ECE 385

Fall 2019

Experiment #8

SOC with USB and VGA Interface in SystemVerilog

Pouya Akbarzadeh and Patrick Stach

Section: ABC

Lab TA: Vikram Anjur, Yuhong Li

Index

8.1 Introduction

The purpose of this lab was to experiment with C code VHDL to work with a monitor and a keyboard. The experiment allowed us to create a very simple game in which we controlled a ball where it bounced off the walls and where it could not have travelled diagonally. We used the W, A, S, and D key to travel up, left, down, and right. After each key press, the corresponding ASCII value was also displayed to the FPGA board.

The monitor was able to display content through a VGA connection, and the keyboard used USB interface to send data to the FPGA board. Universal Serial Bus (USB) is an industry standard that sets out requirements for cables and connectors and protocols for connection, communication and power supply between computers, peripherals and other devices. The NIOS II chip can handle the keycode sent from keyboard, and the DE2 board has a USB controller which can handle data transmission. Video Graphics Array, also known as VGA, also considered my favorite I/Os has 15 pins. These 15 pins are used to send analog component RGBHV video signals. RGBHV stands for red, green, blue, horizontal sunc, and vertical sync.

8.2 Written Description of Lab 8 System

8.2.1 NIOS Interaction

We have similar functionality of the NIOS interaction as Lab 7 because some modules were re-used, specifically clk_0, nios2_gen2_0, onchip_memory2_0, sdram, and sdram_pll. These modules are the backbone to the operation of the system. Clk_0 outputs a clock signal that connects to all components that are synchronous on the same clock. The nios2_gen2_0 is our processor which will perform operations and read instructions. For memory storage, we have onchip_memory2_0 which although is faster than the SDRAM, there is much less of it in terms of storage space. The sdram_pll generates the clock that goes into the SDRAM, 3ns behind the system clock in order to give output enough time to stabilize.

One of the new modules we introduce is the jtag_uart modules, which allows you to use a terminal on your computer to communicate with the NIOS II. The other modules we had were specific to the way the Cypress EZ-OTG (CY7C67200) chip handles the protocol. These were specifically hpi_address (chooses which HPI register to write to), hpi_cs (chip select), hpi_r (read), hpi_w (write), and otg_hpi_data (our data).

8.2.2 USB Protocol

The Nios II handles the USB protocol with its software, and the keycode that is extracted is outputted to other hardware that checks for keypresses (ie. ball.sv). We had to write out functions such as IO_write and IO_read based on the datasheet showed the timing diagram and the order of the operations. For example:

IO_write:

1) Write address in hpi_address location
2) Set cs to 0 (active low)
3) Set w to 0 (active low)
4) Write data in hpi_data location
5) Set w to 1
6) Set cs to w

And for

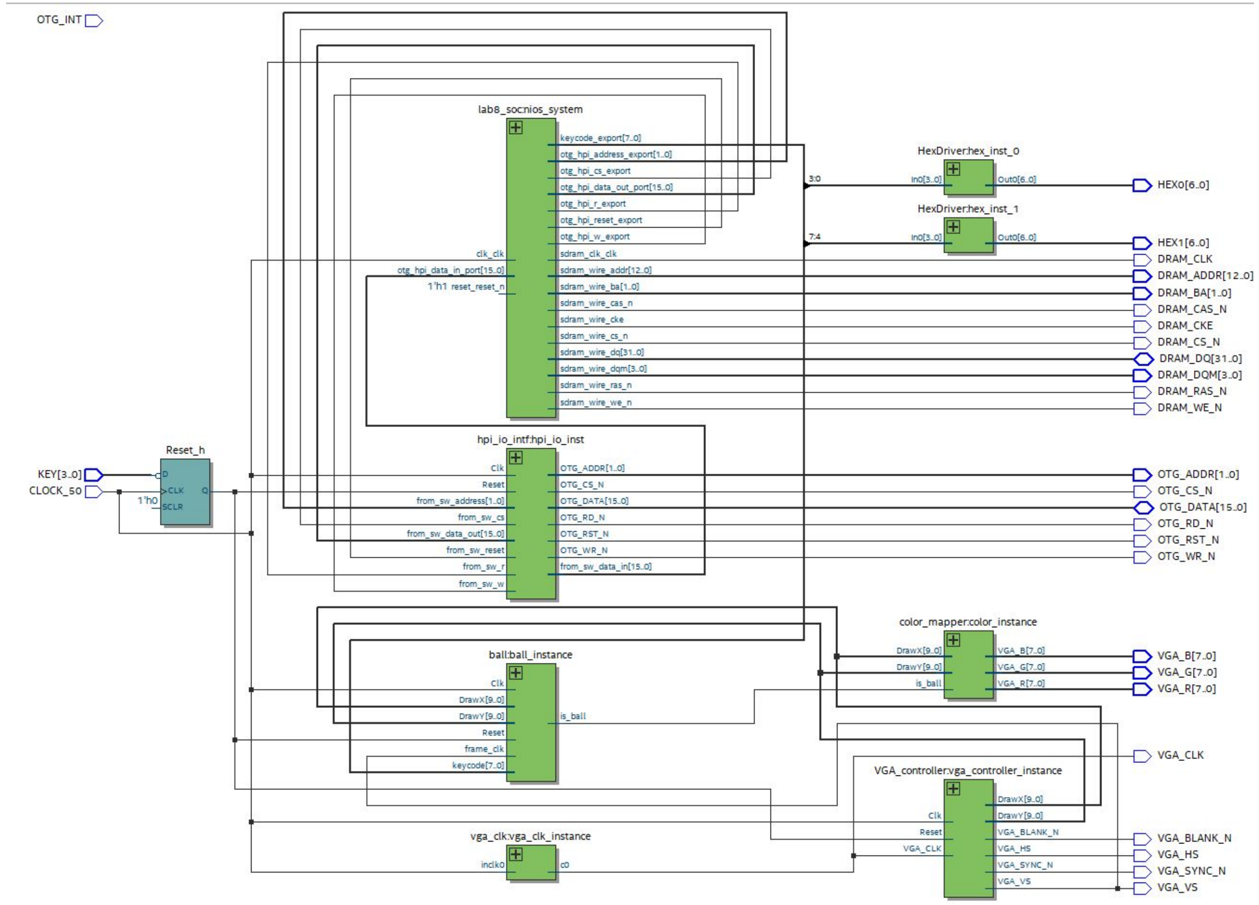
IO_read:

1) Write address in hpi_address location
2) Set cs to 0 (active low)
3) Set w to 0 (active low)
4) Store hpi_data in temp variable
5) Set w to 1
6) Set cs to 1
7) Return temp.

