

**ECE 385**

Fall 2020

Experiment # 8

**SOC with USB and VGA Interface  
in SystemVerilog**

Lian Xinyu & Liu Tianyu

TA: Li Shuren

## **1. Introduction:**

In lab8, we are required to use the USB and VGA interface to control the ball that displays on the screen. We use W, A, S, D to control the ball move up, left, down, right correspondingly. When the ball touches the boundary of the screen, it will bounce back. And after each key press, the result will also display by the LED on the FPGA board.

The interface receives a signal from the USB keyboard and stores the signal to RAM. Then the NIOS II/e processor sends signal to the USB controller to read the signal sent by keyboard from the RAM. The signal goes through HPI registers on the EZ-OTG chip, hardware Tri-state buffer on the FPGA and finally reached the NIOS II/e processor. Then FPGA chip generates signals to the VGA monitor to display the result.

## **2. Written Description of the Lab 8 System:**

### **a. Written Description of the entire Lab 8 system.**

This lab is a new combination of hardware and software, including the USB and VGA interface which can be regard as a part of the hardware. Our task is to design the project that the ball displayed on the screen can respond to the keyboard signal. The detailed keyboard signal is W, A, S, D, which is corresponding to moving up, left, down and right. We want the ball to bounce back whenever it hit an edge of the screen and do not have any diagonal movement in some special case. For reset, our design is that when we press the reset button, the ball will appear on the center of the screen and move downward. For the hardware part, the USB port received signal from the keyboard and send it to the controller. If needed, the controller sends the signal through the HPI interface to NIOS II/e cpu. Then display the signal by the ball on the screen.

### **b. Describe in words how the NIOS interacts with both the CY7C67200 USB chip and VGA components.**

The NIOS interacts with the USB chip through the host protocol interface(HPI). Whenever information is received by the chip, the chip sends it through the HPI module to EZ-OTG. In our real design, chip writes the RAM address to HPI address register and the data to HPI Data

register, then the EZ-OTG chip writes the data in HPI Data register to RAM.

For the VGA components, then NIOS sends the signal to the FPGA chip, and then modules ball, color\_mapper and VGA\_controller on the FGPA chip generate combined signals to the VGA monitor.

### **c. Written description of the CY7 to host protocol(HPI)**

Through software running on the Nios2 processor, the CY7 is a controller and can interact with the host.

For C code, Firstly, store the address we want to read from into the HPI address register, then changing the chip select signal and the writing signal to 0 to tell the HPI the process is ready and store the data we want to write into the HPI data register. After that, telling HPI the write process is done and turn the chip select signal and the writing signal back to 1. The reading part are similar.

