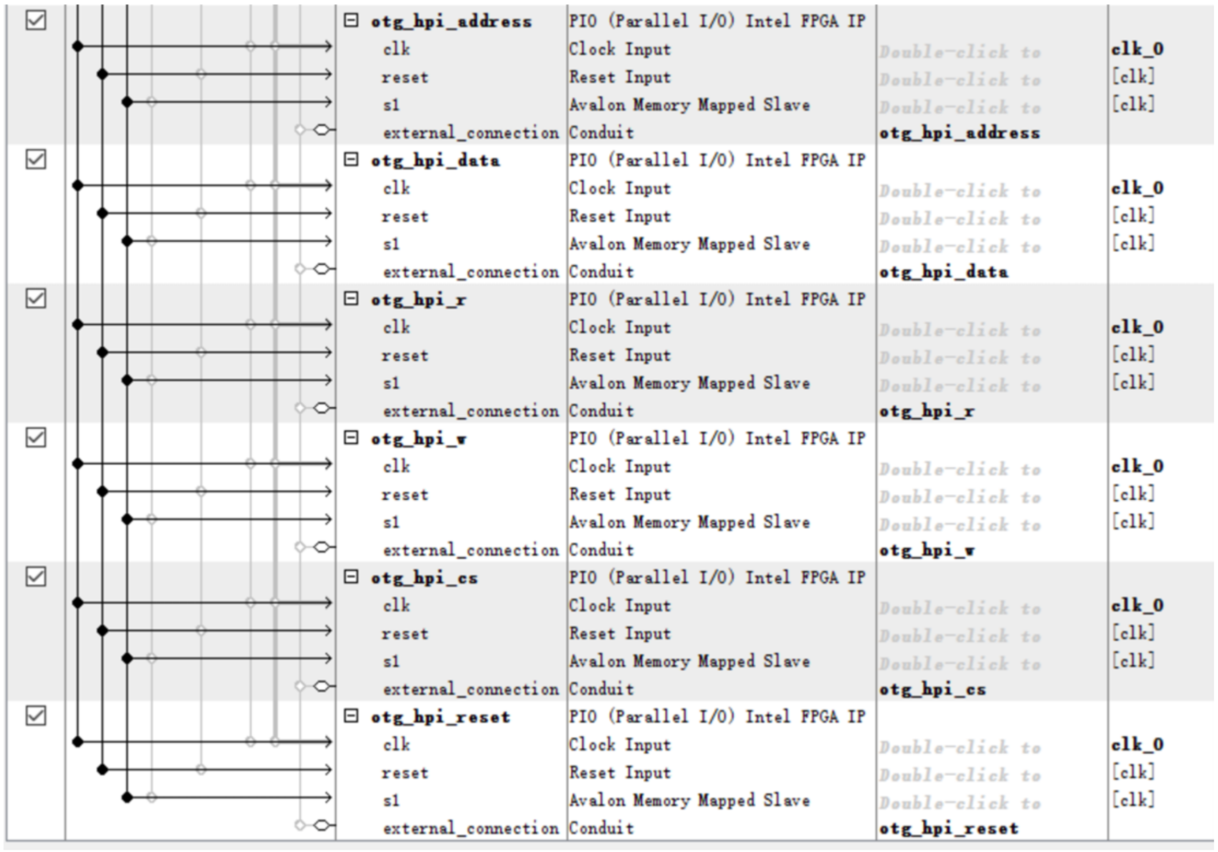
This module allows us to use the terminal of the host computer to communicate with the NOIS using print and scan statements in C.



⊟ keycode	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	Double-click to	clk_0
reset	Reset Input	Double-click to	[clk]
s1	Avalon Memory Mapped Slave	Double-click to	[clk]
external_connection	Conduit	keycode	

Figure 3: Keycode.

This module outputs the keycode from USBRead.