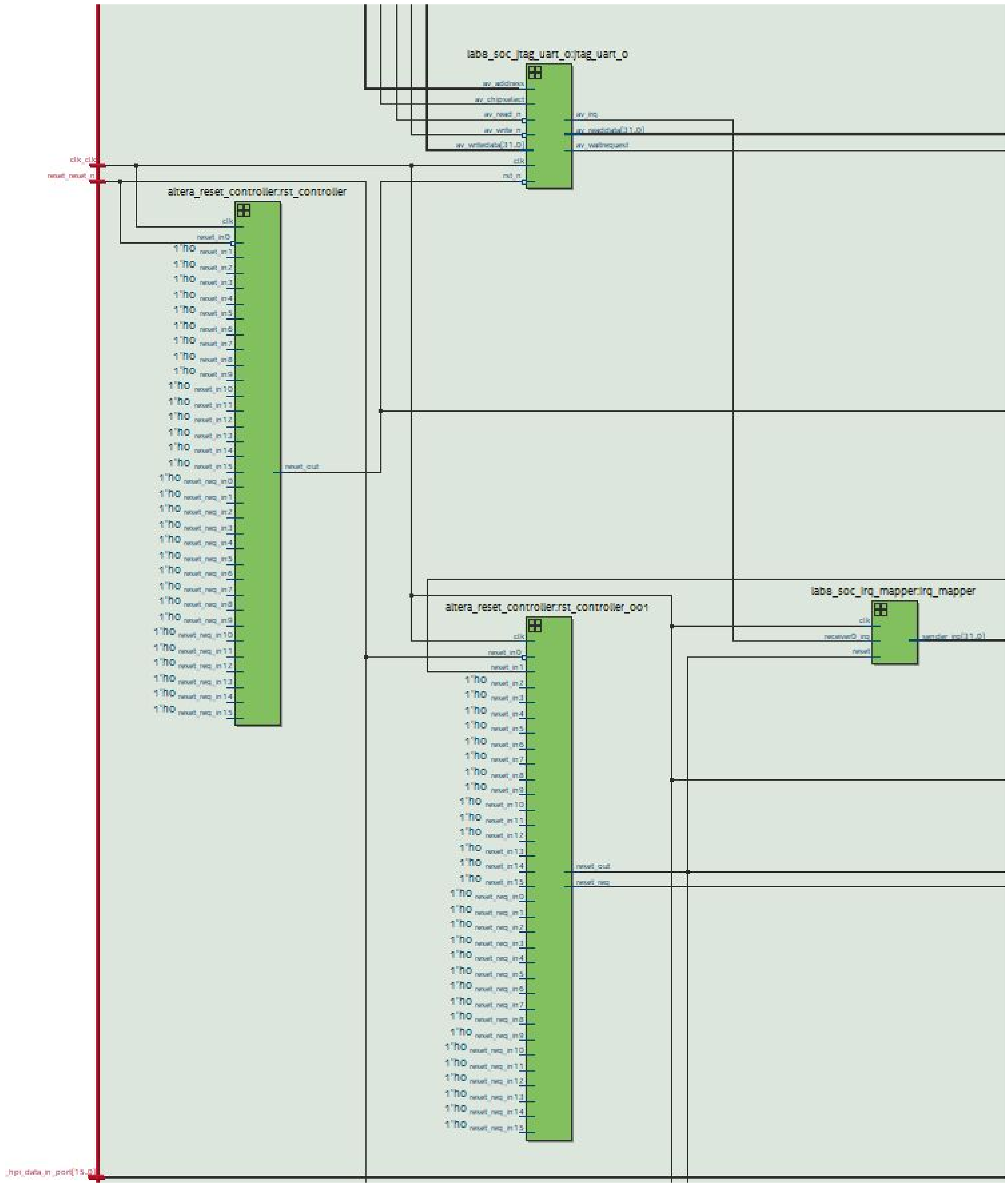
The order of the signals is very important in order to match up with the protocol, and incorrect order/ programming can lead to failure of reads or writes