### **d. Describe the purpose of the USBRead, USBWrite, IO\_read and IO\_write functions in the C code.**

**USBRead:** this function reads the data from the RAM that the address is specified by the input.

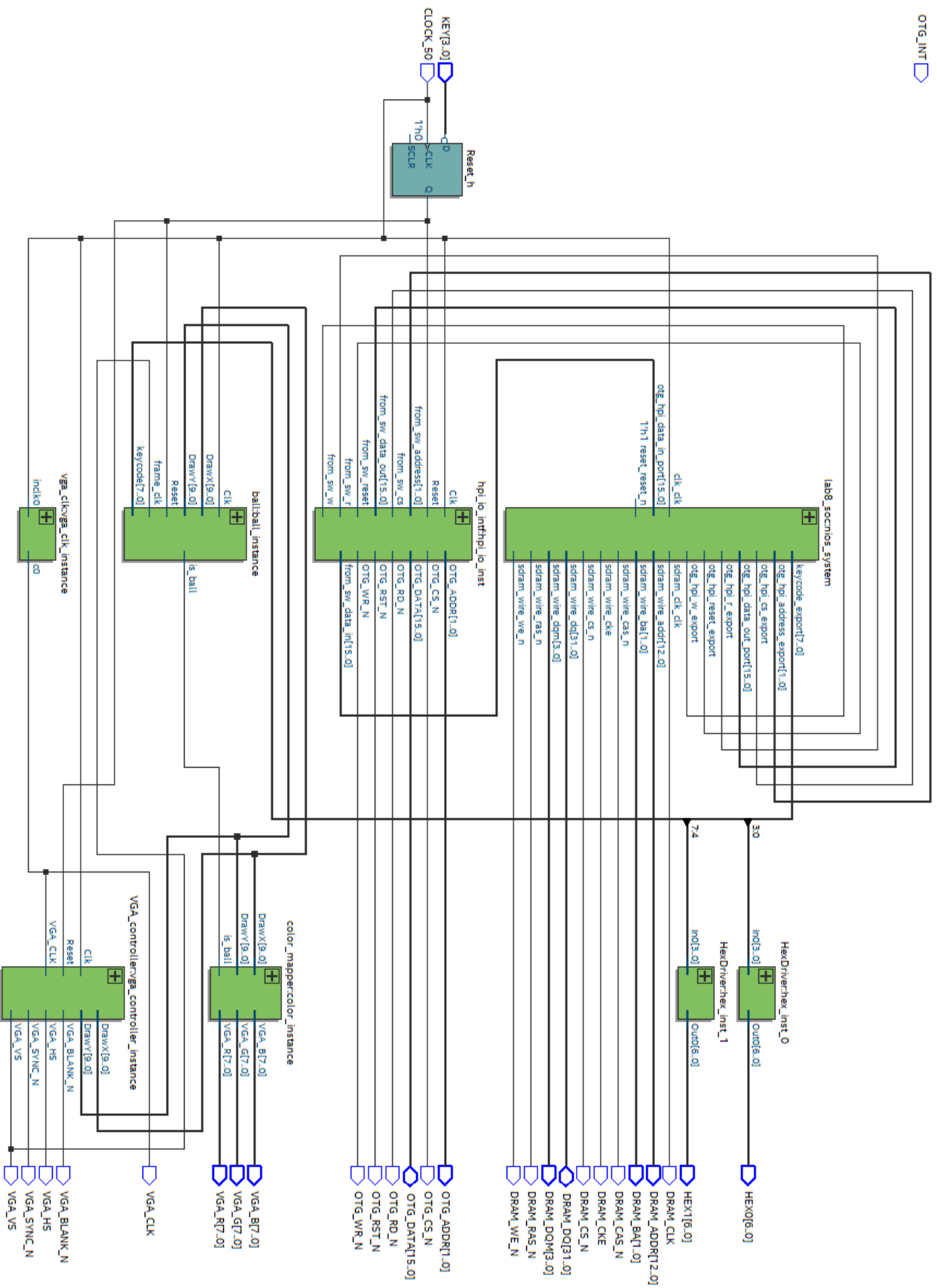
**USBWrite:** this function writes the input Data to the address specified by the input Address to the RAM.

**IO\_read:** this function reads and returns the data from the HPI register specified by the input Address.

**IO\_write:** this function writes the input Data to the HPI register specified by the input Address.

For more details, please check files usb.c and io\_handler.c.

## **3. Block diagram**

OTG\_INT 

## 4. Module descriptions

### Module: VGA Controller

```
module VGA_controller (input      clk,           // 50 MHz clock
                      output logic Reset,        // Active-high reset signal
                      output logic VGA_HS,       // Horizontal sync pulse. Active low
                      input  logic VGA_VS,       // Vertical sync pulse. Active low
                      output logic VGA_CLK,       // 25 MHz VGA clock input
                      input  logic VGA_BLANK_N,   // Blanking interval indicator. Active low.
                      output logic VGA_SYNC_N,   // Composite Sync signal. Active low. We don't use it in this lab,
                                                  // but the video DAC on the DE2 board requires an input for it.
                      output logic [9:0] DrawX,   // horizontal coordinate
                      output logic DrawY        // vertical coordinate
);
```

**Description:** This module produces synchronization timing signals to VS and HS of the VGA to help display on the VGA monitor.

**Purpose:** This module acts as the VGA controller.

### Module: ball

```
module ball (input      clk,           // 50 MHz clock
             input      Reset,        // Active-high reset signal
             input      frame_clk,    // The clock indicating a new frame (~60Hz)
             input [9:0] DrawX, DrawY, // Current pixel coordinates
             input logic [7:0] keycode,
             output logic is_ball     // whether current pixel belongs to ball or background
);
```

**Description:** At every rising edge of clock, update the position and motion of the ball. And prevent the ball to run out of the monitor.

