ECE 385

Fall 2019

Experiment #8

SOC with USB and VGA Interface in SystemVerilog

Eric Dong, Yifu Guo
ericd3, yifuguo3
Section AB3, Thursday 2pm
Gene Shiue

Introduction

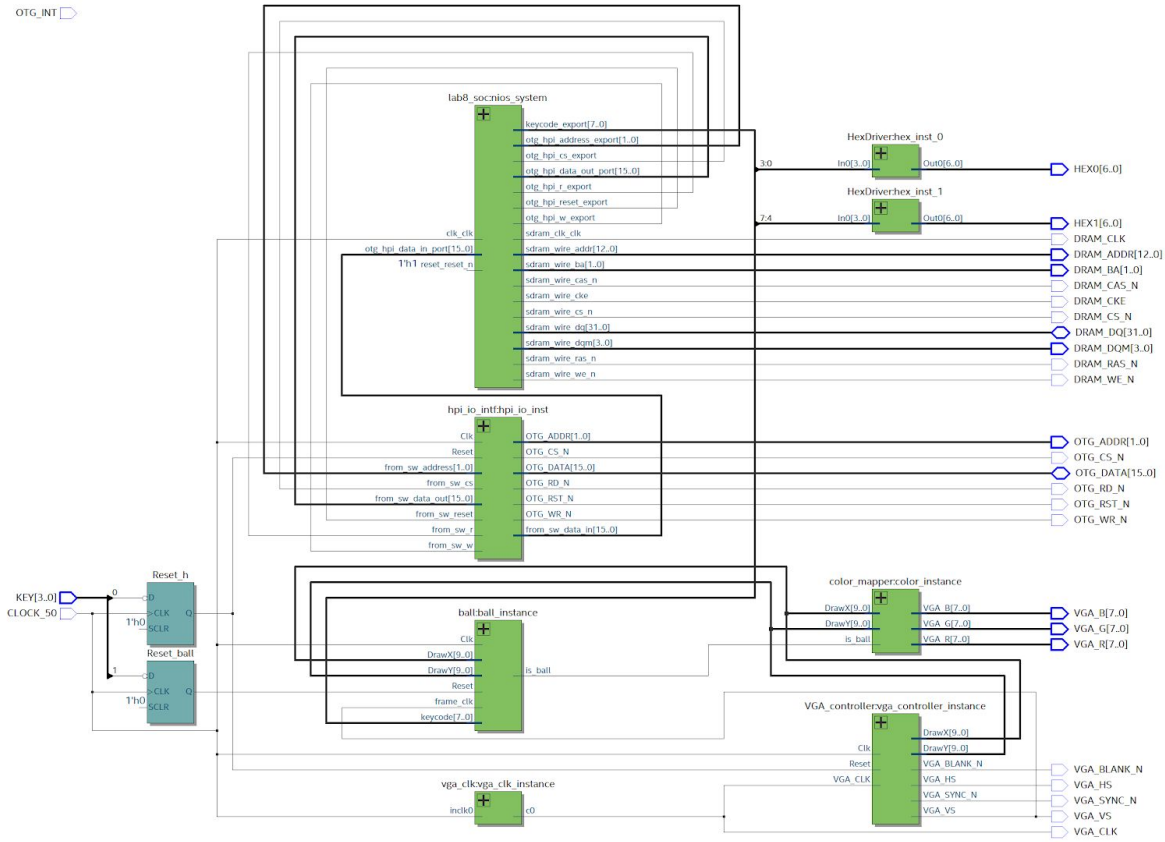
In this lab we were able to successfully implement a USB and VGA system that is able to control the movement of a ball on a screen with USb keyboard inputs. Depending on which WASD keys we type, the ball will go into the corresponding direction.

Written Description of Lab 8 System

Written Description of the entire Lab 8 system

- In lab 8, we setup a USB keyboard and VGA monitor to out DE2 board and implemented a USB driver that is able to write and read from the USB buffer. We first write an address of where we want to read or write, then we would send or receive data from the USB driver. For the VGA driver, we implemented a function that connects the top level of the VGA driver together. We connected a special VGA clock that runs half as slow as the DE2 board in order to achieve around 60Hz refresh rate. We then looked at the code that drives the horizontal and vertical sync and finally implemented the keyboard control for the ball.
- For the USB chip, the DE2 board has to first write an address into the chip in order to get any data back from it. Then based on the address that we wrote, we then will get data back if we are in read mode, or we are able to write data to the chip to send data to the host. As for the VGA chip, the DE2 board generates a 25Mhz signal in order to run the VGA component. This is for the horizontal sync to work properly since there are 480 horizontal lines and every time we start a new lines we have to tell set the horizontal sync.
- The CY7 interacts with the host through the NIOS 2 processor's software. In the C code, we write to several pointers in order to send the OTG chip the address, chip select, read, write, and data signals. When we want to send data, we first write to the address register the address that we want to write to, then we send to the data register address with the data we want to write. When we want to read, we first again write the address we want to read from the address register. Then we read from the data register with the data already ready.
- The IO_Write function simply writes to the address of the OTG chip with the address we want to write to with the data that we would like to write. The IO_Read function simply reads the data available from the address that we provide it with. The USBWrite function writes data from the NIOS processor to the OTG chip. It first writes an address to the address register, then it writes the desired data to the data register to send to the host. The USBRead function is used for reading data available from the device. We first write to the address register the address we want to read from, then we can read from the data register with the data from device already available.