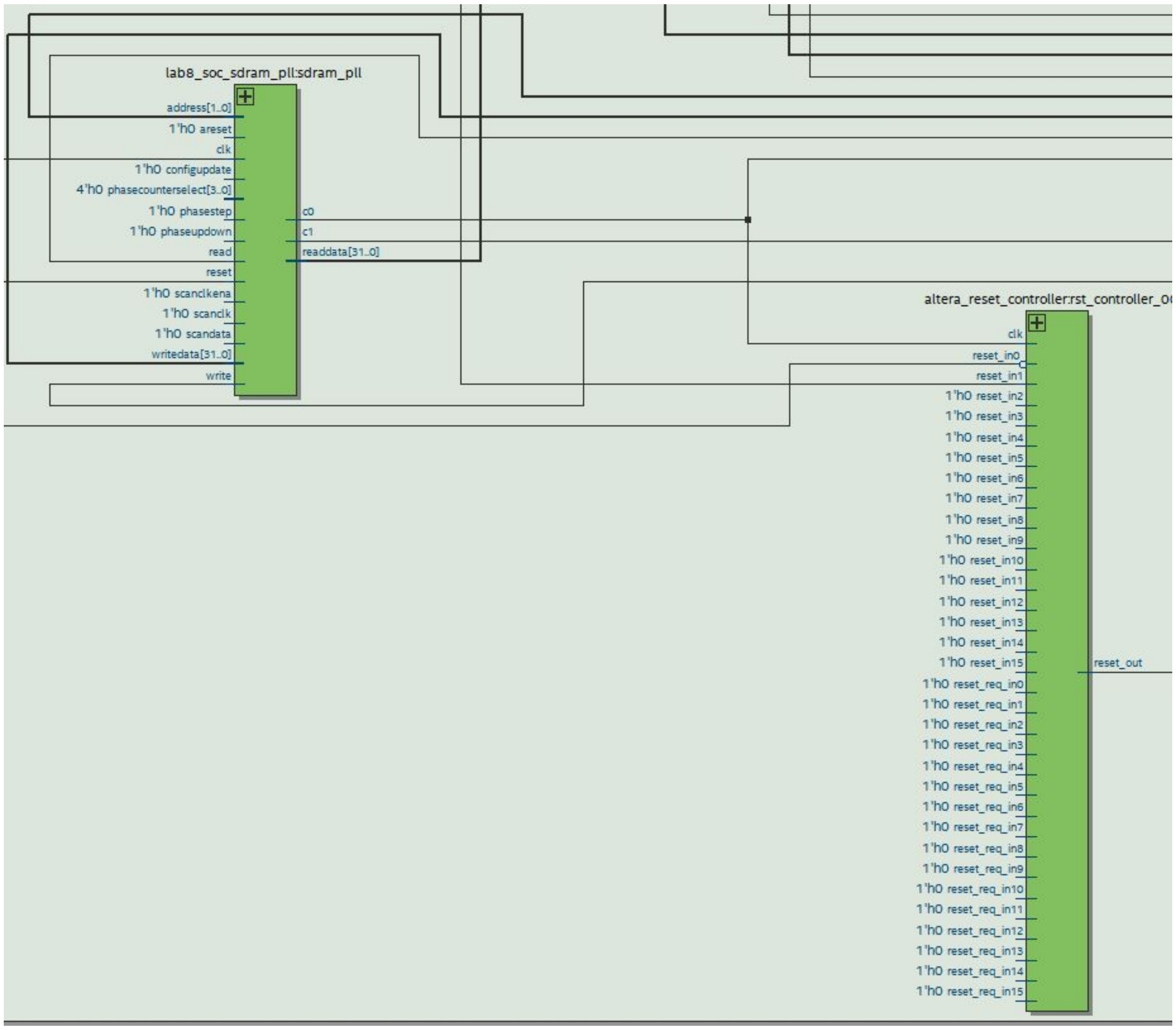
8.2.3 Block Diagram

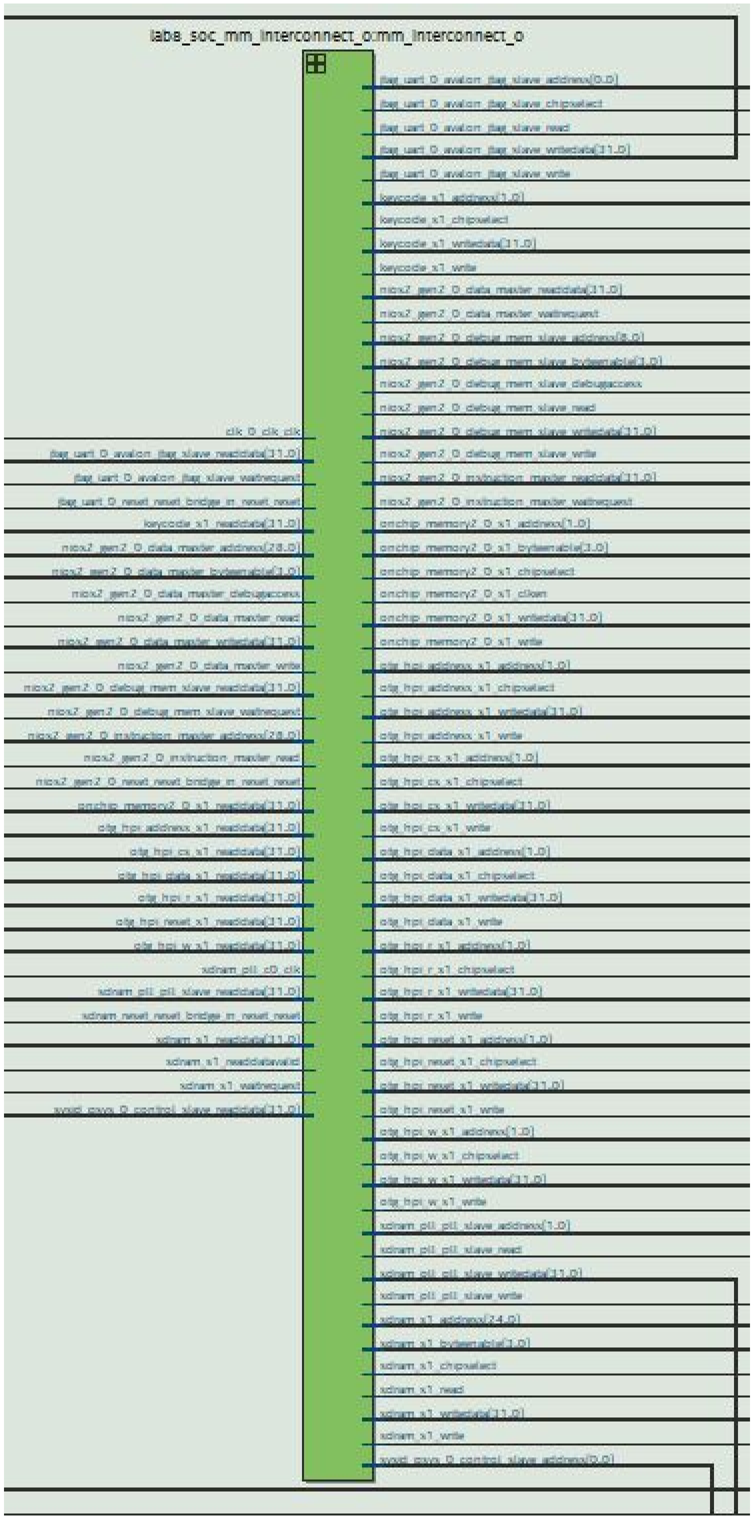
Top Level

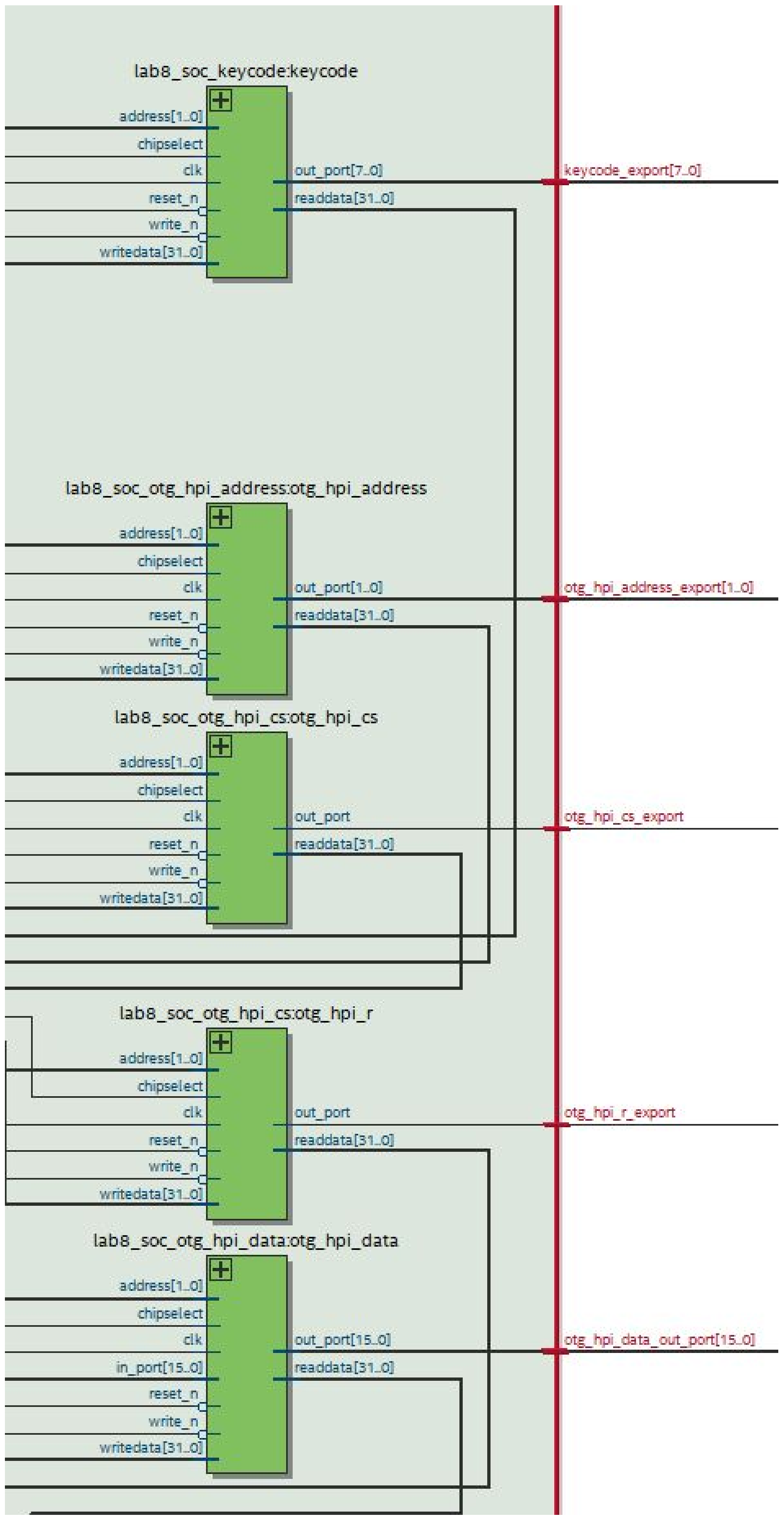

OTG_INT
lab8_soc:nios_system
keycode_export[7..0]
otg_hpi_address_export[1..0]
otg_hpi_cs_export
otg_hpi_data_out_port[15..0]
otg_hpi_r_export
otg_hpi_reset_export
otg_hpi_w_export
clk_clk
sdram_clk_clk
otg_hpi_data_in_port[15..0]
sdram_wire_addr[12..0]
1'h1 reset_reset_n
sdram_wire_ba[1..0]
sdram_wire_cas_n
sdram_wire_cke
sdram_wire_cs_n
sdram_wire_dq[31..0]
sdram_wire_dqm[3..0]
sdram_wire_ras_n
sdram_wire_we_n
HexDriver:hex_inst_0
3:0
In0[3..0]
Out0[6..0]
HEX0[6..0]
HexDriver:hex_inst_1
7:4
In0[3..0]
Out0[6..0]
HEX1[6..0]
DRAM_CLK
DRAM_ADDR[12..0]
DRAM_BA[1..0]
DRAM_CAS_N
DRAM_CKE
DRAM_CS_N
DRAM_DQ[31..0]