**Purpose:** This module defines the state of the ball.

### Module: color\_mapper

```
module color_mapper (input      is_ball,           // whether current pixel belongs to ball
                    input [9:0] DrawX, DrawY,     // or background (computed in ball.sv)
                    output logic [7:0] VGA_R, VGA_G, VGA_B // Current pixel coordinates
                    // VGA RGB output
);
```

**Description:** This module decides the color of every pixel for the VGA. It calculates the RGB values based on the current ball position.

**Purpose:** This module is used to draw the ball, background, and implement RGB colors on screen.

### Module: HexDriver

```
module HexDriver (input [3:0] In0,
                  output logic [6:0] out0);
```

**Description:** This is a hexdriver that transforms the binary value in In0 to a form that is suitable to be displayed on the DE-2 board.

**Purpose:** This module is used to display the 8-bit keycode in Hex Displays on the DE-2 board.

### Module: hpi\_to\_intf

```
module hpi_io_intf( input      clk, Reset,
                   input [1:0]  from_sw_address,
                   output [15:0] from_sw_data_in,
                   input  [15:0] from_sw_data_out,
                   input  from_sw_r, from_sw_w, from_sw_cs, from_sw_reset, // Active low
                   inout  [15:0] OTG_DATA,
                   output [1:0]  OTG_ADDR,
                   output      OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N // Active low
                 );
```

**Description:** This module ensures the OTG\_DATA is high when NIOS is not writing to OTG\_DATA inout bus and handles buffers.

**Purpose:** This module is used to create an interface between NIOS II/e and EZ-OTG chip.

### Module: lab8

```
module lab8( input      CLOCK_50,
             input  [3:0] KEY, //bit 0 is set up as Reset
             output logic [6:0] HEX0, HEX1,
             // VGA Interface
             output logic [7:0] VGA_R, //VGA Red
             VGA_G, //VGA Green
             VGA_B, //VGA Blue
             output logic VGA_CLK, //VGA Clock
             VGA_SYNC_N, //VGA Sync signal
             VGA_BLANK_N, //VGA Blank signal
             VGA_VS, //VGA vertical sync signal
             VGA_HS, //VGA horizontal sync signal
             // CY7C67200 Interface
             inout wire [15:0] OTG_DATA, //CY7C67200 Data bus 16 Bits
             output logic [1:0] OTG_ADDR, //CY7C67200 Address 2 Bits
             output logic OTG_CS_N, //CY7C67200 Chip Select
             OTG_RD_N, //CY7C67200 Read
             OTG_WR_N, //CY7C67200 Write
             OTG_RST_N, //CY7C67200 Reset
             OTG_INT, //CY7C67200 Interrupt
             input
             // SDRAM Interface for Nios II Software
             output logic [12:0] DRAM_ADDR, //SDRAM Address 13 Bits
             inout wire [31:0] DRAM_DQ, //SDRAM Data 32 Bits
             output logic [1:0] DRAM_BA, //SDRAM Bank Address 2 Bits
             output logic [3:0] DRAM_DQM, //SDRAM Data Mask 4 Bits
             DRAM_RAS_N, //SDRAM Row Address Strobe
             DRAM_CAS_N, //SDRAM Column Address Strobe
             DRAM_CKE, //SDRAM Clock Enable
             DRAM_WE_N, //SDRAM Write Enable
             DRAM_CS_N, //SDRAM Chip Select
             DRAM_CLK //SDRAM Clock
           );
```

**Description:** This module is the top level for lab8, it connects the NIOS


to all the blocks and drivers.