⊟ otg_hpi_address	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	Double-click to	clk_0
reset	Reset Input	Double-click to	[clk]
s1	Avalon Memory Mapped Slave	Double-click to	[clk]
external_connection	Conduit	otg_hpi_address	
⊟ otg_hpi_data	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	Double-click to	clk_0
reset	Reset Input	Double-click to	[clk]
s1	Avalon Memory Mapped Slave	Double-click to	[clk]
external_connection	Conduit	otg_hpi_data	
⊟ otg_hpi_r	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	Double-click to	clk_0
reset	Reset Input	Double-click to	[clk]
s1	Avalon Memory Mapped Slave	Double-click to	[clk]
external_connection	Conduit	otg_hpi_r	
⊟ otg_hpi_w	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	Double-click to	clk_0
reset	Reset Input	Double-click to	[clk]
s1	Avalon Memory Mapped Slave	Double-click to	[clk]
external_connection	Conduit	otg_hpi_w	
⊟ otg_hpi_cs	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	Double-click to	clk_0
reset	Reset Input	Double-click to	[clk]
s1	Avalon Memory Mapped Slave	Double-click to	[clk]
external_connection	Conduit	otg_hpi_cs	
⊟ otg_hpi_reset	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	Double-click to	clk_0
reset	Reset Input	Double-click to	[clk]
s1	Avalon Memory Mapped Slave	Double-click to	[clk]
external_connection	Conduit	otg_hpi_reset	

Figure 4: Otg_hpi interface.

This module is the otg_hpi interface signals. All of them except hpi_data are output signals from Nios. Hpi_data is inout signal which transfer data between Nios and USB chip. Hpi_address is used to specify the desired hpi register. Hpi_r indicates read operation, hpi_w indicates write operation and hpi_cs indicates chip select. Hpi_reset is used to reset.

5 Answers to Both Hidden Questions

Question: What are the advantages and/or disadvantages of using a USB interface over PS/2 interface to connect to the keyboard? List any two.

Answer: The USB interface only accepts 6 keypresses at once while the PS/2 interface has no such limit. For USB devices to send information to the host device, they wait for the host to continuously request polls. While the PS/2 devices send interrupt signals to pause the CPU and send information.

Question: Notice that Ball_Y_Pos is updated using Ball_Y_Motion. Will the new value of Ball_Y_Motion be used when Ball_Y_Pos is updated, or the old? 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;"? How will this impact behavior of the ball during a bounce, and how might that interact with a response to a keypress?

Answer: Old motion is used. "Ball_Y_Pos_in = Ball_Y_Pos + Ball_Y_Motion;" updates the position with the old motion, while "Ball_Y_Pos_in = Ball_Y_Pos + Ball_Y_Motion_in;" updates the position with the new motion. If we use the latter one, the ball will change the direction as soon as it hits the boundary or takes in a keypress but if we use the former one, the ball will go along the direction before until the next period (rising edge of the clock signal).

6 Answer to Post Lab Questions

Question: What is the difference between VGA_clk and Clk?

Answer: Clk is 50 MHz clock and VGA_clk is 25 MHz clock. Clk is the clock for the NIOS-II processor and its PIOs, while the VGA_clk is for the screen refreshing pixel by pixel (1/25MHz=4ns to update one pixel).

Question: 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?

Answer: The otg_hpi_r stores the read enable signal. There is only 1 or 0 so char (1 byte) is enough. The otg_hpi_data stores the data we read or write so int type will be better.

LUT	2696
DSP	10
BRAM	55296
Flip-Flop	2234
Frequency	100.77MHz
Static Power	105.16mW
Dynamic Power	1.07mW
Total Power	177.37mW

Table 1: Design statistics table for the multiplier.

7 Conclusion

7.1 Functionality

Both the hardware and software worked fine. The ball can be controlled by our instructions through the keyboard. However, when we went to the laboratory and used the keyboard there, we encountered some weird warnings and the NIOS-II stopped taking in the keycode after we pressed the KEY[0] to reset. Later, we changed a keyboard and everything worked fine as before. We considered it to be the keyboard hardware problem but we do not understand the specific reason.

7.2 About the Lab Manual

We hope to get more information about the connection and how signals transfer between EZ-OTG, NIOS-II, HPI, CY7 and FPGA.