DRAM_DQM[3..0]
DRAM_RAS_N
DRAM_WE_N
Reset_h
KEY[3..0]
CLOCK_50
D
CLK
SCLR
Q
1'h0
hpi_io_intf:hpi_io_inst
Clk
Reset
from_sw_address[1..0]
from_sw_cs
from_sw_data_out[15..0]
from_sw_reset
from_sw_r
from_sw_w
OTG_ADDR[1..0]
OTG_CS_N
OTG_DATA[15..0]
OTG_RD_N
OTG_RST_N
OTG_WR_N
from_sw_data_in[15..0]
OTG_ADDR[1..0]
OTG_CS_N
OTG_DATA[15..0]
OTG_RD_N
OTG_RST_N
OTG_WR_N
color_mapper:color_instance
DrawX[9..0]
DrawY[9..0]
is_ball
VGA_B[7..0]
VGA_G[7..0]
VGA_R[7..0]
VGA_B[7..0]
VGA_G[7..0]
VGA_R[7..0]
ball:ball_instance
Clk
DrawX[9..0]
DrawY[9..0]
Reset
frame_clk
keycode[7..0]
is_ball
VGA_CLK
VGA_controller:vga_controller_instance
Clk
Reset
VGA_CLK
DrawX[9..0]
DrawY[9..0]
VGA_BLANK_N
VGA_HS
VGA_SYNC_N
VGA_VS
VGA_BLANK_N
VGA_HS
VGA_SYNC_N
VGA_VS
vga_clk:vga_clk_instance
inclk0
c0

