ECE 385

Spring 2019

Experiment #8

SOC with USB and VGA Interface in SystemVerilog

Rob Audino and Soham Karanjikar

Section ABJ, Friday 2:00-4:50

TA: David Zhang

Introduction:

In lab 8, we were required to utilize a combination of System Verilog and C to create an interface between a USB keyboard and a computer monitor through the FPGA. In doing so, we could create a game of sorts where we use the “wasd” keys on the USB keyboard in order to control the motion of a bouncing ball on the computer monitor screen. When a key was pressed, the corresponding ASCII value of the key would be displayed on the FPGA.

USB (Universal Serial Bus) is an industry standard which establishes specs for cables and connectors for communication, connection, and power supply between PC’s and their corresponding peripheral devices. Since there is a USB input on our FPGA board, the FPGA board also has a corresponding EZ-OTG USB Controller to handle data transmission through the port. In addition, the NIOS II chip on the board also has software to handle data from a USB input. NIOS itself can extract the keycode from the USB input to see what exactly the user wanted to convey through each press of a key. VGA operates as a matrix of pixels, which are updated by a beam of light that travels left to right, and then top to bottom. By tracking the beam of light, we can know the contents of each pixel.

Written Description of Lab 8 System:

<u>Written Description of the entire Lab 8 system:</u>

❏ Describe in words how the NIOS interacts with both the CY7C67200 USB chip and the VGA components

The NIOS chip needs a JTAG interface to communicate through the USB port available on the DE2 board. This is done through the memory-mapped parallel input outputs (PIO blocks). The software for this interface is written in C, with functions USB_READ and USB_WRITE. These two functions read/write to the internal registers in the CY7C67200 USB chip. Through this we can access keystrokes to recognize W A S D and map the appropriate movement of the ball. Temporary variables hpi_addr and hpi_data are used from transmitting data between lab8 SOC and OTG chip in top level.

To interact with the VGA there are 3 modules: VGA controller, ball and color mapper. VGA controller sends data of which pixels are currently in use to ball. Ball uses color mapper and its own logic for moving the ball on screen and controlling the bouncing off edges and reading from the USB keyboard and mapping the movement. After this calculation is done, color mapper actually writes the outputs to the monitor with appropriate RGB values. Depending on what row and column is being drawn, appropriate vertical and horizontal signals are sent to signal the VGA monitor to draw the next frame and row respectively. The pin assignments are done in top level so that the hardware connections are connected in top level outputs.

Written description of the USB protocol:

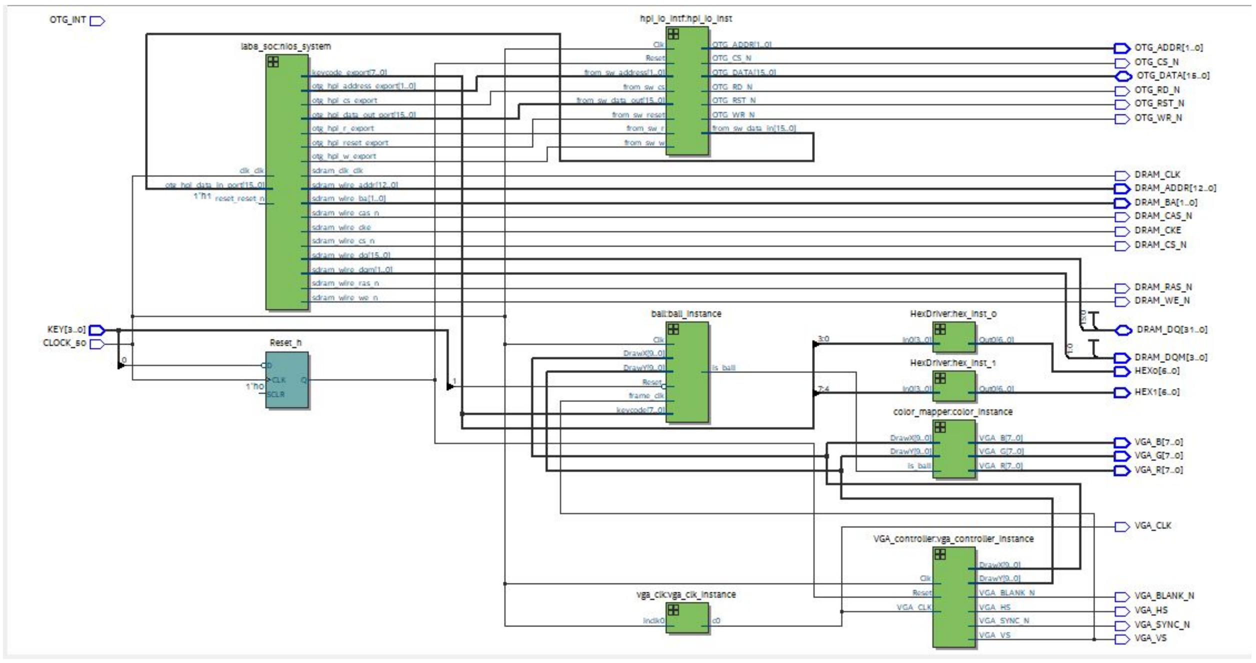
USBRead: The USBRead function uses C to read data from the internal registers of the Cypress USB controller.

