ECE 385

Spring 2024

Experiment # 8

Lab8 : SoC with USB and VGA Interface in SystemVerilog

Name: Jie Wang, Shitian Yang

Student ID: 3200112404, 3200112415

Prof. Chushan Li,
Prof. Zuofu Cheng
ZJU-UIUC Institute
April 15, 2024, Sunday D-225
TA: Jiebang Xia
Demo Point: 5/5

1.Introduction

In this lab, we need to connect a USB keyboard and a VGA display with the DE2 board using SystemVerilog. After writing the hardware tcl interface connecting the monitor to the VGA port and the keyboard to the USB port, we should control a small ball to move and bounce on the monitor screen based on the keyboard inputs .

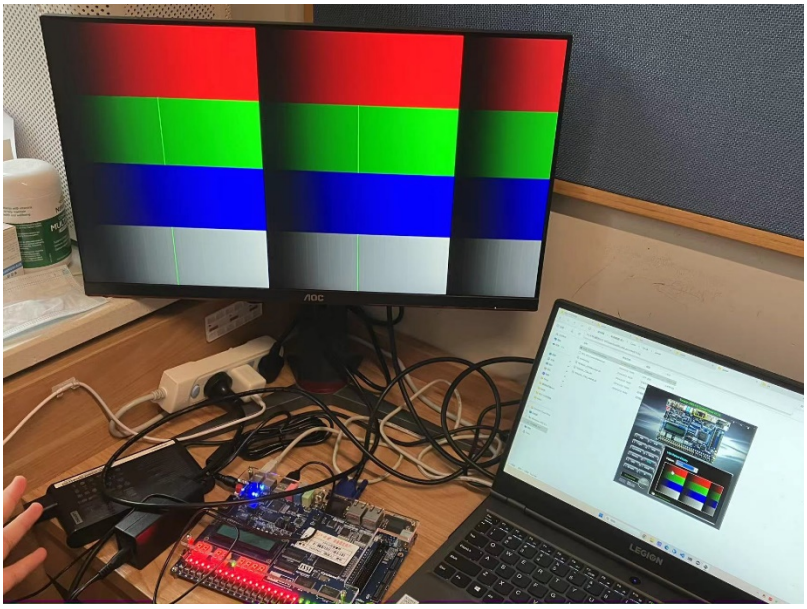

Fig-1: a test setup with an FPGA board connected to a laptop and our monitor displaying a color bar pattern with a noticeable light green line segment error.

2. System Operation

USB: The system uses the CY7C67200 USB controller for interfacing with the USB keyboard. The Nios II processor handles the USB protocol in software, managing data transfers between the keyboard and the FPGA. The USB protocol involves writing an address to the CY7C67200 to either send or receive data. This is accomplished through functions like USBRead and USBWrite.

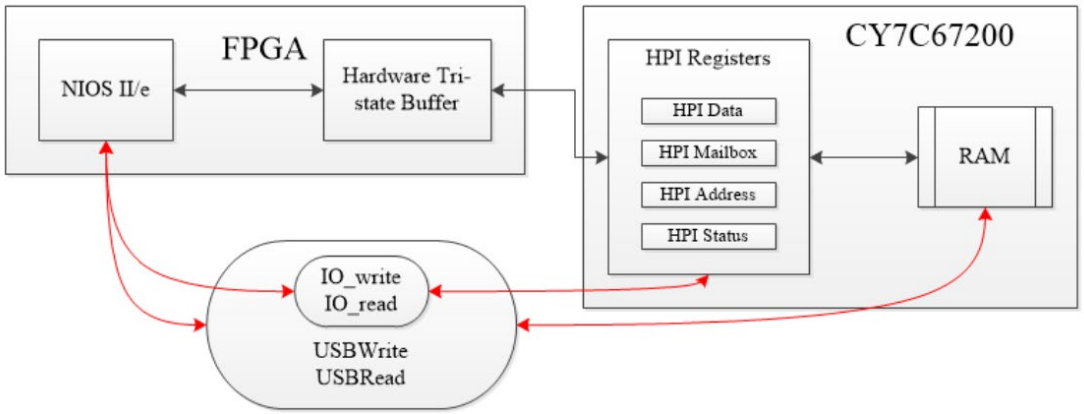

Fig-2: Hardware Tri-state Buffer in hpi_io_intf.sv

VGA: The VGA interface uses a controller module that generates synchronization signals compatible with the VGA standards. A 25 MHz clock signal from the FPGA is utilized to maintain

a refresh rate of 60 Hz for smooth motion. The display logic involves a color mapper module, which decides the pixel colors based on the ball's position, updating the display to show the ball's movement across the screen.

