ECE385 Experiment #8

Eric Meyers, Ryan Helsdingen

Section ABG; TAs: Ben Delay, Shuo Liu

March 30th, 2016

emeyer7, helsdin2

I. Introduction

The purpose of this lab is to introduce concepts pertaining to USB protocol/communication and VGA display. The main goal of this lab was to connect a USB keyboard to the Altera FPGA and allow a user to control a ball displayed on a VGA-connected monitor.

Throughout this lab we interface with both a keyboard through an on-board USB input port and a monitor through a VGA output port on the DE2 board. Direction keys (W-A-S-D) are used to control the direction of a ball moving on the VGA monitor. W is up, S is down, A is left, and D is right. If the ball hits the edge of the screen with no other keys being pushed, it switches direction by 180 degrees. The ball continues to move in the current direction if no new key is pressed and no screen edge is hit. The last pressed 8 bit hex scan code is displayed on two of the hex displays and the direction key pushed is shown via the green LEDs on the board.

II. Description of Circuit

Much of the code needed for the lab was provided. This included most of the modules used, the framework of the SystemVerilog code needed to connect these modules, and the base C code to make the ball move in two directions on a VGA screen.

Tasks that needed to be completed included creating a proper Qsys setup for the NIOS II processor to interface with the CY7C67200, building a top level entity in System Verilog to bring all the files provided together, defining a hardware I/O wrapper to connect the PIO's from the NIOS II to the HPI registers on the CY7C67200 chip, correctly interpreting and implementing the IO_read and IO_write methods for accessing the register bank of the CYC67200 chip, using these read and write commands to build USBRead and USBWrite methods, and lastly editing the C code for the ball to function as stated above adding 2 more directions and if/else conditions for boundary cases.

A. NIOS II System

The NIOS II System is the processor used for low-speed low-performance tasks such as executing the high level C code. The NIOS II System is made up of several submodules or PIOs including the NIOS II processor which works with the C code , on-chip memory, SDRAM controller for simplifying interfacing with the SDRAM, SDRAM_PLL which provides a phase shifted clock for the SDRAM, the JTAG_UART submodule which allows for debugging through printf and scanf functions. The PIOs help the C code interact with SystemVerilog by providing addresses for the code to send information to and from various hardware.

B. CYC67200 chip

The CYC67200 chip manages information sent through the USB port. It is controlled by a hardware I/O wrapper in System Verilog that connects the HPI registers to the NIOS II processor. In Qsys, PIO ports are created for keycode, HPI address, HPI chip select, HPI data, HPI read, and HPI write bits. HPI_DATA holds the 16 bit data that will be moved between the FPGA and EZ-OTG. HPI_ADDRESS is two bits of the address where the data will be read/written to whether it be HPI DATA, HPI MAILBOX, HPI ADDRESS, or HPI STATUS. Fig. 1 shows the four registers matching up with their respective addresses.

The IO_read and IO_write methods are used for accessing the register bank of the CYC67200 chip. We use these read and write commands to build USBRead and USBWrite methods to read and write to the internal memory of the EZ-OTG.

Port Registers	HPI A [1]	HPI A [0]	Access
HPI DATA	0	0	RW
HPI MAILBOX	0	1	RW
HPI ADDRESS	1	0	W
HPI STATUS	1	1	R

Fig. 1: Port registers for different addresses. (lab manual)

1) IO_read: To read and return data from the memory location specified by the two bit memory address, we use the IO_read function.

2) IO_write: To write data to the memory location specified by the two bit memory address, we use the IO_write function.

Two helper functions, USB_Read and USB_Write are used to simplify the process.

3) USB_read: This helper function takes in an address from EZ-OTG memory, writes it to the EZ-OTG address register using IO_write function, and reads the EZ-OTG data register at that address using IO_read.

4) USB_write: This helper function takes in an address from EZ-OTG memory, writes it to the EZ-OTG address register using IO_write function, and this time writes to the EZ-OTG data register at that address using IO_write.

