ECE 385

2021 FA

LAB 8
SOC with USB and VGA Interface in SystemVerilog

Zhou Qishen, Xie Mu

Introduction

In Lab 8 we implement a USB input VGA output hardware system on FPGA board then run a bounced ball program on it and control it with USB keyboard. The FPGA board can display the scan code of the pressed key and use WASD to control the ball's moving direction.

Description & Diagram

Hardware Description

The entire lab8 system require us to poll data from USB buffer, then process it with C codes at the NIOS 2 processor, then display the result to the VGA monitor. We need to set up the drivers that communicate with the OTG chip and create an interface by connected the top-level program to the given VGA driver that we can output the data to the VGA driver. Note that the two drivers are running at different clocks to make sure that the screen's refresh rate is at 60HZ.

To connect the NIOS 2 to the OTG chip, we need to build a JTAG interface, and the interface is done through memory mapped Parallel Input Output module, and the software interface is given by C code. When we want to read data from the OTG chip, we can simply call function UsbRead, and if we want to write data to the chip, we can call UsbWrite. These function work directly on the register on the OTG chip, and through this we can fetch the input scan code to the processor to control the program.

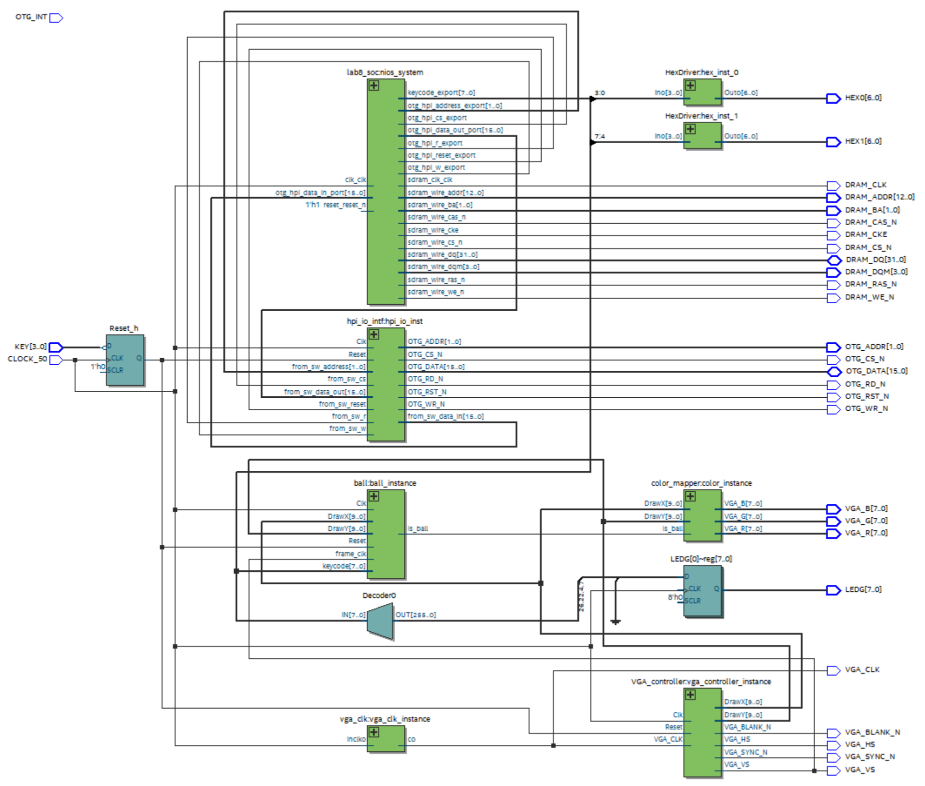
To connect the NIOS 2 to the VGA monitor, we use VGA controller and the color mapper. The VGA controller module receive the current pixels' data and send it to the controller, and the color mapper actually writes the pixel's color based on the RGB data to the monitor. As a result, the NOIS 2 process the BALL module and update the current pixel position to the VGA controller and the color mapper.

The CY7 chip interact with a host protocol through the software running on the NIOS 2 processor. The software stores the hardware address of the registers. Then the software can help transport the data from the CY7 ram to the Processor.

USB protocol

The IO_Write function writes the value given to the targeted address to the OTG chip. The IO_Read read values and then return it from the address we given. The USB Write function help write the data from the NIOS processor to the OTG chip. With the previous function, we can use IO_Write function to write address to the HPI_ADDR register, and write the desire data to the HPI_DATA register, then the OTG chip will write the data to the target address. The USB read function write the address to the HPI_ADDR with IO_Write, then it uses IO_Read to fetch data in the HPI_DATA.

System Level Block Diagram



Block Diagram

Hardware Module Description

Module: sdram
Input: clk, reset, s1
Output: wire
Description: This module is a 32-bit dynamic memory of SOC, which contains 1 chip, 4 banks, 13 rows and 10 columns, with the size of 1GB data.
Purpose: It is used for store data and address.