Operation Flow: Upon system start-up, the ball is initially positioned at the center of the screen. As inputs are received from the USB keyboard, the system interprets these to adjust the ball's direction. The Nios II processor plays a crucial role here, interfacing directly with the USB controller to fetch new input data. The VGA controller continuously updates the display based on the new positions calculated for the ball. This involves adjusting the ball's trajectory when it encounters the screen's boundaries to simulate bouncing.

3. Module Description

This part is included in Appendix A.

4. Block diagram

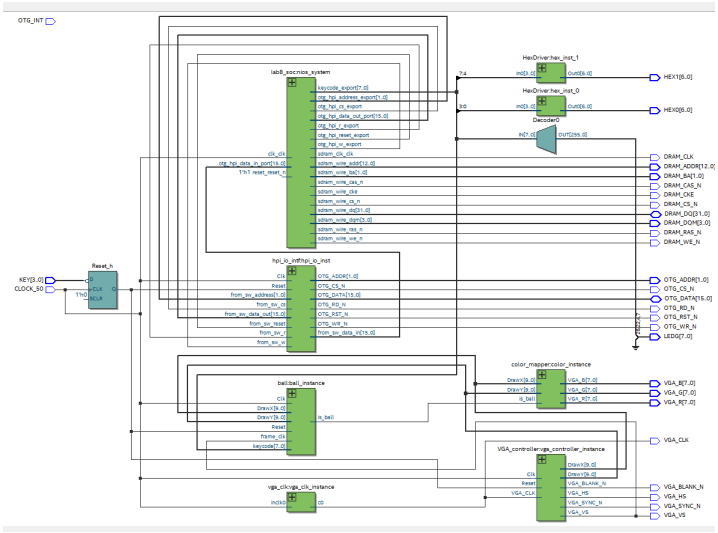

Fig-3: RTL Viewer of the Lab8 SV Module

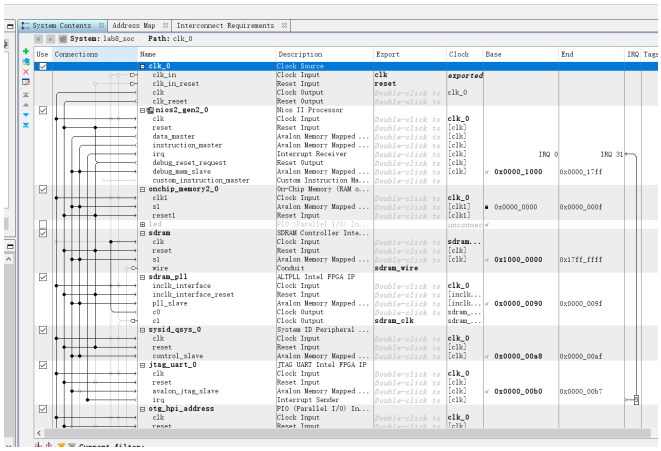

Fig-4: Our Connection in lab8_soc, notice the base address affects the keyboard mapping

5.Post-lab Questions

1. Design Resources and Statistics

Resource	Utilization
LUT (Logic Elements)	2,694
DSP (Embedded Multiplier 9-bit elements)	10
Memory (BRAM bits)	11134
Flip-Flop	2220
Frequency	133.89MHz
Static Power	103.03mW
Dynamic Power	0.98mW
Total Power	199.12mW

Table 3: System Information, I think it does not include the SoC computation

2. What is the difference between VGA_CLK and Clk?

VGA_CLK is 25MHz while Clk is 50 MHz, which makes sure VGA port can be updated at 60Hz.

3.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?

To save the memory. Otg_hpi_data need much larger data memory(32 bits) which is same as the space of int. While Otg_hip_r is just needed to control the register and 4 bit is enough which is same as the space of char.

6.Hidden Question

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. Give an answer in your Post-Lab.

Hot-swapping Capability: USB interfaces support hot swapping, allowing external devices to be used immediately upon connection without the need to restart the computer.

Limitations on Simultaneous Key Presses: With a USB interface, only up to six keys can be pressed simultaneously because only 8 bytes of data can be transferred in one data packet. This means a USB keyboard can only register the states of six keys at the same time, potentially leading to "key rollover" issues. In contrast, the PS/2 interface does not have this limitation, allowing for more complex key combinations without conflict.

3. Does the *"Ball_Y_Pos"* update use the new or old *"Ball_Y_Motion"* value, and how do *"Ball_Y_Pos + Ball_Y_Motion" and "Ball_Y_Pos + Ball_Y_Motion_in"* differ in ball bounce behavior and keypress interaction?

When updating *Ball_Y_Pos*, the old value of *Ball_Y_Motion* is used due to parallel assignments.