Code for Ball Movement

For the ball, C code was added to control ball movement. Section III of this report labeled 'Purpose of Modules' goes into further detail on each module used in the lab including the ball.sv module.

III. Purpose of Modules

A. lab8.sv

This is the top-level module for lab 8. Inputs include the 50 MHz clock, push button, data and interrupt from the CY7C67200 chip, and data from the SDRAM. Outputs include the two hex displays, the green LEDs, the DRAM interface, rest of the interface to the CY7C67200, and VGA.

The module includes several instantiations. hpi_io_intf instantiates the interaction between the hardware and usb, nios_system instantiates the connections for the NIOS II system, vga_controller, ball, and collor_mapper instantiate connections for their respective modules.

B. hpi_io_intf.sv

This method takes HPI output values from the NIOS II as input, checks for a call to reset, and if no reset, assigns these values to the proper values to be outputted by the top-level module to the CY7C67200 chip. Inputs and outputs used are shown below.

```
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,
 inout [15:0] OTG_DATA,
 output[1:0]  OTG_ADDR,
 output       OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N,
 input        OTG_INT, Clk, Reset);
```

C. VGA_controller.sv

The VGA_controller manages output to the monitor. Inputs include clock and reset. Outputs include several sync signals, a pixel clock specified for 25 MHz and 10 bit horizontal and vertical coordinate signals as shown below.

```
input        Clk,       // 50 MHz clock
             Reset,     // reset signal
output logic hs,        // Horizontal sync pulse.  Active low
             vs,        // Vertical sync pulse.  Active low
             pixel_clk, // 25 MHz pixel clock output
             blank,     // Blanking interval indicator.  Active low.
             sync,      // 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 [9:0] DrawX,     // horizontal coordinate
             DrawY );   // vertical coordinate
```

Most of this module was given to the team.

D. Ball.sv

This module defines the motion of the ball. How big the ball size is, how fast and far the steps are for the ball movement, and the x and y coordinate for the ball are all given in this module. Inputs include the reset, clock, and signals for the four directionals. Outputs include the 10-bit values for x and y coordinates of the ball as well as size of the ball, BallS.

```
module  ball ( input Reset, frame_clk,
               output [9:0]  BallX, BallY, BallS,
               input up, down, left, right);
```

Parameters are set to control where the ball can travel and step size of the ball.

```
parameter [9:0] Ball_X_Center=320;  // Center position on the X axis
parameter [9:0] Ball_Y_Center=240;  // Center position on the Y axis
parameter [9:0] Ball_X_Min=0;       // Leftmost point on the X axis
parameter [9:0] Ball_X_Max=639;     // Rightmost point on the X axis
parameter [9:0] Ball_Y_Min=0;       // Topmost point on the Y axis
parameter [9:0] Ball_Y_Max=479;     // Bottommost point on the Y axis
parameter [9:0] Ball_X_Step=1;      // Step size on the X axis
parameter [9:0] Ball_Y_Step=1;      // Step size on the Y axis
```

The following code sets the ball to reverse direction when the ball hits one of its boundaries.

```
if ( (Ball_Y_Pos + Ball_Size) >= Ball_Y_Max )
    Ball_Y_Motion <= (~ (Ball_Y_Step) + 1'b1);
else if ( (Ball_Y_Pos - Ball_Size) <= Ball_Y_Min )
    Ball_Y_Motion <= Ball_Y_Step;
else if ( (Ball_X_Pos + Ball_Size) >= Ball_X_Max )
    Ball_X_Motion <= (~ (Ball_X_Step) + 1'b1);
else if ( (Ball_X_Pos - Ball_Size) <= Ball_X_Min )
    Ball_X_Motion <= Ball_X_Step;
```

