ECE 385

Experiment #8

Spring 2018

USB/VGA

Sahil Shah/sahils2
Samir Kumar/samirkk2
ABC - Tues, 11:30-2:20
Vibhakar Vemulapati

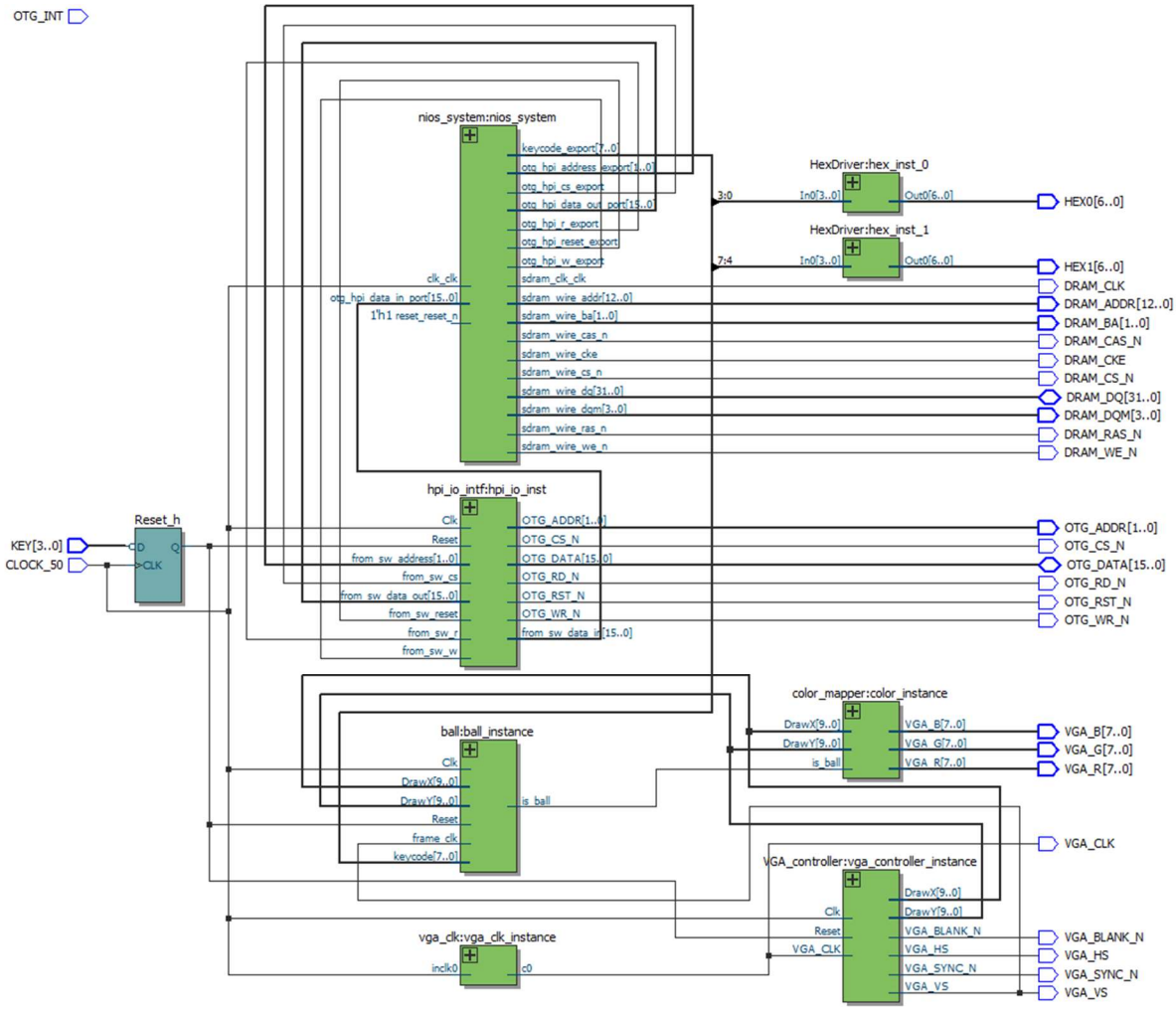
Introduction:

This lab entails designing a USB and a VGA interface that allows a user to control the movement of a ball on the display. First the USB protocol and interface was written. When a key is pressed, its keycode is stored on the EZ-OTG USB Chip on the board, transferred to the hardware using NIOS II, and is then displayed on the hex display of the FPGA using the hex drivers. Once complete, this interface is extended to the VGA interface. When the user presses one of the direction keys on the keyboard, the ball on the VGA display moves in that direction. The ball is bounded on all four sides of the monitor. If a ball hits a boundary, the ball will bounce and move in the opposite direction. This bounce will take priority over a key press. Meaningful keycodes are keypresses of "w" "a" "s" and "d". The keycode is ignored if any other key is pressed, but it is displayed on the hex display regardless.

Written Description of the entire Lab 8 system:

The NIOS system on a chip exports the otg_hpi signals. These signals are then connected to the EZ-OTG chip and are used to determined whether a read or write is to be performed. These signals are set in the C code in each IO function. Specifically, the EZ-OTG polls the keyboard for an input and stores it in RAM. The C code for NIOS then fetches the data (the keycode) via the hpi-io-intf. The keycode will then be sent to the ball.sv module which controls the movement of the ball on the VGA display.

Block Diagram:



Written Description of USB Protocol:

IO_write: Writes data to the HPI data register at the address stored in the HPI address rUess passed to the function. Then, the HPI chip select and HPI write registers are dereferenced and assigned to zero. Now the data HPI data register is assigned the data passed to the function. Lastly, the chip select and write values are set back to one.
wr
IO_read: Reads data from the address specified in the HPI address register. The HPI address register is dereferenced and assigned the target address. Then chip select and read are set to zero and the data in the HPI data register is returned.