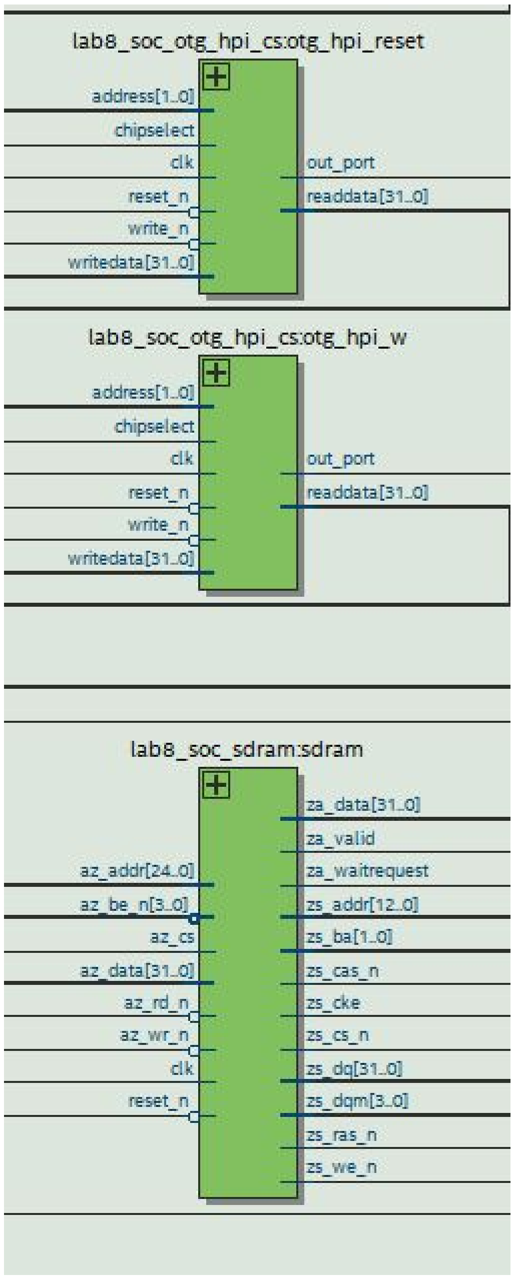
Nios II Expanded

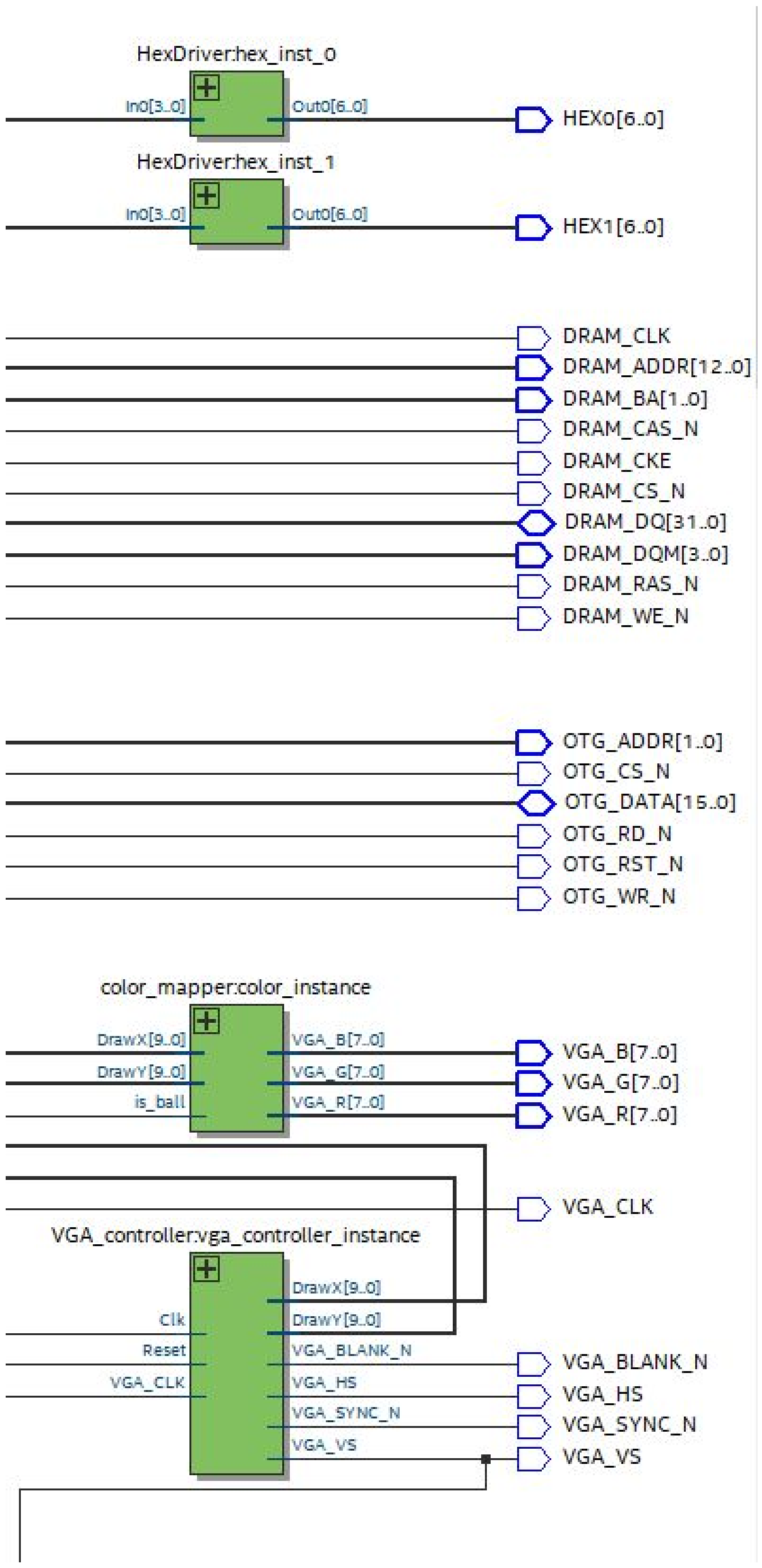



lab8_soc_sdram_pll:sdram_pll
address[1..0]
1'h0 areset
clk
1'h0 configupdate
4'h0 phasecounterselect[3..0]
1'h0 phasestep
1'h0 phaseupdown
read
reset
1'h0 scanclkena
1'h0 scanclk
1'h0 scandata
writedata[31..0]
write
c0
c1
readdata[31..0]
altera_reset_controller:rst_controller_0
clk
reset_in0
reset_in1
1'h0 reset_in2
1'h0 reset_in3
1'h0 reset_in4
1'h0 reset_in5
1'h0 reset_in6
1'h0 reset_in7
1'h0 reset_in8
1'h0 reset_in9
1'h0 reset_in10
1'h0 reset_in11
1'h0 reset_in12
1'h0 reset_in13
1'h0 reset_in14
1'h0 reset_in15
1'h0 reset_req_in0
1'h0 reset_req_in1
1'h0 reset_req_in2
1'h0 reset_req_in3
1'h0 reset_req_in4
1'h0 reset_req_in5
1'h0 reset_req_in6
1'h0 reset_req_in7
1'h0 reset_req_in8
1'h0 reset_req_in9
1'h0 reset_req_in10
1'h0 reset_req_in11
1'h0 reset_req_in12
1'h0 reset_req_in13
1'h0 reset_req_in14
1'h0 reset_req_in15
reset_out