The ball is then given set directions for given key codes from the keyboard as shown below. Directions are based off directional input values.

```
begin
   if(up)//RIGHT BUTTON PRESSED = "W"
   begin
      Ball_X_Motion <= 10'b0;
      Ball_Y_Motion <= ~(Ball_Y_Step) + 1;
   end
   else if(down) //DOWN BUTTON PRESSED = "S"
   begin
      Ball_X_Motion <= 10'b0;
      Ball_Y_Motion <= Ball_Y_Step;
   end
   else if(left) //LEFT BUTTON PRESSED = "A"
   begin
      Ball_Y_Motion <= 10'b0;
      Ball_X_Motion <= ~(Ball_X_Step) + 1;
   end
   else if(right) //RIGHT BUTTON PRESSED = "D"
   begin
      Ball_Y_Motion <= 10'b0;
      Ball_X_Motion <= Ball_X_Step;
   end
   else //DEFAULT CASE - KEEP MOTION SAME
   begin
      Ball_Y_Motion <= Ball_Y_Motion;
      Ball_X_Motion <= Ball_X_Motion;
   end
```

E. color_mapper.sv

The color_mapper module controls the color of the pixels where the ball is located and maintains proper ball shape as the ball moves from location to location. Inputs for this module include the coordinates to the center of the ball, the draw coordinates, and the size of the ball. Outputs include the RGB array for that draw coordinate.

```
input        [9:0] BallX, BallY, DrawX, DrawY, Ball_size,
output logic [7:0]  Red, Green, Blue );
```

This module was given to the team.

IV. Description of USB Protocol & Changes

The USB protocol in Experiment 8 utilized the Cypress EZ-OTG (CY7C67200) USB controller on board the Altera. The CY7C67200 was used as a host controller and once a USB keyboard is plugged in, the keyboard acts as a device controller.

The USB Keyboard is not an interrupt-based device, rather the host must poll the keyboard and send requests to receive the scancode in return. For this, the team must write functions to perform input/output reading and writing. These functions must be used in conjunction with the system verilog hardware that is synthesized to retrieve data and write data.

The two functions that were written by the team were in ”usb.c” and ”io_handler.c” were pertaining to the reading and writing of data to and from the USB device controller (keyboard).

io_handler.c:

- void IO_init(void)
- void IO_write(alt_u8 Address, alt_u16 Data)

```
void IO_write(alt_u8 Address, alt_u16
    Data)
{
   *otg_hpi_address = Address;
   *otg_hpi_cs = 0;
   *otg_hpi_w = 0;
   *otg_hpi_data = Data;
   *otg_hpi_w = 1;
   *otg_hpi_cs = 1;
}
```

This function takes in parameters as the Address and Data to be written. It then sets the appropriate bits on the signal lines and sends the Data to the otg_hpi_data variable to be written to the usb hardware.

```
alt_u16 IO_read(alt_u8 Address)
{
   alt_u16 temp;
   //printf("%x\n",temp);
   *otg_hpi_address = Address;
   *otg_hpi_cs = 0;
   *otg_hpi_r = 0;
   temp = *otg_hpi_data;
   *otg_hpi_r = 1;
   *otg_hpi_cs = 1;
   return temp;
}
```

This function performs similar actions to the IO_write function in that it must read from a particular address on the USB hardware. The team had to use a temp variable to read it properly, set the appropriate signals, then send the data from otg_hpi_data into the temp variable which is then returned to the user by the function.

usb.c:

- void UsbWrite(alt_u16 Address, alt_u16 Data)
- alt_u16 UsbRead(alt_u16 Address)

```
void UsbWrite(alt_u16 Address, alt_u16
    Data)
{
  IO_write(HPI_ADDR, Address);
  IO_write(HPI_DATA, Data);
}
```

This function utilizes the two functions written in the previous section (in io_handler.c) and simply writes the particular address and the data in the parameters.

```
alt_u16 UsbRead(alt_u16 Address)
{
  IO_write(HPI_ADDR, Address);
  alt_u16 temp = IO_read(HPI_DATA);
  return temp;
}
```

This function uses the two functions written in io_handler.c to perform a read on the USB hardware. it first must write the address to HPI_ADDR to receive the proper address, then the temp variable must store the result of IO_read from HPI_Data.