USBWrite: The USBWrite function uses C to write data to the internal registers of the Cypress USB controller.

IO_read: IO_read is used in the Reading function, and manages timing so that the correct values are read from Cypress into the FPGA

IO_write: IO_write is used in both the Reading and Writing functions, and manages timing so that the correct values are read and written from cypress to the FPGA.

Block diagram



Module descriptions

Module: tristate
Inputs: Clk, tristate_output_enable, [15:0] Data_write,
Outputs: [15:0] Data_read
Description: This component takes an input and a signal that allows the output to be HI-Z or the input.
Purpose: This allows us to not have a 0 or 1 but rather a NOINPUT. This means that we can have inputs to the bus that don't matter for the current state.

Module: VGA_controller
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 operation of the Monitor
Purpose: This module allows us to display our bouncing ball on the monitor.

Module: ball
Inputs: Clk, Reset, frame_clk, [9:0] DrawX, [9:0] DrawY, [7:0] keycode
Outputs: is_ball
Description: This module describes the position and movement of the ball
Purpose: This module allows our ball to look, move, and react properly.

Module: color_mapper
Inputs: is_ball, [9:0] DrawX, [9:0] DrawY
Outputs: [7:0] VGA_R, [7:0] VGA_G, [7:0] VGA_B
Description: This module describes how the ball and background look
Purpose: This module makes sure that our ball and background are drawn correctly.

Module: HexDriver
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: This module drives the LED display on our FPGA board.
Purpose: The purpose of this module is to ensure that we can see each keyboard input to our FPGA.

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_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
Description: This module is the top level file of the whole lab.
Purpose: This module connects NIOS to the other blocks and drivers involved in the lab so we can run C on the FPGA.

Module: hpf_io_intf
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
Description: This module is the interface between NIOS II and the EZ-OTG chip on the FPGA board.

Purpose: This module implements the transfer of USB data.

PIO Blocks:
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).

JTAG_UART: This is used so that we can use the terminal of the host computer to communicate with the NIOS II.

keycode: This is used so that we can see which key on the keyboard is being pressed.

otg_hpi_address: This is address to write/read to from the EZ-OTG chip.

Otg_hpi_data: This is the data written/read from the registers in EZ-OTG chip, from the specified address.

Otg_hpi_r: This is the read enable signal to control the read function from the EZ-OTG chip. To read keystrokes from keyboard.

Otg_hpi_w: This is the write enable signal to control the write function from the EZ-OTG chip.

Otg_hpi_cs: This is the chip select to enable reading or writing. This has to be used in parallel with read/write signals.

Otg_hpi_reset: This is the reset signals to clear the registers in the EZ-OTG chip.

All of these signals are decided to control in C code that and used through the memory-mapped PIO blocks to interface with hardware.