lab8_soc_mm_interconnect_0:mm_interconnect_0
clk_0_clk_clk
jtag_uart_0_avalon_jtag_slave_readdata[31..0]
jtag_uart_0_avalon_jtag_slave_waitrequest
jtag_uart_0_reset_reset_bridge_in_reset_reset
keycode_s1_readdata[31..0]
nios2_gen2_0_data_master_address[28..0]
nios2_gen2_0_data_master_byteenable[3..0]
nios2_gen2_0_data_master_debugaccess
nios2_gen2_0_data_master_read
nios2_gen2_0_data_master_writedata[31..0]
nios2_gen2_0_data_master_write
nios2_gen2_0_debug_mem_slave_readdata[31..0]
nios2_gen2_0_debug_mem_slave_waitrequest
nios2_gen2_0_instruction_master_address[28..0]
nios2_gen2_0_instruction_master_read
nios2_gen2_0_reset_reset_bridge_in_reset_reset
onchip_memory2_0_s1_readdata[31..0]
otg_hpi_address_s1_readdata[31..0]
otg_hpi_cs_s1_readdata[31..0]
otg_hpi_data_s1_readdata[31..0]
otg_hpi_r_s1_readdata[31..0]
otg_hpi_reset_s1_readdata[31..0]
otg_hpi_w_s1_readdata[31..0]
sdram_pll_c0_clk
sdram_pll_pll_slave_readdata[31..0]
sdram_reset_reset_bridge_in_reset_reset
sdram_s1_readdata[31..0]
sdram_s1_readdatavalid
sdram_s1_waitrequest
sysid_qsys_0_control_slave_readdata[31..0]
jtag_uart_0_avalon_jtag_slave_address[0..0]
jtag_uart_0_avalon_jtag_slave_chipselect
jtag_uart_0_avalon_jtag_slave_read
jtag_uart_0_avalon_jtag_slave_writedata[31..0]
jtag_uart_0_avalon_jtag_slave_write
keycode_s1_address[1..0]
keycode_s1_chipselect
keycode_s1_writedata[31..0]
keycode_s1_write
nios2_gen2_0_data_master_readdata[31..0]
nios2_gen2_0_data_master_waitrequest
nios2_gen2_0_debug_mem_slave_address[8..0]
nios2_gen2_0_debug_mem_slave_byteenable[3..0]
nios2_gen2_0_debug_mem_slave_debugaccess
nios2_gen2_0_debug_mem_slave_read
nios2_gen2_0_debug_mem_slave_writedata[31..0]
nios2_gen2_0_debug_mem_slave_write
nios2_gen2_0_instruction_master_readdata[31..0]
nios2_gen2_0_instruction_master_waitrequest
onchip_memory2_0_s1_address[1..0]
onchip_memory2_0_s1_byteenable[3..0]
onchip_memory2_0_s1_chipselect
onchip_memory2_0_s1_clken
onchip_memory2_0_s1_writedata[31..0]
onchip_memory2_0_s1_write
otg_hpi_address_s1_address[1..0]
otg_hpi_address_s1_chipselect
otg_hpi_address_s1_writedata[31..0]
otg_hpi_address_s1_write
otg_hpi_cs_s1_address[1..0]
otg_hpi_cs_s1_chipselect
otg_hpi_cs_s1_writedata[31..0]
otg_hpi_cs_s1_write
otg_hpi_data_s1_address[1..0]
otg_hpi_data_s1_chipselect
otg_hpi_data_s1_writedata[31..0]
otg_hpi_data_s1_write
otg_hpi_r_s1_address[1..0]
otg_hpi_r_s1_chipselect
otg_hpi_r_s1_writedata[31..0]
otg_hpi_r_s1_write
otg_hpi_reset_s1_address[1..0]
otg_hpi_reset_s1_chipselect
otg_hpi_reset_s1_writedata[31..0]
otg_hpi_reset_s1_write
otg_hpi_w_s1_address[1..0]
otg_hpi_w_s1_chipselect
otg_hpi_w_s1_writedata[31..0]
otg_hpi_w_s1_write
sdram_pll_pll_slave_address[1..0]
sdram_pll_pll_slave_read
sdram_pll_pll_slave_writedata[31..0]
sdram_pll_pll_slave_write
sdram_s1_address[24..0]
sdram_s1_byteenable[3..0]
sdram_s1_chipselect
sdram_s1_read
sdram_s1_writedata[31..0]
sdram_s1_write
sysid_qsys_0_control_slave_address[0..0]

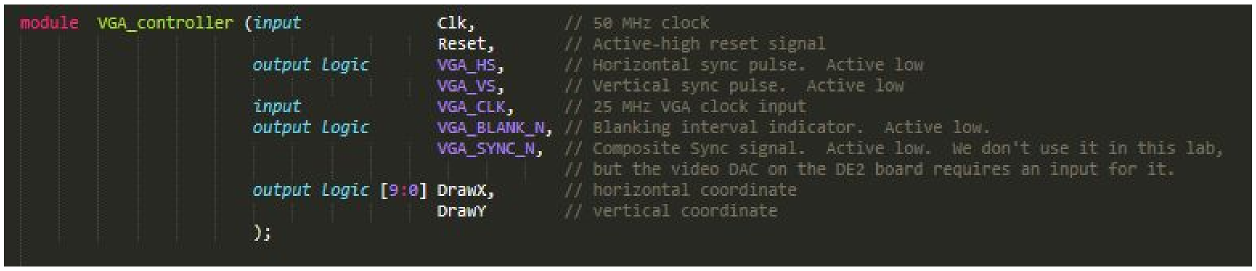

lab8_soc_keycode:keycode
address[1..0]
chipselect
clk
reset_n
write_n
writedata[31..0]
out_port[7..0]
readdata[31..0]
keycode_export[7..0]
lab8_soc_otg_hpi_address:otg_hpi_address
address[1..0]
chipselect
clk
reset_n
write_n
writedata[31..0]
out_port[1..0]
readdata[31..0]
otg_hpi_address_export[1..0]
lab8_soc_otg_hpi_cs:otg_hpi_cs
address[1..0]
chipselect
clk
reset_n
write_n
writedata[31..0]
out_port
readdata[31..0]
otg_hpi_cs_export
lab8_soc_otg_hpi_cs:otg_hpi_r
address[1..0]
chipselect
clk
reset_n
write_n
writedata[31..0]
out_port
readdata[31..0]
otg_hpi_r_export
lab8_soc_otg_hpi_data:otg_hpi_data
address[1..0]
chipselect
clk
in_port[15..0]
reset_n
write_n
writedata[31..0]
out_port[15..0]
readdata[31..0]
otg_hpi_data_out_port[15..0]


lab8_soc_otg_hpi_cs:otg_hpi_reset
address[1..0]
chipselect
clk
reset_n
write_n
writedata[31..0]
out_port
readdata[31..0]
lab8_soc_otg_hpi_cs:otg_hpi_w
address[1..0]
chipselect
clk
reset_n
write_n
writedata[31..0]
out_port
readdata[31..0]
lab8_soc_sdram:sdram
za_data[31..0]
za_valid
za_waitrequest
zs_addr[12..0]
zs_ba[1..0]
zs_cas_n
zs_cke
zs_cs_n
zs_dq[31..0]
zs_dqm[3..0]
zs_ras_n
zs_we_n
az_addr[24..0]
az_be_n[3..0]
az_cs
az_data[31..0]
az_rd_n
az_wr_n
clk
reset_n



