ECE 385

Fall 2020

Experiment #8

SOC with USB and VGA interface in SV

Name: Xiao Shuhong & Lu Yicheng

Lab Section: D225

1, introduction

1.1 Briefly summarize the operation of the USB/VGA interface

For USB interface:
We construct lots of PIO in the nios system. C program can visit registers in these PIO though their base address. That is: we can write software to “give or receive” the value stored in these PIO.

These PIO are connected to hpi io interface. That is: Value in PIO will “give to or receive from” the corresponding registers in hpi io interface.

In hpi io interface, some registers control ez-otg as control signals. Some can receive data from ez-otg.

So, by the above procedure, FPGA can communicate with ez-otg by running C code. To be specific, we can just assume C code can directly access four HPI registers in ez-otg.

```
IO_write:
        Assign address to otg_hpi_address(give to PIO first, will choose which HPI register)
        Give some open signals to ez-otg(give to PIO first, eventually to ez-otg)
        Assign Data to otg_hpi_data(give to PIO first, eventually to the chosen HPI register)
        Give some close signals to ez-otg(give to PIO first, eventually to ez-otg)

IO_read:
        Assign address to otg_hpi_address(give to PIO first, will choose which HPI register)
        Give some open signals to ez-otg(give to PIO first, eventually to ez-otg)
        Read Data from otg_hpi_data(from the chosen HPI register)
        Give some close signals to ez-otg(give to PIO first, eventually to ez-otg)

UsbWrite:
        Assign address to HPI ADDRESS(using IO_write )
        Assign Data to HPI DATA(using IO_write)

UsbRead:
```

```
Assign address to HPI ADDRESS(using IO_write)

Read Data from HPI DATA(IO_read)
```

For VGA interface:
VGA stands for video graphics array. It is the port which we use to display the picture to the computer screen. Imagine we have an electron beam. This beam starts at the top left corner of screen and will travels from left to right, top to bottom of the screen. So, this beam can access every pixel and finally show the image. Notice that beam have to move from the bottom right corner to top left corner. It takes time so that we need Vertical Sync signal. With this signal, we will stop moving the ball for a certain time to wait until beam move to its right position.

2, Written Description of Lab 8 System

2.1 Written Description of the entire Lab 8 system

In lab 8, we will have a single ball on screen. This ball will opposite its velocity if it touches the wall. Also, we can use keyboard to control the direction of the ball. That is: W (up), A (left), S (down), D (right).

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

STEP1: NIOS change or get the PIO registers' value through visiting the base address.

STEP2: Value in PIO will "give to or receive from" the corresponding registers in hpi io interface.

STEP3: In hpi io interface, some registers control ez-otg as control signals. Some can receive data from ez-otg.
For examples:

OTG_RD_N register controls when ez-otg should "give expected data to from_sw_data_in"

OTG_ADDR register controls which HPI register should be access.

from_sw_data_in register will sometimes receive data from OTG_DATA.

STEP4: OTG_ADDR in hpi io interface tells which HPI register to choose. OTG_DATA tells what data need to write into the chosen register(or HPI Data gives value to OTG_DATA).

So, by the above procedure, FPGA can communicate with ez-otg by running C code. After our CPU get the keycode, it will send to the ball_instance. ball_instance will use the keycode to know the direction of the ball and decide whether current pixel is in the ball. ball_instance will give the “is ball” signal to color_mapper, color_mapper will generate proper RGB signal base on the position of pixel and is ball” signal. Those RGB signal is directly connect with VGA.

2.3 written description of the CY7 to host protocol

We have four HPI registers. They are:

HPI DATA: contain the data that may wait to be read by CPU or may wait to be write to the expected address in ez-otg.

HPI ADDRESS: contain the address. Data in that address may be “give “to HPI DATA or may be overwritten by data in HPI DATA.

HPI MAILBOX: The HPI Mailbox register provides a common mailbox between the CY7C67200 and the external host processor.

HPI STATUS: The HPI Status Port provides the external host processor with the MailBox status bits plus several SIE status bits. This register is not accessible from the on-chip CPU.