- If we use *Ball_Y_Motion,* the Y position in the current clock cycle is determined by the old Y position and the old motion in the Y direction. This will not be influenced by previous key presses or whether the ball is at the boundary.
- If we use *Ball_Y_Motion_in*, the Y position in the current clock cycle is determined by the old Y position, previous key press actions, and the boundary condition.

Using *Ball_Y_Motion_in* results in a quicker response of the ball. When the ball reaches the wall, it will bounce back immediately, and when a key press occurs, the direction will change immediately. Using *Ball_Y_Motion* does not achieve this as the reaction of the picture is one frame slower than the code.

7. Conclusion

In this lab, we successfully interfaced a USB keyboard and VGA display with the DE2 board using SystemVerilog. We utilized the CY7C67200 USB controller and a custom VGA controller module, managed by the Nios II processor, to enable dynamic ball movement on the display based on keyboard inputs.

The hard part is the display synchronization and input responsiveness, achieving smooth motion and accurate bounce mechanics on the VGA screen. This experiment enhanced our skills in SystemVerilog and FPGA programming, emphasizing the integration of hardware and software components in real-time systems, preparing us for the future final project design. For example, we could write a Gluttonous Snake game easily based on it.

8. References

[1] KTTECH. (2017, January 31). ECE 385 Lab 8: SoC with USB and VGA Interface in SystemVerilog. Retrieved from https://kttechnology.wordpress.com/2017/03/10/ece-385lab-8-soc-with-usb-and-vga-interface-in-systemverilog/. Teaching Assistant Blog
[2] ECE385 Faculty. (n.d.). Lab 8 description
[3] ECE385 Faculty. (n.d.). Introduction to SystemVerilog (pdf)
[4] ECE385 Faculty. (n.d.). Introduction to Quartus Prime in the lab manual.

9. Appendix

VGA Interface

Module Name:ball

Inputs:
Clk (50 MHz clock)
Reset (Active-high reset signal)
frame_clk (The clock indicating a new frame, approximately 60 Hz)
DrawX (Current X coordinate of the pixel)
DrawY (Current Y coordinate of the pixel)
keypress (The current key pressed, new input)

Output:
is_ball (Indicates whether the current pixel belongs to the ball or the background)

Description:
The ball module is responsible for computing the position and motion of a ball on a display based on keyboard inputs. It determines whether a specific pixel should be part of the ball based on its current calculated position.

Operation:
Upon a reset, the ball is positioned at the center of the display. The module updates the ball's position and motion based on key presses (W, A, S, D) which correspond to directions (up, left, down, right).
The motion updates occur at the rising edge of the frame clock. Each press of the directional keys updates the motion variables (Ball_X_Motion and Ball_Y_Motion), which then influence the ball's position in the next frame. Boundary checks are implemented to make the ball bounce off the edges of the display area.
For each pixel , the module calculates if it falls within the area defined by the ball's radius (Ball_Size).

Module Name: color_mapper

Inputs:
is_ball (Indicates whether the current pixel belongs to the ball or the background)
DrawX (Current X coordinate of the pixel)
DrawY (Current Y coordinate of the pixel)

Outputs:
VGA_R (Red component of the VGA output, 8 bits)
VGA_G (Green component of the VGA output, 8 bits)
VGA_B (Blue component of the VGA output, 8 bits)

Description:
The color_mapper module is designed to assign color values to pixels based on whether they are part of the ball or the background. It outputs RGB values to a VGA display, using different colors to distinguish between the ball and the background.

Operation:
If is_ball is true, the pixel is part of the ball, and the RGB output is set to white (255, 255, 255), rendering the ball in white color on the display.
If is_ball is false, the pixel belongs to the background. In this case, the color of the background is a gradient depending on the X-coordinate of the pixel. The red component is set to a constant value (63), the green component is set to 0, and the blue component decreases linearly with increasing X-coordinate.

Module Name: hpi_io_intf

Inputs:
Clk
Reset from_sw_address from_sw_data_out from_sw_r from_sw_w from_sw_cs from_sw_reset

Outputs:
from_sw_data_in OTG_ADDR OTG_RD_N OTG_WR_N OTG_CS_N OTG_RST_N
OTG_DATA

Description:
The hpi_io_intf module interfaces the Human Processor Interface (HPI) with the OTG (On-The-Go) device, handling signals and data transfer between a processor (like a NIOS II) and the OTG chip. It manages the read and write operations through an inout data bus (OTG_DATA), which requires careful signal control to ensure proper data handling and integrity.

Operation:
The module continuously monitors the control signals and the address signal . Based on these inputs, it manages the data flow to and from the OTG chip.
Data sent to the OTG chip is buffered in a register before being sent to the OTG_DATA bus, ensuring that the data bus is driven only during appropriate write operations.

The OTG_DATA bus is set to high impedance (tri-state) when not actively driven by the NIOS II processor to prevent bus conflicts and ensure proper read operations from the OTG chip.

