ECE 385 Lab 8 Report

Chengting Yu, 3170111707
Yangkai Du, 3170112258

❑ Introduction

❑ Briefly summarize the operation of the USB/VGA interface

USB stands for the universal serial bus. Which is used for connecting external devices like keyboard. If the appropriate key is pressed, USB will put the keycode to the EZ-OTG chip, and EZ-OTG chip will store the keycode in it's memory and wait to be fetched by processor.

VGA stands for video graphic array. It is a connect port which can display the graphics on the monitor through VGA cable. VGA use electric beam scan the entire screen line by line. We can access the current pixel location of the electric beam and assign the color for specific pixel locations with color mapper.

❑ Written Description of Lab 8 System

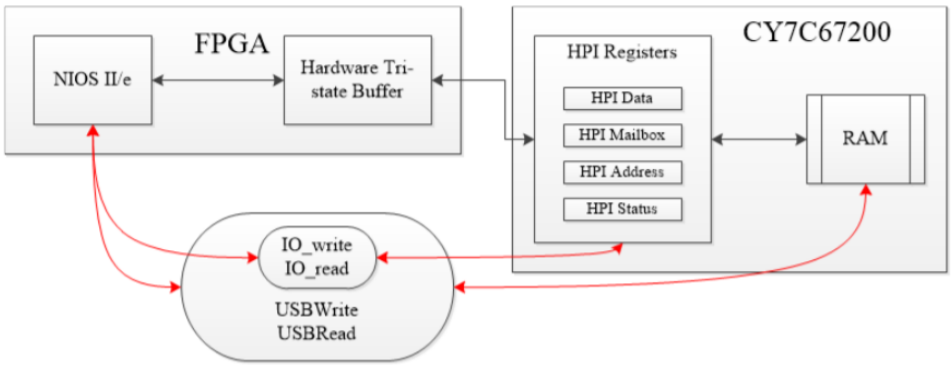
❑ Written Description of the entire Lab 8 system

The objective of Lab 8 system is to enable interaction between NIOS II CPU and VGA, USB port. The system support keyboard input through USB port and user can use keyboard "w", "s","a","d" to control the moving direction of the ball displayed by VGA. The ball will bounce back when it touch the edge of the screen.

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

CY7C67200 USB chip:

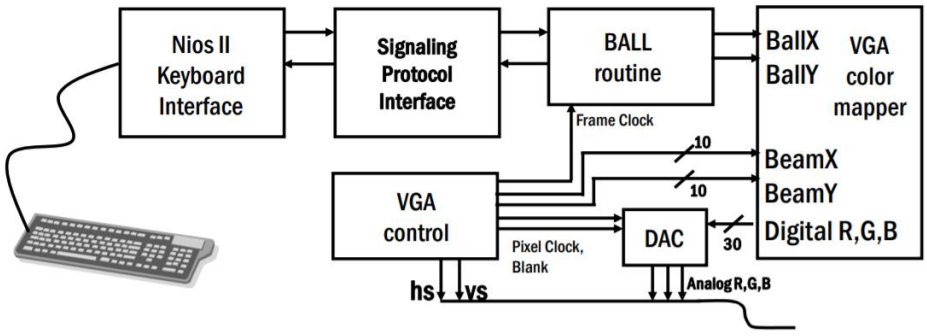
CY7C67200 USB chip is just ET-OZP chip, which serve as the internal chip between CPU and external device. It will keep poling data from the external devices like keyboard, and put the data on the chip RAM. On the CPU side, NIOS II can fetch the data from the chip RAM through USB Read/Write function, which serve as the protocol driver to interact with EZ-OTG chip.



VGA:

The NIOS II systems has a 8-bit keycode PIO, after CPU fetched the keycode which specify the key pressed by user from ET-OZP chip, the NIOS II system will then output the

fetched keycode of the key through 8-bit keycode PIO module to VGA side. Different key represents different moving direction of the Ball, the Ball routine of the VGA component will keep record position of the ball and its moving direction. Then, at each frame, the color mapper will assign each pixel with different color depends on the position of the ball.