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

For USBRead:

With this function, CPU can read ez-otg’s ram. In other word, CPU just need to know the address, and data in that address will just give to CPU and store in its own memory.

For USBWrite:

With this function, CPU can write to ez-otg’s ram.

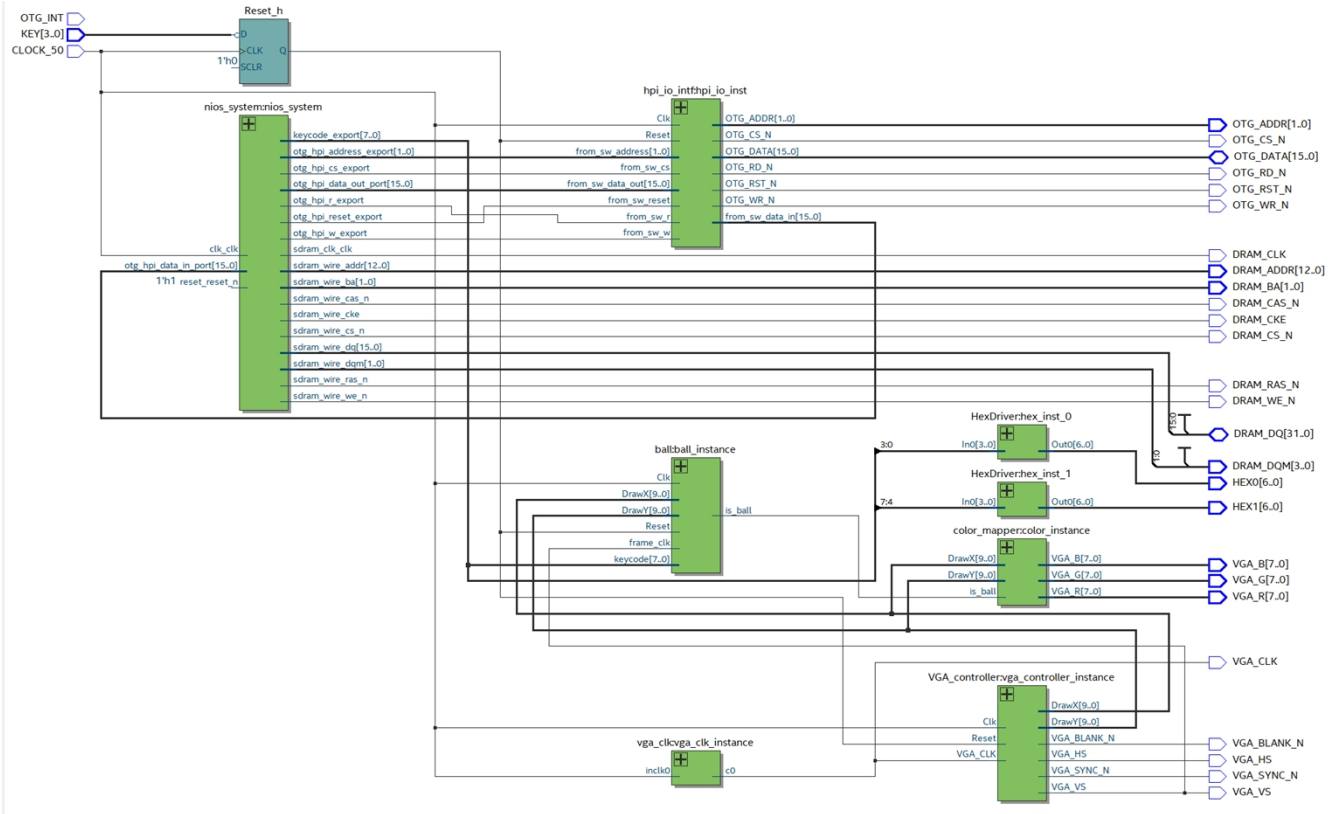
For IO_read:

With this function, CPU can read from HPI registers.

For IO_write:

With this function, CPU can write to HPI registers.