**Purpose:** This module creates the SOP.

## Platform Designer Modules:

<b>clk_0</b>	Clock Source		
clk_in	Clock Input	<b>clk</b>	<b>exported</b>
clk_in_reset	Reset Input	<b>reset</b>	
clk	Clock Output	<i>Double-click to</i>	clk_0
clk_reset	Reset Output	<i>Double-click to</i>	

This block will generate the main block signal for all other modules

 <b>nios2_gen2_0</b>	Nios II Processor		
clk	Clock Input	<i>Double-click to</i>	<b>clk_0</b>
reset	Reset Input	<i>Double-click to</i>	[clk]
data_master	Avalon Memory Mapped Master	<i>Double-click to</i>	[clk]
instruction_master	Avalon Memory Mapped Master	<i>Double-click to</i>	[clk]
irq	Interrupt Receiver	<i>Double-click to</i>	[clk]
debug_reset_request	Reset Output	<i>Double-click to</i>	[clk]
debug_mem_slave	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]
custom_instructi...	Custom Instruction Master	<i>Double-click to</i>	

This block is NIOS-II (economic), a processor we use to perform computing and processing tasks in hardware level.

<b>onchip_memory2_0</b>	On-Chip Memory (RAM or ROM) I...		
clk1	Clock Input	<i>Double-click to</i>	<b>clk_0</b>
s1	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk1]
reset1	Reset Input	<i>Double-click to</i>	[clk1]

This block is the memory for NIOS-II to use. It can load or store data required in this lab.

<b>sdram</b>	SDRAM Controller Intel FPGA IP		
clk	Clock Input	<i>Double-click to</i>	<b>sdram_p...</b>
reset	Reset Input	<i>Double-click to</i>	[clk]
s1	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]
wire	Conduit	<b>sdram_wire</b>	

This block is also known as synchronous dynamic random-access memory. The system can use the memory specified by SDRAM.

<b>sdram_pll</b>	ALTPLL Intel FPGA IP		
inclk_interface	Clock Input	<i>Double-click to</i>	<b>clk_0</b>
inclk_interface...	Reset Input	<i>Double-click to</i>	[inclk_in...
pll_slave	Avalon Memory Mapped Slave	<i>Double-click to</i>	[inclk_in...
c0	Clock Output	<i>Double-click to</i>	sdram_pll_c0
c1	Clock Output	<b>sdram_clk</b>	sdram_pll_c1

This block works as the PLL for the whole system, which controls a clock signal with a -3 ns prior compared with main block. Under this design, the sdram block

will start processing before the clock edge, so the sdram will process the data faster.

▢ <b>sysid_qsys_0</b>	System ID Peripheral Intel FP...		
clk	Clock Input	<i>Double-click to</i>	<b>clk_0</b>
reset	Reset Input	<i>Double-click to</i>	[clk]
control_slave	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]

This block is a system id checker. To make sure that the id configuration of our hardware part is corresponding to the software part, we need to add this module.

▢ <b>keycode</b>	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	<i>Double-click to</i>	<b>clk_0</b>
reset	Reset Input	<i>Double-click to</i>	[clk]
s1	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]
external_connection	Conduit	<b>keycode</b>	

This block is a PIO block that output the 8-bit keycode from the keyboard.

▢ <b>otg_hpi_address</b>	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	<i>Double-click to</i>	<b>clk_0</b>
reset	Reset Input	<i>Double-click to</i>	[clk]
s1	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]
external_connection	Conduit	<b>otg_hpi_address</b>	

This block is a PIO block that output the 2-bit value corresponding to the specific HPI register

▢ <b>otg_hpi_data</b>	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	<i>Double-click to</i>	<b>clk_0</b>
reset	Reset Input	<i>Double-click to</i>	[clk]
s1	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]
external_connection	Conduit	<b>otg_hpi_data</b>	

This block is a PIO block that is inout because the data can be both read from and written to.

▢ <b>otg_hpi_r</b>	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	<i>Double-click to</i>	<b>clk_0</b>
reset	Reset Input	<i>Double-click to</i>	[clk]
s1	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]
external_connection	Conduit	<b>otg_hpi_r</b>	

This block is a PIO block that is 1 bit output corresponding to a “read” enable signal.

▢ <b>otg_hpi_v</b>	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	<i>Double-click to</i>	<b>clk_0</b>
reset	Reset Input	<i>Double-click to</i>	[clk]
s1	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]
external_connection	Conduit	<b>otg_hpi_v</b>	

This block is a PIO block that is 1 bit output corresponding to a “write” enable signal.