Ball routine: partially given
Color mapper: given
VGA controller: given

❑

Written description of the USB protocol

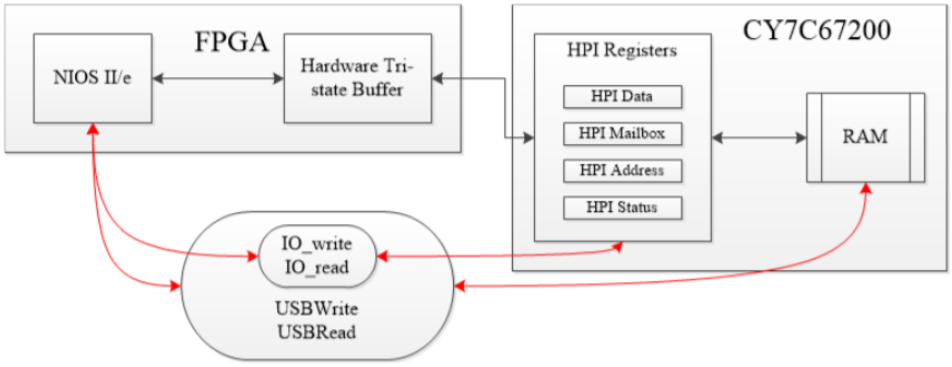
❑ Describe the purpose of the USBRead, USBWrite, IO_read and IO_write functions in the C code

USBRead: Allow external processor to directly acess the entire on-chip memory of EZ-OTG by first loading the hpi_address_register(0b10) with read address pointer through IO_write function, and then read the data in the memory pointed by the address register to data register(0b00), finally call the IO_read function to get the data from hpi_data_register.

USBWrite: Allow external processor to directly access the entire on-chip memory of EZ-OTG by first loading the hpi_address_register with write address pointer through IO_write function, and then load the data register(0b00) with data needs to be written through IO_write function, then it will be written to the address pointed by address register.

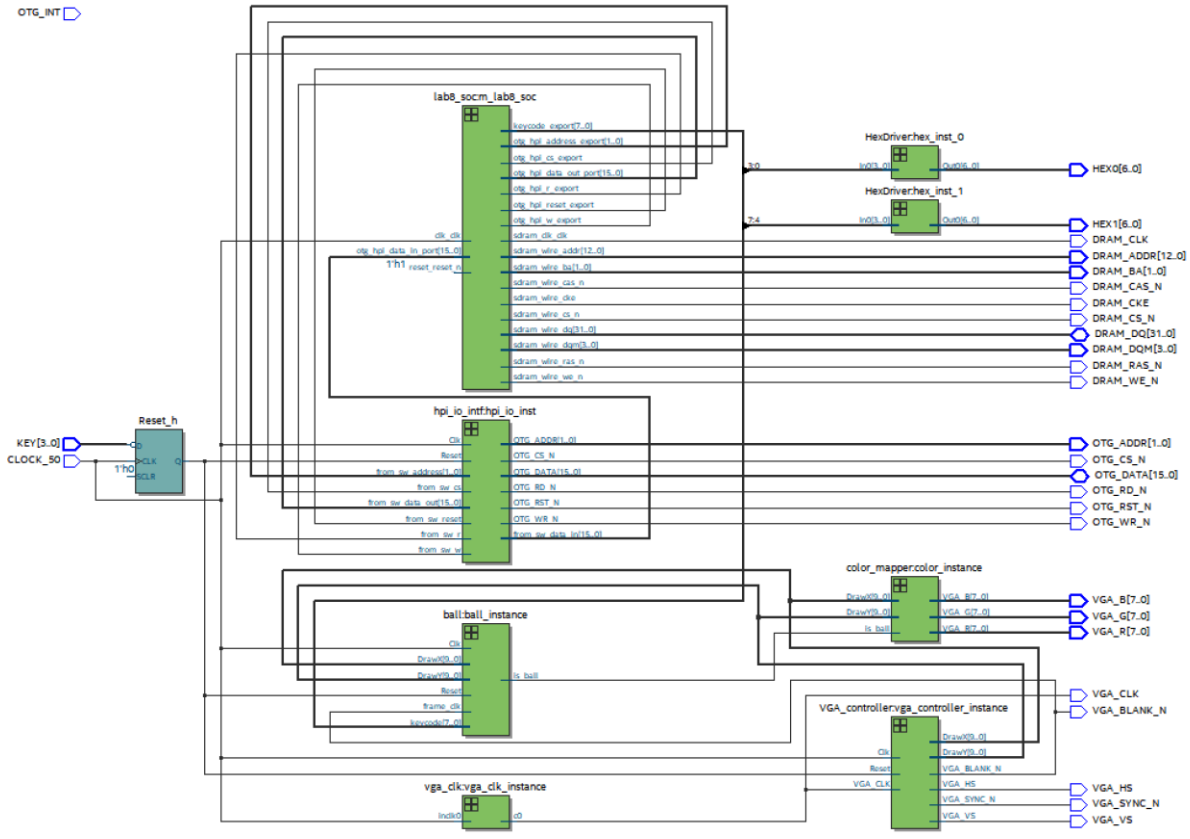
IO_read: Read the hpi registers specified by the 2-bit address. Serve as the routine to read the content from hpi register.