V. Schematic/Block Diagram

The Schematic / Block Diagrams for this lab can be found in the Figures section of this document ("Section XI: Figures"). The Top Level Module Diagram can be found in Figure 2, the Ball Module Diagram can be found in Figure 3, the VGA Module Diagram can be found in Figure 4, and the Color Mapper Module can be found in Figure 5.

VI. Post Lab

Resource	Value
LUT	2663
DSP	10
Memory (BRAM)	82,944
Flip-Flop	646
Frequency	132.89 MHz
Static Power	102.09 mW
Dynamic Power	25.86 mW
Total Power	203.64 mW

TABLE I: Design Statistics

1. What is the difference between VGA_clk and Clk?

*Answer:*The VGA Clock runs at 25MHz to change how wide the individual pixels are, whereas the Clk onboard the processor runs at 50MHz (and does not affect the pixel size).

2. In the file io_handler.h, why is it that the otg_hpi_address is defined as an integer pointer while the otg_hpi_r is defined as a char pointer?

Answer: otg_hpi_r is a single bit and does not need to be declared as an int (16 bits - it would be a waste of space), whereas otg_hpi_address is multiple bits wide and must have multiple bits available to use.

3.What are the advantages and/or disadvantages of using a USB interface over PS/2 interface to connect to the keyboard? List any two.

Answer: PS/2 keyboards aren't polled, but are completely interrupt based. This allows the processor to complete tasks while waiting. Drivers for PS/2 are much simpler than USB keyboard drivers. Another disadvantage is that the USB keyboard only takes in 6 keys every message, so this is limited compared to the PS/2. These are two disadvantages for using a USB keyboard over PS/2.

4. Note that Ball_Y_Motion in the above statement may have been changed at the same clock edge that is causing the assignment of Ball_Y_pos. Will the new value of Ball_Y_Motion be used, or the old? How will this impact behavior of the ball during a bounce, and how might that interact with a response to a keypress? Can you fix it?

Answer: The new value of Ball_Y_Motion that would be used would be the old one and this would make it so that the ball bounces one clock cycle after the keypress is handled. A fix would be to put the assignment inside of the section of code that bounces the ball so that there is no delay in processing.

VII. Conclusion

For experiment 8, the team was successfully able to complete all of the tasks. The only issue came about in properly writing the C code to interface with the USB drivers. Setting the proper ports in each read/write method was causing errors if done incorrectly. A significant amount of time was spent debugging this interface code.

This lab will prove very important for the final design project, especially if the project involves interfacing with a keyboard input and monitor output.