8.2.4 Module Description

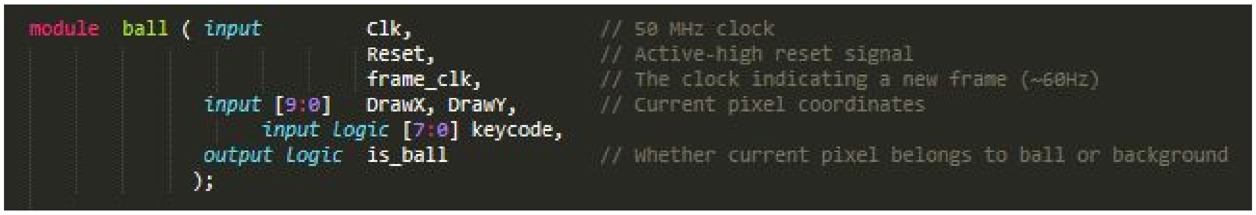
Module: VGA Controller



```
module  VGA_controller (input              Clk,         // 50 MHz clock
                                           Reset,       // Active-high reset signal
                        output logic       VGA_HS,      // Horizontal sync pulse.  Active low
                                           VGA_VS,      // Vertical sync pulse.  Active low
                        input              VGA_CLK,     // 25 MHz VGA clock input
                        output logic       VGA_BLANK_N, // Blanking interval indicator.  Active low.
                                           VGA_SYNC_N,  // 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 logic [9:0] DrawX,       // horizontal coordinate
                                           DrawY        // vertical coordinate
                        );
```

Description: This module handles the synchronization of signals(VS = vertical sync, HS = horizontal sync) of the VGA signal we are outputting in addition to "drawing" pixels
Purpose: Used to display the ball bouncing on the screen, as an output from the FPGA

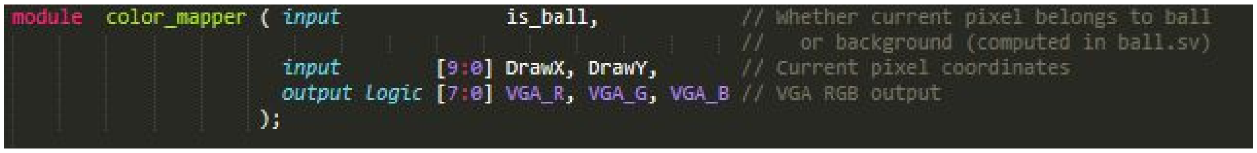
Module: Ball



```
module  ball ( input         Clk,                // 50 MHz clock
                             Reset,              // Active-high reset signal
                             frame_clk,          // The clock indicating a new frame (~60Hz)
               input [9:0]   DrawX, DrawY,       // Current pixel coordinates
                   input logic [7:0] keycode,
               output logic  is_ball             // Whether current pixel belongs to ball or background
              );
```

Description: Updates the position and motion of the ball only at the rising edge of frame clock. If no keys are pressed it keeps the motion unchanged.
Purpose: Calculate the positions and reacts to keypresses which are from the user via the keyboard. Defines ball operation

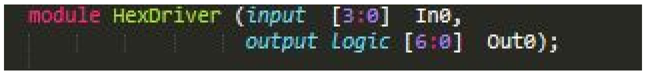
Module: Color Mapper



```
module  color_mapper ( input              is_ball,            // Whether current pixel belongs to ball
                                                              //   or background (computed in ball.sv)
                       input        [9:0] DrawX, DrawY,       // Current pixel coordinates
                       output logic [7:0] VGA_R, VGA_G, VGA_B // VGA RGB output
                     );
```

Description: Decides which color to be output to VGA for each pixel. Whether the pixel belongs to ball or background, and uses RGB color selection.
Purpose: Is used to draw the ball, background, and implement RGB colors on screen.

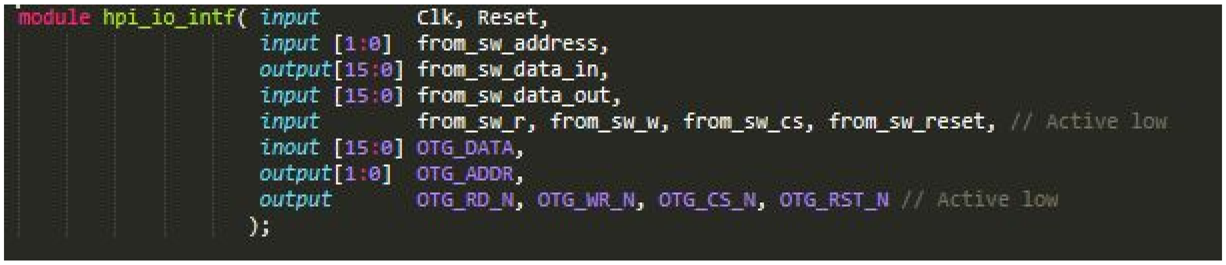
Module: Hex Driver



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

Description: The HexDriver was provided to us during Experiment 4 and was simply recycled for this experiment. The HexDriver translates a binary input representing a number into the specific segments of a 7-segment display that would need to be able to display that number in hex.
Purpose: This is used to display inputs on the FPGA board in run-time.

