ECE 385

Fall 2016

Experiment #8

SOC with USB and VGA Interface in SystemVerilog

Shounak Ray, Kurt Gale

Section ABC

Zhenhong Liu, John Kim, Jack Krol

Introduction:-

The NIOS II processor is used to map the VGA controller and the CY7C67200 chips on the FPGA board to memory so software is able to interact with those components. This is used to receive keyboard inputs and to use those inputs to control the motion of a ball generated by the VGA controller.

Summary of Operation:-

The NIOS has access to five outer registers on the CY7C67200 chip. It writes to and reads from these registers in order to interact with the four internal registers of the chip, which allow it to read and write from the chip's memory. These five outer registers are otg_hpi_address, otg_hpi_data, otg_hpi_cs, otg_hpi_r, and otg_hpi_w. Otg_hpi_address is a two bit register, and is used to select between the internal registers. Otg_hpi_data is used to read data from the selected internal register or to write data to that selected internal register. Otg_hpi_cs is used to enable all interactions with the chip. Otg_hpi_r is used to tell the CY7 chip to put the data from the selected internal register into the otg_hpi_data register. Otg_hpi_w is used to tell the CY7 chip to put the data from the otg_hpi_data register into the selected internal register.

The NIOS only interacts with the VGA controller by passing in the keycode that it received from the CY7 chip. We use this keycode to determine the ball's motion and position, and use that information to tell the VGA what pixels to color on the screen.

The IO_read function involves a sequence of changing the five outward facing CY7 registers in order to access data from one of the four internal registers.
The IO_write function involves a sequence of changing the five outward facing CY7 registers in order to write data to one of the four internal registers.
The USBread function involves calling IO_write to change the internal address register on the CY7 and then calling IO_read to read from the internal data register, which holds the value from the CY7 chip's memory at the address we wrote into the internal address register.
The USBwrite function involves calling IO_write to change the internal address register on the CY7 and then calling IO_write to write to the internal data register, which will write that data into the CY7 chip's memory at the address we wrote into the internal address register.

Modules:-

Module: lab8.sv
Inputs: CLOCK_50, [3:0] KEY, OTG_INT
Outputs: [6:0] HEX0, HEX1, [7:0] VGA_R, [7:0] VGA_G, [7:0] VGA_B, VGA_CLK, VGA_SYNC_N, VGA_BLANK_N, VGA_VS, VGA_HS, OTG_ADDR, OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N, [12:0] DRAM_ADDR, [1:0] DRAM_BA, [3:0] DRAM_DQM, DRAM_RAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK
Inouts: [15:0] OTG_DATA, [31:0] DRAM_DQ

Description: This is our top level module.
Purpose: This module's job is to integrate the Nios II system with the rest of the hardware. This module instantiates the HPI interface module, the VGA interface module, the ball module, the color mapper module, the hex driver modules, and the NIOS system module.

Module: nios_system.sv
Inputs: clk_clk, [1:0] keys_wire_export, reset_reset_n, [17:0] switches_wire_export, [1:0] keys_wire_export, reset_reset_n, [15:0] otg_hpi_data_in_port
Outputs:[15:0] keycode_export, [1:0] otg_hpi_address_export, otg_hpi_cs_export, [15:0] otg_hpi_data_out_port, otg_hpi_r_export, otg_hpi_w_export, sdram_clk_clk, [12:0] sdram_wire_addr, [1:0] sdram_wire_ba, sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, [1:0] sdram_wire_dqm, sdram_wire_ras_n, sdram_wire_we_n
Inouts: [15:0] sdram_wire_dq
Description: This is the module generated by qsys.
Purpose: This module connects our System Verilog variables to memory so we can write to them in software. The PIO blocks added in the Qsys file were for the keycode, which was used for find the ASCII value of the key pressed, for the otg_hpi_address, otg_hpi_data, otg_hpi_r, otg_hpi_w and otg_hpi_cs. These PIO blocks were added to interact with USB chip, and they take in their connections from the HPI interface in the top level.

Module: hpi_io_intf.sv
Inputs: [1:0] from_sw_address, OTG_INT, Clk, Reset, [15:0] from_sw_data_out,
Outputs: [15:0] from_sw_data_in, [1:0] OTG_ADDR, OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N
Inouts: [15:0] OTG_DATA

Description: The HPI interface takes in inputs from the NIOS system module in the top level entity, and sets the registers when the appropriate values, which are again sent to the NIOS system module in the top level entity.

Module: ball.sv
Inputs: Reset, frame_clk, [15:0] keycode
Outputs: [9:0] BallX, [9:0] BallY, [9:0] BallS
Description: This module describes the motion the ball across the screen.

Module: Color_Mapper.sv
Inputs: [9:0] BallX, [9:0] BallY, [9:0] DrawX, [9:0] DrawY, [9:0] Ball_size
Outputs: [7:0] Red, [7:0] Green, [7:0] Blue
Description: This module sets up the circle pixels and their colors, using the given coordinates and size.

Module: vga_controller.sv
Inputs: Clk, Reset
Outputs: hs, vs, pixel_clk, blank, sync, [9:0] DrawX, [9:0] DrawY
Description: This module sets up the interface for the interaction between the FPGA and the VGA.

Module: HexDriver.sv
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: The Hex driver module converts a 4 bit input to display the right output in hex on the hex displays.

Block Diagram:-

Hidden Questions:-

Hidden Question #1/2:
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.

USB keyboards require polling while PS/2 keyboards send interrupts whenever a key is pressed, so the PS/2 keyboard doesn't occupy the CPU as much.
USB keyboards can handle a maximum of six simultaneous key presses.

Hidden Question #2/2:
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? Give an answer in your Post-Lab.

The old value will be used. During a bounce, the ball will reach the edge and move one step past before the motion is fixed. At that point, it will move back towards the center. This reassignment of Ball_Y_Motion is based on which edge the ball has reached, instead of just telling the ball to reverse its motion. Because of this, even though the ball will reach the edge twice in a single bounce, it will end up going in the correct direction.

Postlab:-

1) The VGA_clk is 25 MHz, or half the speed of clk.
2) It could be that we wanted to have space in Nios around where data is mapped. If we made otg_hpi_r an int pointer, it's value would be right next to the value for otg_hpi_data. Another option could be that we wanted to be consistent, and use char pointers for single bit variables and int pointers for multiple bit variables. Other than that, I can think of no good reason to not use a char pointer for otg_hpi_address.

	Accumulator
LUT	2706
DSP	10
Memory (BRAM)	55296
Flip-Flop	2043

Frequency	136.91 MHz
Static Power	101.97 mW
Dynamic Power	0.95 mW
Total Power	173.37 mW

Conclusion:-

Our design functioned properly. We thought that the timing diagrams were misleading as they seemed to imply that we should write to otg_hpi_address before setting otg_hpi_cs low. This was not the case, and was the only significant delay we encountered while working on this lab.