IO_write: Write the hpi registers specified by the 2-bit address. Serve as the routine to write the hpi register.



❏ Block diagram

❏ This diagram should represent the placement of all your modules in the top level. *Please only include the top level diagram and not the RTL view of every module*. The Qsys view of the NIOS processor is not necessary for this portion.



❏ Module descriptions

Module: lab8.sv

Inputs: CLOCK_50, OTG_INT,[3:0] KEY

outputs: [6:0] HEX0, HEX1, [7:0] VGA_R, VGA_G, VGA_B logic VGA_CLK,VGA_SYNC_N, VGA_BLANK_N, VGA_VS, VGA_HS, [1:0] OTG_ADDR, logic 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

Inouts: [15:0] OTG_DATA, [31:0] DRAM_DQ
Description: This is top level of the project, which connect all the modules together.
Purpose: Bridge between all the modules, make all the module work as a system.
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 // Active low
Inout: [15:0] OTG_DATA
Description: Read Write Interface between the NIOS II system and hpi registors.
Purpose: This module is need for read/write data from EZ-OTG chip and provide tri-state buffer to synchronous design,

Module: VGA_controller
Inputs: Clk, Reset, VGA_CLK,
Outputs: VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, [9:0] DrawX,DrawY
Description: Control the electron beam scan from the top left corner to the bottom right corner. A counter is used to keep track of the beam location
Purpose: It outputs the location of current pixel where the electron beam is located and the Color Mapper can assign color to that location.

Module: color_mapper.sv
Inputs: is_ball, [9:0]DrawX, DrawY
Outputs: [7:0] VGA_R, VGA_G, VGA_B
Description: If the given position descrebed DrawX and DrawY is within the ball, assign the color for ball, else assign the color for background.
Purpose: Assign color for each pixel to actually draw out the ball.

Module: ball.sv
Inputs: Clk, Rese, frame_clk, [9:0]DrawX,DrawY,[7:0]keycode
Outputs: is_ball
Description: Keep track of the ball position and updates ball position according to the keycode.
Purpose: Serve as the state machine for ball and implement the bouncing ball.

Module: HexDriver.sv
Inputs: [3:0] In0
Outputs: [6:0]Out0
Description: Convert the 4-bit value to 6-bit code for corresponding hex representation in LED
Purpose: Display the key pressed by user

PIO blocks:

Module: keycode
outputs: keycode_export[8:0]
Description: Output the keycode of the pressed key
Purpose: Interface to output the keycode out of NIOS II system

Module: otg_hpi_cs
Outputs: otg_hpi_cs_export
Description: Chip select signal for EZ-OTG chip
Purpose: it is needed for hpi_io_intf module