3 Block diagram



4 Module descriptions

module VGA_controller

```
(input              clk,
                    Reset,
 output logic       VGA_HS,
                    VGA_VS,
 input              VGA_CLK,
 output logic       VGA_BLANK_N,
                    VGA_SYNC_N,

 output logic [9:0] DrawX,
                    DrawY
 );
```

Description:

In this module, the current position of pixel is start at the top left corner of screen(0,0) and will travels from left to right, top to bottom of the screen. Also, VGA_VS will set to 0 when current position of pixel is in some specific range.

Purpose:

This module determines two things:

1,current pixel we are looking at.

2, proper frame clock

module ball

```
( input          clk,
                 Reset,
                 frame_clk,
  input [9:0]    DrawX, DrawY,
  output logic   is_ball,
  input logic [7:0] keycode
 );
```

Description:

This module will simulate the movement of the ball. To be specific, ball will opposite its velocity if it touches the wall. And receive keycode signal to determine the direction of the ball. That is: W (up), A (left), S (down), D (right).

Also, this module compares the (distance between center of the ball and pixel) with R to decide whether the current pixel is in the ball or not.

Purpose:

With this module, we can know whether the current pixel is in the ball or not.

module color_mapper

```
( input              is_ball,

  input        [9:0] DrawX, DrawY,
  output logic [7:0] VGA_R, VGA_G, VGA_B
);
```

Description:

This module will give different color base on with current pixel is in the ball or not. Also, color_mapper will give Background a nice color gradient.

Purpose:

Output the RGB signals for the current pixel.

module HexDriver

```
r (input  [3:0]  In0,
   output logic [6:0]  Out0);
```

Description:

Turn four bits data to the 7 bits control signals that control the 7-segment display.

Purpose:

display information on the FPGA board

module hpi_io_intf

```
F( 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 is a bridge between the CPU and EZ-OTG.

It also let OTG_DATA be high Z (tristated) when NIOS is not writing to OTG_DATA inout bus.

Purpose:

To avoid glitch. Act as a bridge between the CPU and EZ-OTG.

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 virtical 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 Write
                      OTG_WR_N,     //CY7C67200 Read
                      OTG_RST_N,    //CY7C67200 Reset
  input               OTG_INT,      //CY7C67200 Interrupt
  // 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 Mast 4 Bits
  output logic        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
      );
```

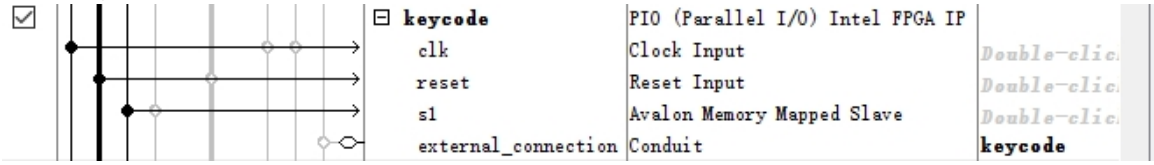
Description:

This is the top level. it connects the individual components. It contains hpi_io_inst, nios_system, vga_clk_instance, vga_controller_instance, ball_instance, color_instance and HexDriver

Purpose:

Connect the individual components

PIO keycode



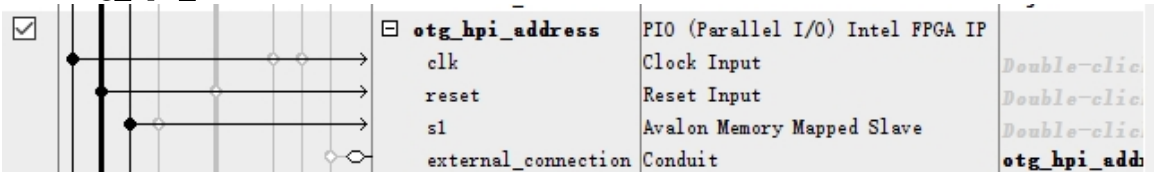
Description:

8 bits width. CPU send data to keycode. Keycode will then “give” data to ball instance.