Module Description:

USB_write: perform an IO_write to write the address to the address register, then an IO_write to write the data to the data register.

USB_read: perform an IO_write to write the address to the address register, then an IO_read to read the data from the data register.

Module: ball.sv
Input: Clk, Reset, frame_clk,
[7:0] keycode
[9:0] DrawX, DrawY
Output: is_ball
Description: This module tracks of the motion of the ball by determining its direction based on the key pressed and its proximity to a boundary. There are also checks to ensure that diagonal movement is not a possibility. This is done by giving the bouncing off the wall the priority over changing direction.
Purpose: This module is needed to control the movement of the ball. This module also determines whether or not the the VGA controller and the Color mapper should print white for the ball or the color for the background.

Module: Color_mapper.sv
Inputs: is_ball
[9:0] DrawX, DrawY
Outputs: [7:0] VGA_R, VGA_G, VGA_B
Description: This module decides what color is to be mapped to VGA for each pixel
Purpose: Create the background color and the color of the ball on the VGA

Module: HexDriver.sv
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: Creates the interface to the hex displays
Purpose: Allows the program to output data to the hex displays

Module: hpi_io_intf.sv
Inputs: Clk, Reset, from_sw_r, from_sw_w, from_sw_cs, from_sw_reset
[1:0] from_sw_address
[15:0] from_sw_data_out, OTG_DATA
Outputs: OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N
Description: Assign the value of the “On the Go” signals
Purpose: Take the values of the “On the Go” signals and pass them to the on board USB chip. The output is set to high impedance unless the bus is being written to.

Module: lab8.sv
Inputs: CLOCK_50, OTG_INT
[3:0] KEY

[15:0] OTG_DATA
[31:0] DRAM_DQ

Outputs: OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N, DRAM_RAS_N, DRAM_CAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK, VGA_CLK, VGA_SYNC_N, VGA_BLANK_N, VGA_VS, VGA_HS

[1:0] OTG_ADDR, DRAM_BA
[3:0] DRAM_DQM
[6:0] HEX0, HEX1
[7:0] VGA_R, VGA_G, VGA_B
[12:0] DRAM_AtDDR
[15:0] OTG_DATA
[31:0] DRAM_DQ

Description: This module is the top level design. It instantiates all modules.
Purpose: Connects all modules together in hardware such that the final product is the ball and background displayed on a VGA monitor

Module: VGA_controller.sv
Inputs: Clk, Reset, VGA_CLK
Output: VGA_HS, VGA_VS, VGA_BLANK, VGA_SYNC_N
[9:0] DrawX, DrawY
Description: Determines the coordinates to draw
Purpose: Controls how the screen is drawn by outputting the horizontal and vertical coordinates

Answers to hidden questions:

Hidden Question #1/2:

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**.

ANSWER:

- Advantages of USB:
    a. It is “Universal” so that means that there are multiple devices that can have the same connections to a computer. So you do not need multiple different ports like you would if all of your devices had PS/2 ports.
- Disadvantages of USB:
    a. It has a higher latency than PS/2 ports.
    b. PS/2 uses the interrupt method which will ‘interrupt’ the current operation if its priority is higher. This is faster than polling which is what USB uses.
    c. PS/2 allows an unlimited amount of inputs and one time, while USB is limited.

Hidden Question #2/2:

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:
The reason that Ball_Y_Pos is updated using Ball_Y_Motion, and not Ball_Y_Motion_in, is that Ball_Y_Motion is updated with the value of Ball_Y_Motion_in on the rising edge of the CLK. Now Ball_Y_Motion_in is only able to be updated on the rising edge of the frame clock. So Ball_Y_Motion is essentially updated on the rising edge of the frame clock. This means that Ball_Y_Pos is updated on the rising edge of the frame clock, which would be synced with the drawing of the screen. So for each frame, Ball_Y_Pos is updated exactly once.

If you update Ball_Y_Pos using Ball_Y_Motion_in, then there is no clock edge on which Ball_Y_Pos will be updated, since both Ball_Y_Pos and Ball_Y_Motion_in are in an always_comb procedure block. This means that Ball_Y_Pos could potentially update multiple times within one frame clock cycle and even within one CLK cycle.This means that the ball will just skip around the screen.

Design Resources and Statistics:

	Slow 85 C
LUT	2686
DSP	10
Memory (BRAM)	55,296
Flip-Flop	2,276
Frequency MHz	155.52
Static Power (mW)	105.16
Dynamic Power (mW)	1.13
Total Power (mW)	175.24

What is the difference between VGA_clk and Clk?

- The VGA clock is the clock that runs the VGA display. VGA requires a 25 Mhz clock speed as opposed to the 50 Mhz that controls the rest of the hardware.

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_r simply takes the value 0 or 1 which can be represented with the 1-byte char data type. Otg_hpi_data, however, requires multiple bytes to represent so the integer data type is needed here.

Conclusion:

Our design worked as intended. We struggled with the hpi interface, as the connections of the EZ-OTG with NIOS with the hardware was difficult to understand. Initially, we encountered the error where the ball would travel through the boundary. This occurred because keypresses had higher priority than the boundary. This was fixed by making the boundary operation a higher priority than the keypress operation of the user. Our next error happened when the ball travelled diagonally off of a bounce due to the user pressing a direction in the horizontal or vertical direction and the ball bouncing in the other. This was fixed by clearing the direction in the direction that the ball did not bounce in once it hit a boundary.