VIII. Figures

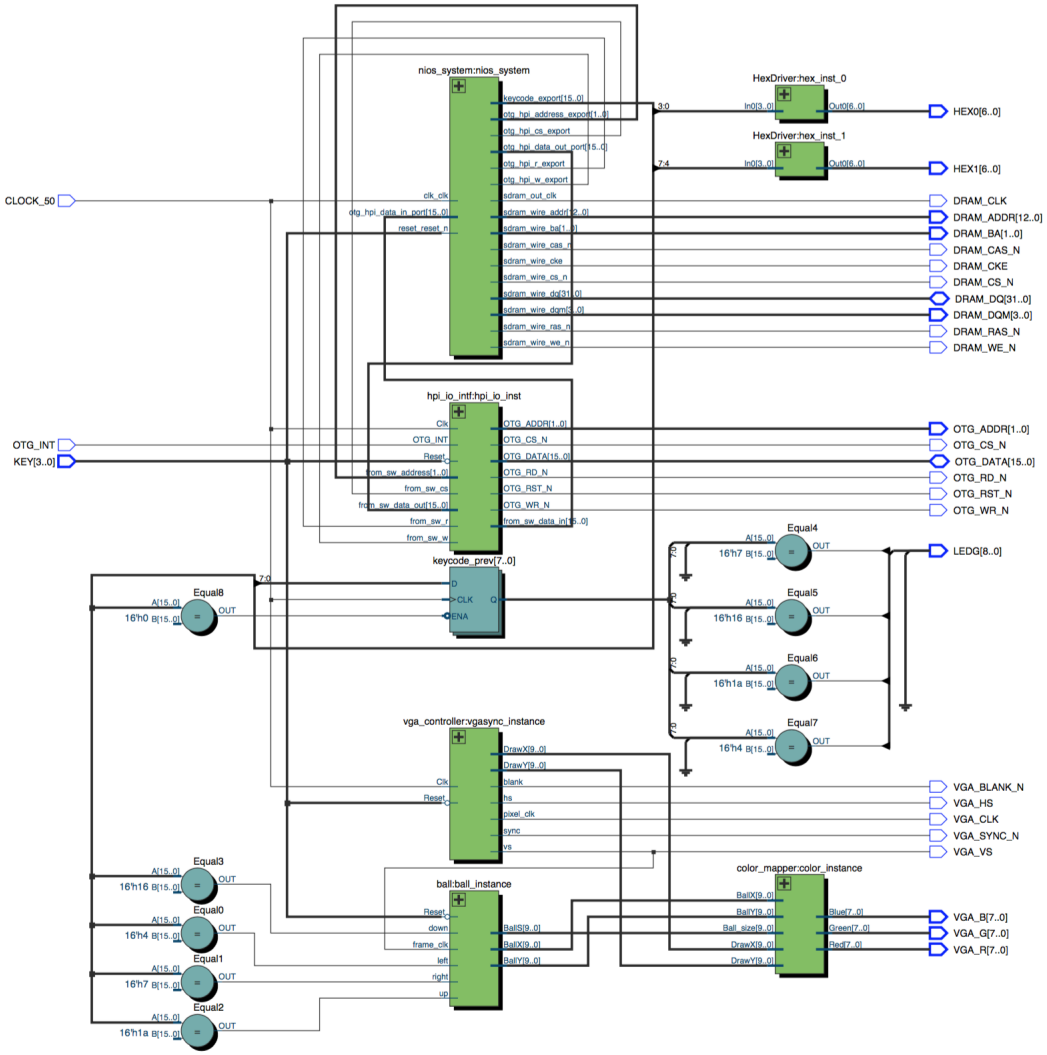

Fig. 2: Lab 8 Top Level SV

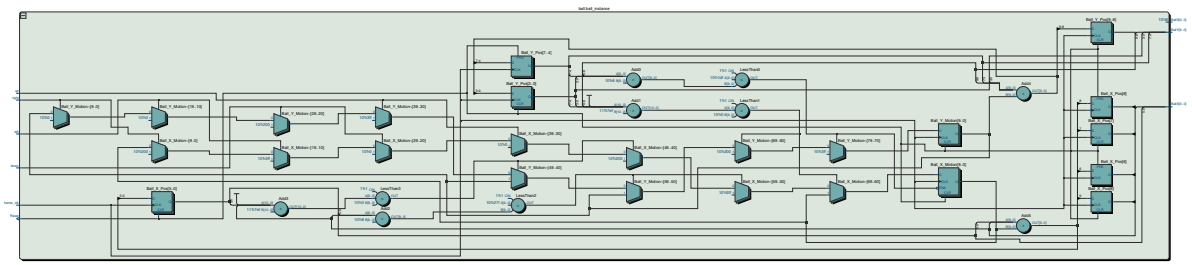

Fig. 3: Ball Circuit Diagram

