ECE 385

Spring 2020

Experiment # 8

SOC with USB and VGA Interface in SystemVerilog

Megh Shah, Ishaan Patel
Lab Section: ABC, Tuesday 11:30 AM
TA: Gene Shiue, David Zhang

Introduction:

The NIOS II is a 32-bit CPU which supports many different I/O peripherals. In this lab, we utilized a keyboard and a monitor as forms of inputs and outputs by implementing a USB and VGA interface. This USB keyboard is a Human Interface Device (HID) which sends information in the form of key presses when the host requests it. The VGA interface operates differently as it is synced to its own clock which it utilizes to update the pixels on the monitor. Overall, by utilizing the NIOS II CPU, a USB keyboard, and a VGA monitor, we were able to connect everything together to create a seamless interaction to ultimately be able to control and display a moving ball.

Written Description of Lab 8 System:

a. **Written Description of the entire Lab 8 system** –
   The Lab 8 system mainly focuses and operates on the ideas of having I/O peripherals. Regarding the input, we utilized the USB port on the FPGA board. The USB port takes in data from the keyboard and sends it to the host controller, which is the CY7C67200 chip on the FPGA board, when the host requests information. Afterwards, the host controller sends the information through the HPI interface which holds the data until the next clock cycle is available when it then sends it to the NIOS II CPU. Regarding the output, we utilized the VGA connection on the FPGA board. The VGA connection has 9 individual connections which we use. Some of these connections are directed straight to the FPGA board while others pass through a VGA DAC converter beforehand. Ultimately, all signals end up at the VGA output in the FPGA board which then sends the signal to the monitor. The VGA DAC, which runs off the 25Mhz clock, uses this data to update the pixels on the screen in the next available clock cycle. For this lab, our keypresses, which control the direction of the moving ball, go through the several software and hardware interfaces before reaching the CPU which updates the pixel location of the ball, and then sends the updated pixel locations through the VGA connection for the monitor to display.

b. **Describe in words how the NIOS interacts with both the CY7C67200 USB chip and the VGA components** –
   The NIOS II CPU interacts with the CY7C67200 USB chip through the host protocol interface (HPI). The USB chip acts as a host controller which constantly polls information from the USB port. Whenever information is requested and received by the chip, it sends it through the HPI module that we created and into the NIOS II CPU where the data is ready to be read and manipulated if needed. The NIOS II CPU interacts with the VGA component differently. The only thing that the CPU does is send a keycode to a ball module that is responsible for the ball's location on the monitor. In terms of general interfacing, the color mapper

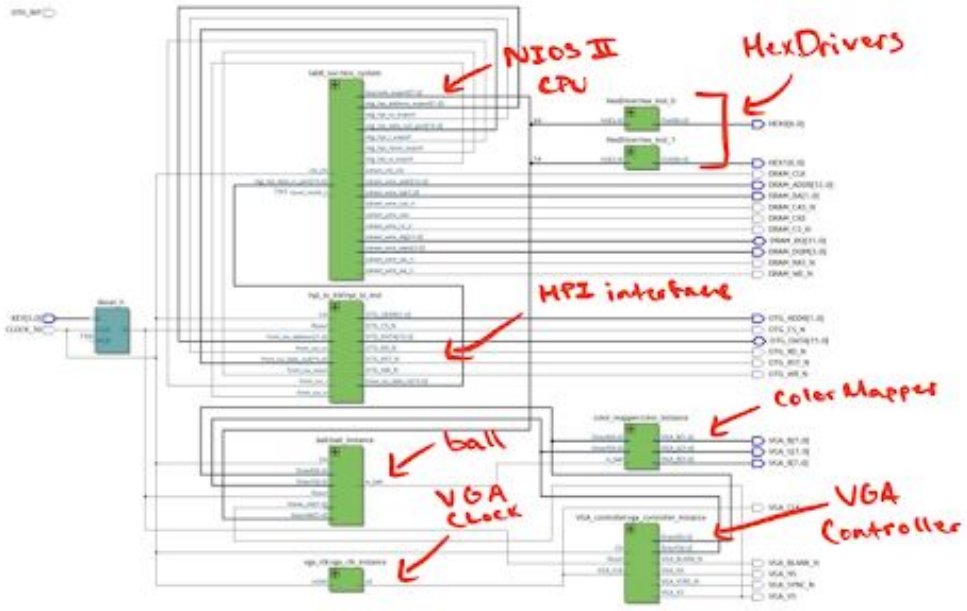
and vga controller are responsible for all the output connections. These connections include the R, G, B, VGA_CLK, VGA_SYNC, and VGA_BLANK which all pass through a VGA digital to analog converter which outputs only three connections: R, G, B. These three connections represent the updated pixel color on the monitor. The HS and VS signals do not pass through a converter, but rather go straight to the VGA output. The VGA output on the board connects with the VGA connection on the monitor through a VGA cable.

c. **Written Description of the CY7 to Host Protocol (HPI)** –
The CY7 is a host controller which constantly polls and requests the USB port for data when it is needed. If data is requested and the USB port sends in data, in our case in the form of keypresses, the CY7 controller receives the data and sends it to the HPI interface. This interface holds the data that it just received and keeps it until the next available clock cycle comes around. When it does come, the interface sends the data to the CPU. The main purpose of the CY7 to HPI is to ensure that the data the CPU receives is synchronized.