Purpose:

To offer a base address to CPU, so that CPU can send keyboard information to this PIO and finally to ball instance.

PIO otg_hpi_address



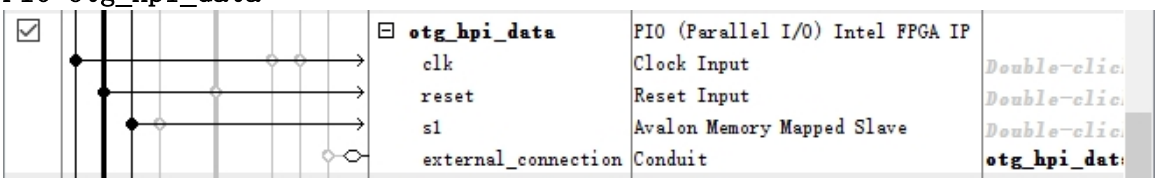
Description:

2 bits width. CPU send in address formation to otg_hpi_address. otg_hpi_address will then “give” address formation to hpi_io_inst. This information is to decide which HPI register we want to choose.

Purpose:

To offer a base address to CPU, so that CPU can send address information to this PIO and then to hpi_io_inst.

PIO otg_hpi_data



Description:

16 bits width.

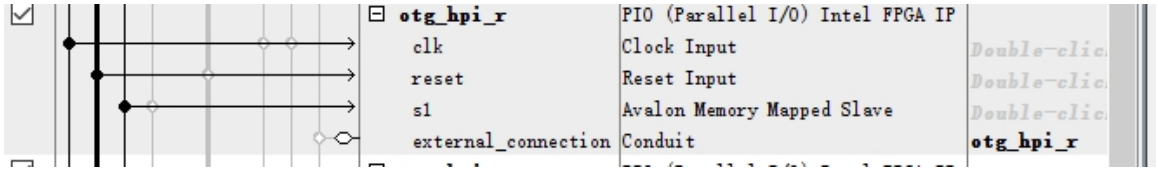
1,CPU send in data to otg_hpi_data. otg_hpi_data will then “give” data to hpi_io_inst.

2, otg_hpi_data receive data from hpi_io_inst, then CPU may read otg_hpi_data.

Purpose:

To offer a base address to CPU, so that CPU can send address information to this PIO and then to hpi_io_inst or read this PIO to get data from hpi_io_inst.

PIO otg_hpi_r



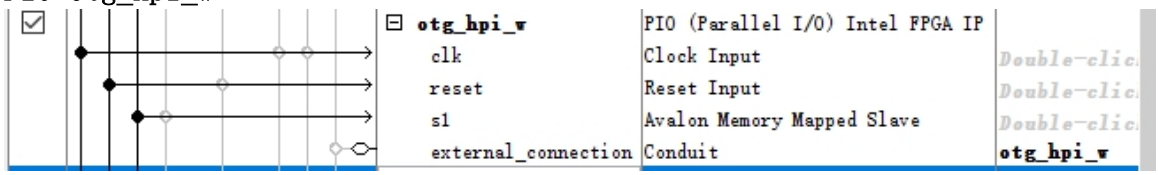
Description:

1-bit width. This is the control signal that controls ez-otg.

Purpose:

To offer a base address to CPU, so that CPU can control ez-otg by just writing information to this PIO.

PIO otg_hpi_w



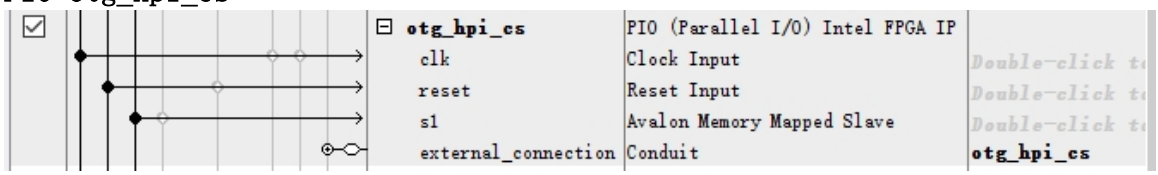
Description:

1-bit width. This is the control signal that controls ez-otg.