Module: otg_hpi_r
Outputs: otg_hpi_r_export
Description: This signal is high when read data from hpi registers
Purpose: Needed to enable IO_Read

Module: otg_hpi_w
Outputs: otg_hpi_w_export
Description: This signal is low when write data to hpi registers is enabled
Purpose: Needed to enable IO_Write

Module: otg_hpi_data
Inputs: [15:0] otg_hpi_data_in_port
outputs: [15:0] otg_hpi_data_out_port
Description: 16 bits data inout wire to read/write hpi_data_register
Purpose: Data Bridge between NIOS II system and otg chip

Module: otg_hpi_address
Input: [1:0] otg_hpi_address_export
Description: 2 bits address to specify wich hpi address to r/w
Purpose: Address wire between NIOS II system and otg chip

- **Answers to both hidden questions**
  - These can be found in the provided SystemVerilog files.

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.

Advantage:
The USB interface supports various types of devices besides keyboard and mouse, which shows much more general and common nowadays. Besides, the transfer rate of USB interface is quite higher than PS/2, which helps to reduce the connectors in real design.

Disadvantage:
Compared with PS/2, USB interface suffers higher latency (while USB uses polling to support keyboard and mouse, PS/2 directly send an interrupt to CPU, accelerating the execution of devices), which may cause problems in some cases. Besides, unlike PS/2 supports all keys conflict-free, the USB interface has the limitation of keys occurring at the same time.

Hidden Question #2/2:

Notice that Ball_Y_Pos is updated using Ball_Y_Motion.

a) Will the new value of Ball_Y_Motion be used when Ball_Y_Pos is updated, or the old?

b) 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;"?

c) How will this impact behavior of the ball during a bounce, and how might that interact with a response to a keypress? Give an answer in your Post-Lab.

a) The old value, stored in registers (in always_ff block), will be used in the updates of position.
b) The difference of two assignment is based on the operand, "Ball_Y_Motion" vs "Ball_Y_Motion_in". By using "Ball_Y_Motion", we use old value to update postions, while new value by using "Ball_Y_Motion_in". That is, when using "Ball_Y_Motion_in", we actually obtain the motion of next state and use it to update the current position, which is not expected in our design. Thus, here we use simply "Ball_Y_Pos_in = Ball_Y_Pos + Ball_Y_Motion;" in our code.
c) Using "Ball_Y_Pos_in = Ball_Y_Pos + Ball_Y_Motion;" is equivalent to wrap up the computation as well as assignment (selecting from MUXex) with the temporary register (Ball_Y_Motion here in always_ff). Replacing it by "Ball_Y_Motion_in", we actually abandon the register, and in this case, several problems occurs. On one hand, the computation of Ball_Y_Motion_in takes time (changing motion at boundary), and needs to be determined with enough time, so fails in this case (may cause some unexpected behaviors during a bounce). On the other hand, it fails to deal with the synchronization between the keypress and our system, which could cause problems in the interact, for example, crash out when typing crazily.

❑ **Answer to Post Lab Questions**

❑ What is the difference between VGA_clk and Clk?

In our design, the VGA_clk represents to 25 MHz VGA clock input using in VGA controller, while Clk is the 50 MHz clock for the other hardwares (flip-flops) of our systems.

❑ **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?**

Since the width band of port of otg_hpi_data is 16-bit, we need define it as integer pointer, which points to alt_u16 (the unsigned 16-bit integer), in this case, we could assign (or obtain) a 16-bit value from address pointed by otg_hpi_data.
Since the value pointed by otg_hpi_r is either 0 or 1, it saves memory space that define it as a char pointer, which points to alt_u8, taking 8-bit instead of 16-bit.

❑ Document the Design Resources and Statistics in following table.

LUT	2692
DSP	10
Memory (BRAM)	55296bits
Flip-Flop	2235
Frequency	88.86MHZ
Static Power	105.29mW
Dynamic Power	28.98mW
Total Power	208.06mW

❑ **Conclusion**

❑ Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it
The design seems work well, though we cannot actually perform it.

❑ Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester? You can also specify what we did right so it doesn't get changed.
More references should have been given.