Block diagram



Module descriptions

- Ball
  - Input
    - Clk, Reset, frame_clk, keycode, DrawX, DrawY
  - Output
    - is_ball
  - Description
    - This creates the ball and combines the keyboard with the motion.
  - Purpose
    - This module allows the ball to move in x+, x-, y+, y-, depending on the keyboard inputs. It also controls the way the ball bounces off the wall.
- Color_mapper
  - Input
    - is_ball, DrawX, DrawY
  - Output
    - VGA_R, VGA_G, VGA_B
  - Description
    - This module determines the color that should be displayed on the monitor.
  - Purpose

        - This module takes an input that tells whether the pixel that it is working on is a ball or not. If it is part of the ball, then the pixel is white, and if it is not part of the ball, then the color is just the background color.
- HexDriver
    - Input
        - In0
    - Output
        - Out0
    - Description
        - This displays the input to the module to the hexdisplays on the DE2 board.
    - Purpose
        - This module displays the keyboard input ASCII character onto the hex display so we know what we are pressing and for debugging.
- Hpi_io_intf
    - Input
        - Clk, Reset, from_sw_address, from_sw_data_out, from_sw_r, from_sw_w, from_sw_cs, from_sw_reset,
    - Output
        - From_sw_data_in, OTG_ADDR, OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N, OTG_DATA
    - Description
        - This module interfaces with the OTG chip that controls the USB and determines what signals should be outputted.
    - Purpose
        - This module sends the read, write, address, databuffer signals to the OTG chip. It also determines what signals should be high when the USB chip is being reset. Finally the module also allows us to control the input output bus.
- Lab8
    - Input
        - CLOCK_50, KEY, OTG_INT,
    - Output
        - HEX0, HEX1, VGA_R, VGA_G, VGA_B, VGA_CLK, VGA_SYNC_N, VGA_BLANK_N, VGA_VS, VGA_HS, OTG_DATA, OTG_ADDR, OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N, OTG_INTDRAM_ADDR, DRAM_DQ, DRAM_BA, DRAM_DQM, DRAM_RAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK
    - Description
        - This is the top level module of lab 8.
    - Purpose
        - This is to connect all the individual modules together. It makes the connections between the NIOS 2, easy OTG chip connection, as well as the VGA controller.
- Vga_clk

    - Input
        - inclk0
    - Output
        - c0
    - Description
        - This module generates the clock that controls the VGA controller.
    - Purpose
        - We need this module in order to generate the 25MHz clock for the VGA controller. This allows the VGA controller to update the screen at clock to 60Hz. We need it to be 25MHz since there are a total of 800 x 525 clock edges per screen which draws the 640x400 pixels.
- VGA_controller
    - Input
        - Clk, Reset, VGA_CLK
    - Output
        - VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, DrawX, DrawY
    - Description
        - This module controls the VGA signal including the vertical and horizontal syncs.
    - Purpose
        - This module determines the edge of the screen as well as when to output the vertical and horizontal sync signals. It also keeps track of which pixel we are working on so that the ball can be drawn at the correct location.

Answer to Post Lab Questions

1. VGA_CLK is only run at 25MHz which is half of what the speed of Clk is run at. This is the case so that we are able to run the VGA port and the monitor at 60Hz.
2. In the file, otg_hpi_r is defined as a char pointer so that the byte size of the pointer is only 1 byte long in comparison to the 4 bytes that an integer pointer would occupy, this would save memory in the chip.

Document the Design Resources and Statistics from the lab manual.

LUT	3,616
DSP	0
Memory	11,392
Flip-Flop	2178
Frequency	74.21 MHz
Static Power	105.28 mW

Dynamic Power	27.81 mW
Total Power	207.03 mW

Conclusion

- Our entire lab worked out nicely, we had trouble understanding how the HPI protocol worked but in the end we spent a lot of time reading the data sheets and we finally understood how everything links together.
- I think the HPI protocol was not clear to a lot of students. We had to dig for the information and I think it would be better if the information was touched on more in the lecture. The data sheets were hard to read and understand, if the professor could give a simpler version of the protocol, it would be great. Also giving information on how everything links together would also be fantastic.