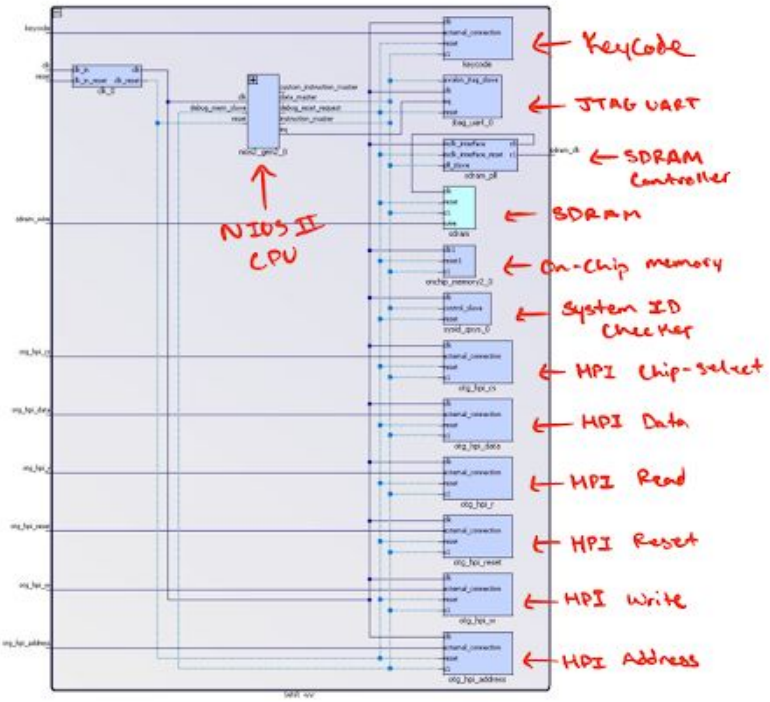
d. **Describe the purpose of the USBRead, USBWrite, IO_read, and IO_write functions in the C code** –
The purpose of the read and write function that we created in the C code is to essentially read and write data at certain memory addresses. The IO_read and IO_write functions are more general versions of read and write functions. They take in a parameter, such as data or an address, and access or write that parameter depending on the function. The USBRead and USBWrite functions call these IO functions. When executing USBWrite, we must first call the IO_write function for the address, to set the address to the correct location, and then call the same function again, but this time for data. When executing the USBRead function, we call the IO_write function for the address, and then call the IO_read function for the data to read to return the data in that address.

Block Diagram:

a. **Top-Level Block Diagram –**



b. **System-Level Block Diagram (Qsys View) –**



Module Descriptions:

Hex Drivers - Used to display data on the HEX displays on the FPGA board

VGA Controller - Contains the several connections for the VGA component and produces data signals, such as R, G, B, and HS and VS.

Keycode - Contains the ASCII value of the keypress on the keyboard, and is exported so the data is accessible to other hardware

SDRAM Controller – Contains instructions which the NIOS II processor executes from

SDRAM Clock – Special clock used for the SDRAM to avoid metastability and propagation delays

On-Chip Memory – Used to store data and values. Needs to be constantly refreshed or else will lose data

System ID Checker – Checks whether the hardware and software are compatible

HPI Chip Select - Contains the value for the chip select to control whether a read or write operation will be executed

HPI Data - Contains the data value passed in through the USB, and is exported as a wire so the data is accessible to other hardware

HPI Read - Contains the value for the read variable to indicate if the read operation should be executed

HPI Reset - Contains the value for the reset variable to indicate if the registers should be resetted

HPI Write - Contains the value for the write variable to indicate if the write operation should be executed

HPI Address - Contains the address that is passed through the USB port and is exported so it can be accessed by other hardware or programs

Post-Lab Questions:

LUT	2660
DSP	10
Memory (BRAM)	55,296
Flip-Flop	652
Frequency	76.78 Mhz
Static Power	105.37 mW
Dynamic Power	46.16 mW
Total Power	229.40 mW

The only problem that we had was that our ball wouldn't start in the center right away. The way we solved this problem was by setting the ball to the center upon hitting the reset button.

1. What is the difference between VGA_CLK and Clk?
    a. VGA_CLK is the pixel frequency (around 25Mhz) and this signal feeds into the VGA DAC. The Clk is the 50Mhz clock frequency for the NIOS 2.
2. 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. Because the usb data is 16 bits for one keycode and the read operation is a character pointer since we only need to use 1 bit to signal a read.

Hidden questions:

1. What are the advantages and/or disadvantages of using a USB interface over PS/2 interface to connect to the keyboard? List any two.
    a. USB:
        i. Advantages: It's universal, the USB interface can connect different types of devices (while PS/2 can only connect keyboard/mice). USB is hot-swappable (You can disconnect the device while the computer is running) while PS/2 is not.
        ii. Disadvantages: More complicated drivers. This can lead to your computer not recognizing your keyboard/mice (PS/2 has a much simpler interface). Larger data packets since USB protocol involves Control transfer, Setup Transfer, Data phase and status phase.
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?
    a. The old Ball_Y_Motion would be used. The difference in writing the two above is that the new position for the latter will be based upon the new motion. This means that the new ball position is not behind by a clock cycle and that when we check if the ball is within the bounds, we are using the newest position available. With a response to a keypress, we would get a faster "reaction" by one cycle.

Conclusion:

The functionality of our design was very straightforward. Since we prepared beforehand, we did not run into many issues or errors. One inconvenience that we did run into was that our ball started off not stationary, but rather moving. This was because we had programmed our ball to start stationary when the FPGA board was reset. We fixed this issue by ensuring that the FPGA board was resetted before the C program ran. Regarding ambiguity, this lab was straightforward as there were many different documentations and manuals provided which detailed and guided how to complete the lab. We believe there is nothing major that should be changed regarding how this lab was taught or implemented. Overall, this lab provided us with a better understanding of how the NIOS II CPU works, but more importantly, taught us how to utilize I/O peripherals and how the internal connections work.