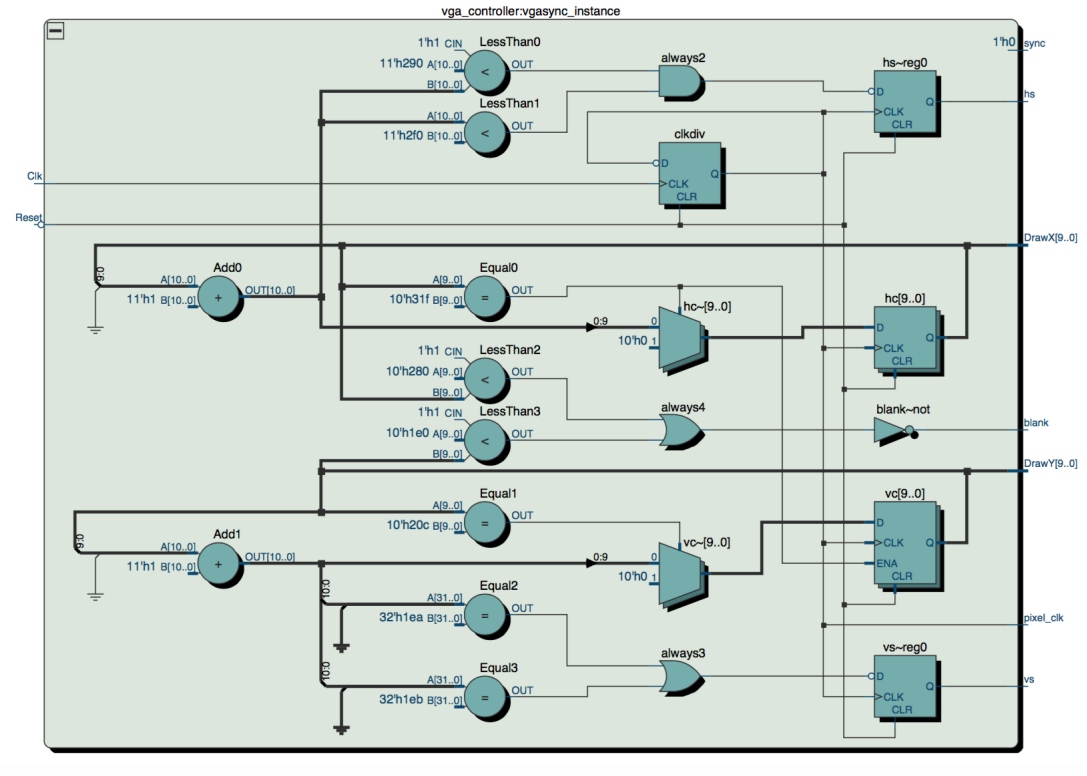

Fig. 4: VGA Circuit Diagram

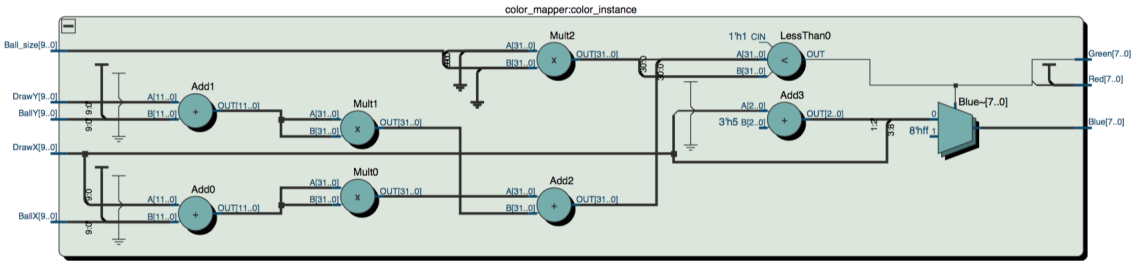

Fig. 5: Color Mapper Circuit Diagram

APPENDIX