Purpose:

To offer a base address to CPU, so that CPU can control ez-otg by just writing information to this PIO.

PIO otg_hpi_cs



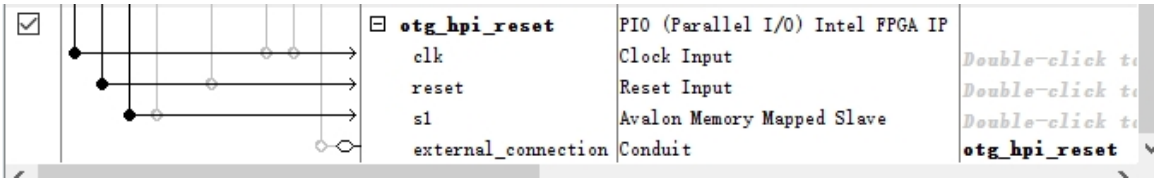
Description:

1-bit width. This is the control signal that controls ez-otg.

Purpose:

To offer a base address to CPU, so that CPU can control ez-otg by just writing information to this PIO.

PIO otg_hpi_reset



Description:

1-bit width. This is the control signal that controls ez-otg.

Purpose:

To offer a base address to CPU, so that CPU can control ez-otg by just writing information to this PIO.

5. Answer to Post Lab Question

5.1. design statistics table

LUT	2728
DSP	10
Memory (BRAM)	55296
Flip-Flop	2075
Frequency	139.74MHZ
Static Power	105.15mW
Dynamic Power	0.75mW
Total Power	176.34mW

5.2. problems

5.2.1. Difference between VGA_CLK and Clk

VGA_CLK is defined for cpu, which we set at about 25MHz, this is used as the pixel clock to generate the sync pulse, each pixel on display will be generated by this VGA_CLK.

Clk is defined for all hardware component on board, which we set 50MHz just as what we do before, the signal flow through hardware will base on this clk.

5.2.2. 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?

an integer here takes 16 bits while a char only 8 bits, for otg_hpi_data which used as datapath, as our data are 16-bit wide, so we need use a int. For otg_hpi_r, as we only should use 1 bit, so use an 8-bit construct char is enough and can save some space compare with a int.

5.3. Hidden Question

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

disadvantage:
1. using PS/2 will give much faster response than USB, as PS/2 is directly connecting with MOBO while USB use internal BUS to communicate with MOBO
2. USB is not so sage as PS/2, as we are free to send files over USB while PS/2 do not allow it. So, for some unsafe USB device, we may meet problems.

advantage:
1. prepare connection on USB is easier and faster than PS/2. For USB, if we want to connect some device in like keyboard, we just plug it in our board, and it will be detected, and we can use. While for PS/s, we need to first shout down our system and plug device in, then open system which take much more time.
2. USB can accept much more different kinds of device than PS/2 while PS/2 only support the basic kinds like mouse and keyboard.

5.3.2. How will this impact behavior of the ball during a bounce, and how might that interact with a response to a keypress?

The old value of Ball_Y_Motion is used, for all this in always_ff block, values update at rising edge and with <=, we get all assignment at the same time.

Ball_Y_Motion_in is the next step value of Ball_Y_Motion, in always_comb block, lines are achieved sequentially, we should assign the current state Ball_Y_Pos_in with current state value Ball_Y_Motion. If we use Ball_Y_Motion_in, we will get exactly one frame delay when bouncing, and the ball will just move into the wall on display.

6. Conclusion

6.1. Functionality of my design

This lab moves much better than the last one (lab 7) as we have gotten much more idea about both Avalon and software. One struggle thing is not about lab itself but the vga-to-hdmi wire is not work when

display and we just not get the idea that it is the problem of the wire but not our code, and we fight with the nonexistent bug for a lot of time.

6.2. Comments

Still, consider it is our first time to deal with USB and display, meaning of some steps on the manual is not so clear, it is better to have more description.