▢ <b>otg_hpi_cs</b>	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	<i>Double-click to</i>	<b>clk_0</b>
reset	Reset Input	<i>Double-click to</i>	[clk]
s1	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]
external_connection	Conduit	<b>otg_hpi_cs</b>	

This block is a PIO block that is 1 bit output corresponding to a “chip enable” enable signal.



❏ <b>otg_hpi_reset</b>	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	<i>Double-click to</i>	<b>clk_0</b>
reset	Reset Input	<i>Double-click to</i>	[clk]
s1	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]
external_connection	Conduit	<b>otg_hpi_reset</b>	

This block is a PIO block that is 1 bit output corresponding to a “reset” enable signal.

❏ <b>jtag_uart_0</b>	JTAG UART Intel FPGA IP		
clk	Clock Input	<i>Double-click to</i>	<b>clk_0</b>
reset	Reset Input	<i>Double-click to</i>	[clk]
avalon_jtag_slave	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]
irq	Interrupt Sender	<i>Double-click to</i>	[clk]

## 5. Answer the Post Lab Questions

<b>LUT</b>	2674
<b>DSP</b>	10
<b>Memory (BRAM) (bits)</b>	11392
<b>Flip-Flop</b>	2180
<b>Frequency(MHz)</b>	129.7
<b>Static Power(mW)</b>	105.16
<b>Dynamic Power(mW)</b>	0.88
<b>I/O Thermal Power(mW)</b>	71.87
<b>Total Power(mW)</b>	177.91

### What’s the difference between VGA\_CLK and Clk?

VGA\_clk is at 25MHz and updated the frames on the monitor while Clk is at 50MHz and drove the FPGA board operations.

**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?**

The data width for the integer is 16bits and for char is 8bits. Otg\_hpi\_r is a signal that determines whether or not the OTG interface is ready to be read, so it only need be a char. Otg\_hpi\_address passes an address so it

would require 16 bits int rather than a char.

**Hidden question1: 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:

1. The USB supports hot plug, which means that you can plug it in and use it immediately, and then you can just pull the USB plug out when it's not in use. But not for PS2 devices.
2. USB is a bus that can connect with many devices, however, the PS/2 port can only connect with one device.

Disadvantages:

1. USB uses polling method while PS/2 uses interrupt method. In addition, the interrupt method will not keep asking the keyboard for input, so the efficiency of PS/2 will be higher than USB.
2. If USB is used by different devices, there will be delay in the response time. However, for PS/2 port, this will not happen because interrupt always work as the highest priority in operating systems.

**Hidden question2: 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? Give an answer in your Post-Lab.**

Answer: 1. It has been updated when Ball\_Y\_Pos is updated. 2. For the design "Ball\_Y\_Pos\_in = Ball\_Y\_Pos + Ball\_Y\_Motion;", the frame will create one delay so that the ball will go through the boundary for a while and then back to the screen; for the design "Ball\_Y\_Pos\_in = Ball\_Y\_Pos + Ball\_Y\_Motion\_in;", the ball will response the keyboard immediately

and there will be no frame delay for the ball moving.

## **Conclusion**

### **1. Bugs and errors:**

Thanks to Pro. Li, we learnt a lot in the lectures, and we can finish the design of NIOS II based on SOC by the help of our TAs. Nevertheless, we still met some strange problems and we spent plenty of time to solve them. The first one is component design for SOC file, at first we found that the IP of our design was different from our board. Although if we ignored this error, the processor could still work well, we intended to solve this problem. Finally we discovered that we could not copy the file directly from the SOC in lab7, since NIOS-II will still recognize the copied file in its original path, this maybe a bug for this platform. Another problem is to prevent the ball from moving diagonally. To solve this problem, we need to make sure that at any moment, whether bouncing or moving, we must set that the ball had velocity in only one of the X or Y directions.

### **2. Summary**

In this lab, we not only implemented the connection among the NIOS-II, EZ-OTG, USB and VGA, but design a program which can control the movement of a ball. It was difficult to design the above features, but we overcame the obstacles and experienced the efficiency

and flexibility of the HW-SW collaborative programming. In addition, it is also interesting to understand how VGA signals are fed into the monitor and how keyboard signals are fed into the processor and processed. We hope that we will learn more in the next lab.