Module Name: lab8

Inputs:

CLOCK_50 (50 MHz clock signal from the FPGA)
KEY[3:0] (Four-bit input for keys, where bit 0 is used as a reset)

Outputs:

HEX0 (7-segment display HEX0)
HEX1 (7-segment display HEX1)
VGA_R (8-bit output for VGA red channel)
VGA_G (8-bit output for VGA green channel)
VGA_B (8-bit output for VGA blue channel)
VGA_CLK (VGA clock signal)
VGA_SYNC_N (VGA synchronization signal)
VGA_BLANK_N (VGA blanking signal)
VGA_VS (VGA vertical synchronization signal)
VGA_HS (VGA horizontal synchronization signal)
OTG_ADDR (2-bit output for CY7C67200 USB controller address)
OTG_CS_N (Chip select for CY7C67200, active low)
OTG_RD_N (Read signal for CY7C67200, active low)
OTG_WR_N (Write signal for CY7C67200, active low)
OTG_RST_N (Reset for CY7C67200, active low)
DRAM_ADDR (13-bit output for SDRAM address)
DRAM_BA (2-bit output for SDRAM bank address)
DRAM_DQM (4-bit output for SDRAM data mask)
DRAM_RAS_N (SDRAM row address strobe, active low)
DRAM_CAS_N (SDRAM column address strobe, active low)
DRAM_CKE (SDRAM clock enable)
DRAM_WE_N (SDRAM write enable, active low)
DRAM_CS_N (SDRAM chip select, active low)
DRAM_CLK (SDRAM clock)

Inout:

OTG_DATA (16-bit bidirectional data bus for CY7C67200 USB controller)
DRAM_DQ (32-bit bidirectional data bus for SDRAM)

Description:

The lab8 module serves as the main integration point for the Lab 8 project in the ECE 385 course. It connects the FPGA to various peripherals including a VGA display and a USB controller, and interfaces with SDRAM for data storage.

Operation:
The module initializes and manages the signals for interfacing with a VGA display and the CY7C67200 USB controller. It also handles communication with SDRAM utilized by the Nios II processor for data storage and retrieval.
The VGA_controller, ball, color_mapper, and HexDriver modules are instantiated to manage the display and interaction elements of the project. Key signals and data are routed between the FPGA and the peripherals, ensuring synchronized operations for display and USB communication.

Module Name: VGA_controller

Inputs:
Clk (50 MHz clock)
Reset (Active-high reset signal)
VGA_CLK (25 MHz VGA clock input, used for timing critical operations)

Outputs:
VGA_HS (Horizontal sync pulse, active low)
VGA_VS (Vertical sync pulse, active low)
VGA_BLANK_N (Blanking interval indicator, active low)
VGA_SYNC_N (Composite Sync signal, active low; not used in this lab but required for hardware)
DrawX (Horizontal coordinate output, 10 bits)
DrawY (Vertical coordinate output, 10 bits)

Description:
The VGA_controller module manages the generation of timing and synchronization signals for a VGA display operating at a resolution of 640x480 pixels. This includes generating horizontal and vertical sync pulses, managing the drawing coordinates for pixels on the display, and controlling the timing for when the display is active or in a blanking interval.
Operation:
The module operates by counting pixels (h_counter) and lines (v_counter) to manage the synchronization pulses and the active display time for a standard VGA display. These counters are incremented on every positive edge of the VGA clock (VGA_CLK), which should be 25 MHz.
Horizontal Timing: The horizontal sync pulse (VGA_HS) is generated by checking the h_counter value and is active low for 96 clock cycles between 656 and 752.
Vertical Timing: The vertical sync pulse (VGA_VS) is active low for 2 lines, specifically when v_counter is between 490 and 492.
Blanking Interval: The VGA_BLANK_N signal is used to indicate when the VGA output should be blanked (not displaying pixels). It is active when both h_counter and v_counter are within the displayable area (0-639 for horizontal and 0-479 for vertical).

USB protocol

USBRead: USBRead is used to retrieve data from the registers in the CY7C67200 USB Controller. It employs IO_Write to set the desired address and then utilizes IO_Read to fetch and return the data stored at that address.

USBWrite: USBWrite facilitates writing data to the internal registers of the USB controller. It achieves this by calling IO_Write twice: once to specify the register address and a second time to write the data to that address.

IO_Read: IO_Read returns 16-bit data from the otg_hpi_data register after updating the otg_hpi_address to the specified 8-bit input address. This function is critical for reading operations where precise control over the USB controller's address space is necessary.

IO_Write: IO_Write sends data to the USB Controller registers. It takes two inputs, address and data, and writes them respectively to the address and data pointers in the USB controller.