Answers to both hidden questions:

1.Q: 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.
A: Since PS/2 connects more closely to the processor than the USB, it can transfer data at a much faster rate than the USB. In addition, PS/2 is cheaper and easier to implement than USB. However, a USB keyboard is hot swappable, meaning that it can be plugged in while the computer is still running and the computer will immediately accept input, whereas a computer would need to be restarted in order to take a PS/2 input. In addition, the PS/2 keyboard doesn't need to be polled, and instead sends a signal to the computer, causing a hardware interrupt. Also, the USB keyboard can only pass 6 keystrokes at a single time, whereas the PS/2 can theoretically handle any number of simultaneous key presses.

2.Q: 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?
A: Since parallel assignments are used in these instances, the old value of Ball_Y_Motion will end up being used when Ball_Y_Pos is updated. If the ball was bouncing during this update, the ball would end up moving through the wall instead of bouncing off of the wall in the opposite direction. If a key was pressed, the ball would still end up having the incorrect motion for a single frame before correcting to the motion intended by the pressed key.

Answer to Post Lab Questions:

1. Q: What is the difference between VGA_clk and Clk?
A: Clk is a 50 MHz clock used to drive the FPGA itself, whereas VGA_clk is a 25 MHz clock that is used to update the frames on the computer monitor.

2. Q: 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?
A: Since otg_hpi_r is simply an identifying signal that is used to show when the OTG interface is ready to read, it can be defined as a char pointer. However, otg_hpi_data is defined as an integer pointer since it must convey actual data that has to be looked up in memory.

3. Q: Document the Design Resources and Statistics in following table.
A:

LUT’s	2712
DSP	10
Memory (BRAM)	11,392 bits
Flip Flop	2011
Frequency	125.34 MHz
Static Power	105.14 mW
Dynamic Power	0.76 mW
Total Power	173.44 mW

Conclusion:

After some issues concerning a system ID error, we finally got our project to work in the lab. The reason we encountered this error was because we had completed the lab on one of the computers in 3022, and this was our first time using this code on our laptop computer to demo in class. Thankfully, after restarting Quartus, NIOS, and the FPGA, and re-compiling and running our code, we were finally able to get the bouncing ball working. We had to run our NIOS program several times to work, because some of the times we ran it resulted in the bouncing ball program being launched without our USB keyboard working to control it or the keys pressed displaying on the FPGA.

Previous to this in class error, we had encountered several problems with our C code and bouncing algorithm and boundary conditions. Firstly, we had a typo in one of our edge cases, so our ball went through the right edge of the screen at first. We recognized this fairly quickly, but another error plagued us for quite a bit longer than this one. Our ball would travel only vertically or horizontally almost all of the time; however, we were able to get the ball to bounce diagonally when we alternated pressing the up and right keys towards the upper right corner, and for some reason the bouncing caused it to retain both its horizontal and vertical movement, and the ball would start to move down and to the left. When we coded the movements of the ball, we made it so that any press of a vertical key (up or down) would kill any previous horizontal velocity, and any press of a horizontal key (left or right) would kill any previous vertical velocity. However, as an extra precaution, we also added this velocity-killing feature to the bounces off of the walls. After making this small change, our diagonal movement ceased to appear no matter how hard we tried to make it appear. We ended up being very fortunate to have forced and solved this error before class, as our TA David ended up forcing it on quite a few groups in class who were unprepared or didn't have enough time to deal with this error during class.

In summary, we enjoyed this lab in a different way than the other labs since it drew sort of a connection to the video games we had played as children. We were able to see how we could integrate C code, SV and the FPGA, and a USB keyboard input into an application that we hadn't explored before. This lab was also important because it allowed us a means by which to create our final project, most likely some sort of video game. We were able to implement the ball mechanics in C, which was more familiar to us than SV, and we also ended up playing with the colors of the ball and the background once we finished the guts of the lab. For future semesters, I don't think much needs to be changed. The tutorial explained the Platform Designer quite well, and the ball mechanics are fairly easy to figure out. The only thing I would think to mention in the lab description to look for would be the aforementioned diagonal glitch near the corners.