Module: onchip_memory2_0
Input: clk, reset, s1, wire
Output: readdata
Description: This module is writable RAM with memory size of 16 byte and datawidth of 32.
Purpose: This module is the on chip memory of NOIS II processor.

Module: led

Input: clk, reset, s1
Output: external_connection
Description: The module of LED on DE2 board with 8-bit datawidth.
Purpose: This module is used for control the LED on board to output data.

Module: sdram_pll
Input: inclk_interface, inclk_interface_reset, pll_slave, c0
Output: c1
Description: Clock module with the offset of -3 ns.
Purpose: This module is used for adjusting the clock input and I/O of sdram.

Module: sysid_qsys_0
Input: clk, reset, control_slave
Output: almost_empty, almost_full, empty, full, rd_data
Description: This module is system ID checker to ensure the compatibility between hardware and software.
Purpose: It can prevents us from loading software onto an FPGA which
has an incompatible NIOS II configuration.

Module: nios2_gen2_0
Input: clk, reset, data_master, instruction_master, irq, debug_deset_request, debug_mem_slave
Output: custrom_instruction_master
Description: The build-in NIOS II/e processor.
Purpose: The main processor of the SOC system.

Module: jtag_uart_0
Input: clk, reset, avalon_jtag_slave
Output: irq
Description: It is a build-in JTAG UART peripheral module.
Purpose: This module enable use the terminal of the host computer to communicate with the NIOS II

Module: keycode
Input: clk, reset, s1
Output: external_connection
Description: It is a 8 bit I/O module.
Purpose: This module is used to pass the look up table of key to otg chip.

Module: otg_hpi_data
Input: clk, reset, s1
Output: external_connection
Description: It is a 16-bit bidirectional PI/O module.
Purpose: This module is used to carry data between otg chip and NOIS.

Module: otg_hpi_address
Input: clk, reset, s1
Output: external_connection
Description: It is a 2-bit output PI/O module.
Purpose: This module is used to carry address between otg chip and NOIS.

Module: otg_hpi_r
Input: clk, reset, s1
Output: external_connection
Description: It is a 1-bit output PI/O module.
Purpose: This module is used to carry read instruction between otg chip and NOIS.

Module: otg_hpi_w
Input: clk, reset, s1
Output: external_connection
Description: It is a 1-bit output PI/O module.
Purpose: This module is used to carry write instruction between otg chip and NOIS.

Module: otg_hpi_reset
Input: clk, reset, s1
Output: external_connection
Description: It is a 1-bit output PI/O module.
Purpose: This module is used to carry reset instruction between otg chip and NOIS.

Post-Lab

What is the difference between VGA_CLK and Clk?

VGA clock is 25 MHz, which is the half of the clock of the SoC.

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 length of character is 1, which is quite smaller compared to the integer one with 4.

Document the Design Resources and Statistics in following table.

Property	Value
LUT	3621

DSP	0
Memory	11392
Flip-Flop	2223
Frequency	50 MHz
Static Power	105.16 mW
Dynamic Power	0.80 mW
Total Power	177.83 mW

Answer to two hidden questions

Q: What are the advantages and/or disadvantages of using a USB interface over PS/2 interface to connect to the keyboard?

A: The PS/2 have much better performance and higher power consume than USB since it is a bidirectional synchronous serial communication protocol, and it don't need host to connect to the processor, and the PS/2 don't need to poll the input, so it can theoretically accept infinite number of key presses, while the USB can only handle 6 key presses at the same time. However, the USB do not need a clock to synchronous the input, this means that it can hot swappable from the devices. The most important reason is that the USB is a Standard unified interface, that means it can connect different peripherals to the PC with single port.

Q2. 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?

A: The old value of the Ball_Y_Motion will stop being used when BALL_Y_POSITION is updated, instead, we use BALL_Y_MOTION_IN. the difference is that the former one will update the motion with delay, because it cannot update the motion in the same clock cycle. If the bounce happened, the ball will end up moving in the current direction and be given a opposite motion. If a key is pressed, the current motion will be overwritten by the input and the other direction's movement will be cleared.

Conclusion

The outcome of the lab is perfect. We can control the ball with the A S D and W, and display the current scan code of the key board on the HEX display. During the lab, we have trouble understanding what the JTAG and the HPI was doing and how the CY7 chip works to receive hardware signals. To solve this problem, we have to look up the document given by the Quartus and the provided files. At first our ball will move diagonally, and to solve this, we clear the movement in other direction when updating the movement. This will lead to the disappearance of the diagonal move of the ball. To solve the problem, we refer to the hidden question and finally add the vertical velocity killing command to the BALL_Y_MOTION_IN.

I think the given documents are not so direct for us student to look up, though they are authoritative source. I think in the tutorial the professor can add more explanation to the hard wares to help us understand it easier instead of given a lot of datasheet to us.