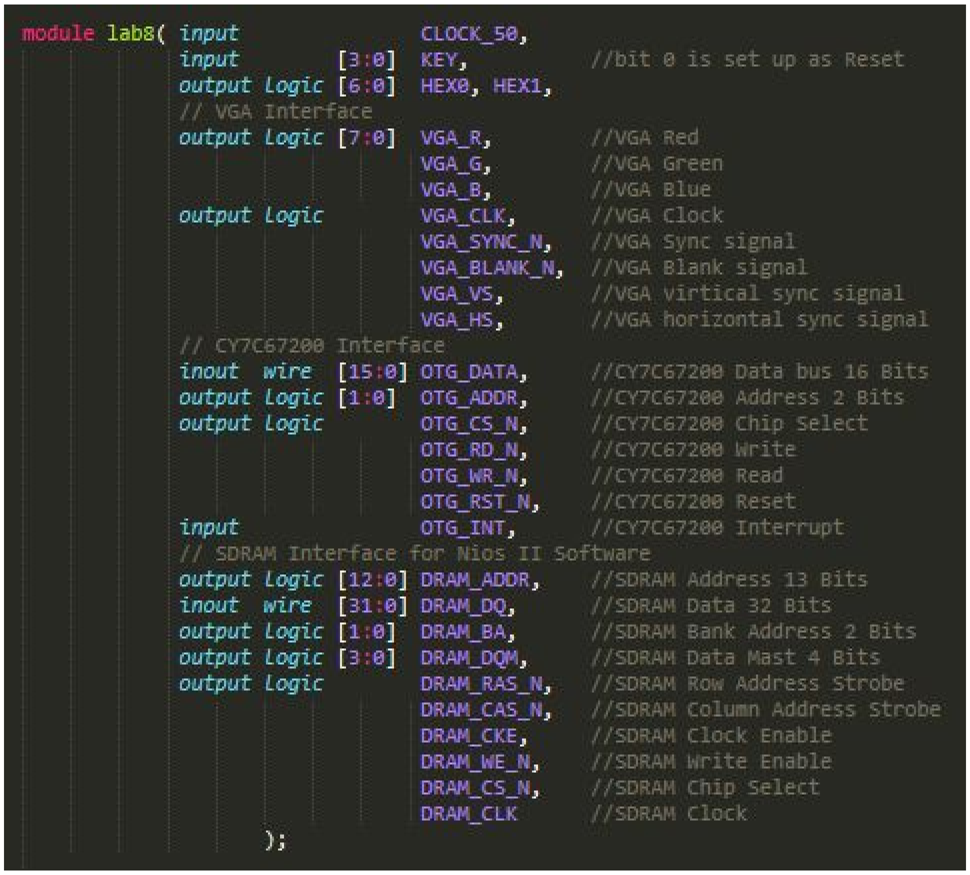
Module: hpi_io_intf



```
module hpi_io_intf( 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: Makes OTG_DATA is high Z (tristated) when NIOS is not writing to OTG_DATA inout bus. Also handles buffers.
Purpose: Processes and handles the OTG data

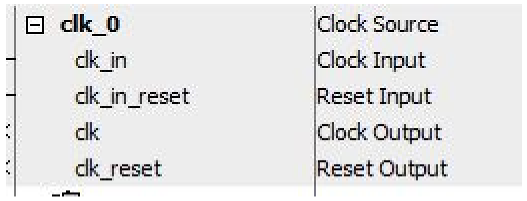
Module: lab8



```
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
                   );
```

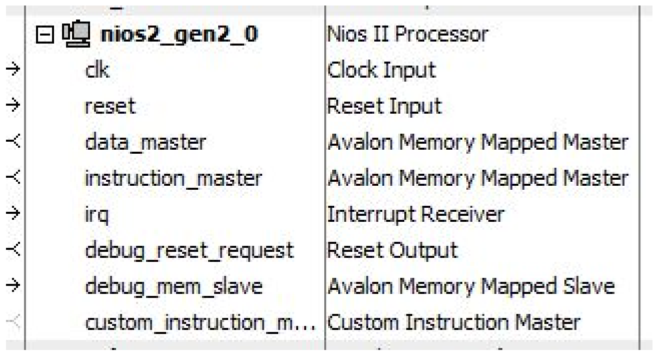
Description: This is the top level, all inputs and outputs go through it, and this module helps them communicate with one another.
Purpose: Connects the NIOS to all the blocks and drivers.

Platform Designer Modules



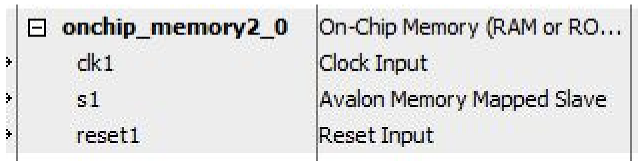
⊟ **clk_0**	Clock Source
clk_in	Clock Input
clk_in_reset	Reset Input
clk	Clock Output
clk_reset	Reset Output

This module is the clock module- it is simply the 50Mhz generated by the FPGA. The clk from here goes to all the other clocks inputs



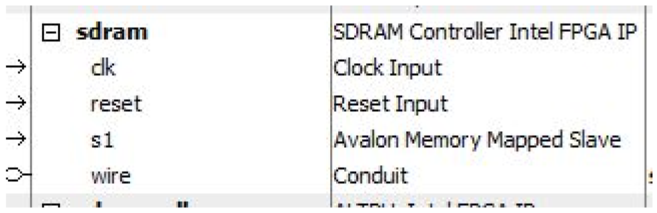
⊟ **nios2_gen2_0**	Nios II Processor
clk	Clock Input
reset	Reset Input
data_master	Avalon Memory Mapped Master
instruction_master	Avalon Memory Mapped Master
irq	Interrupt Receiver
debug_reset_request	Reset Output
debug_mem_slave	Avalon Memory Mapped Slave
custom_instruction_m...	Custom Instruction Master

This is the actual processor- in our case we have the Nios II/e, the economy version of the Nios II/f. This is the module responsible for processing all the instructions. The NIOS II is a 32-bit modified Harvard RISC architecture.



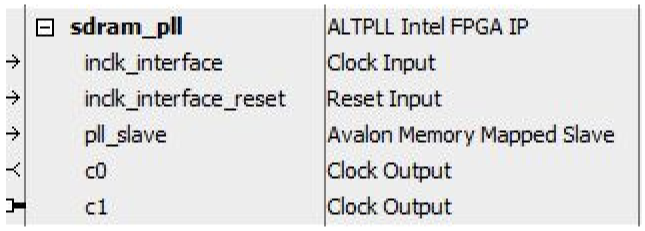
⊟ **onchip_memory2_0**	On-Chip Memory (RAM or RO...
clk1	Clock Input
s1	Avalon Memory Mapped Slave
reset1	Reset Input

This module is our on-chip memory, which is often smaller than SRAM in size but is faster and is actually on the chip. The data width is 32 bits and the total memory size is 16 bytes



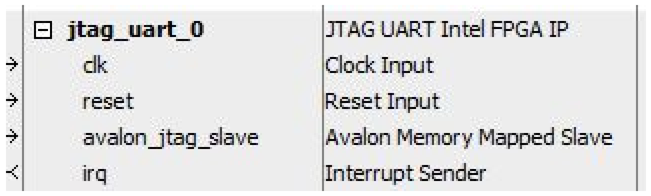
⊟ **sdram**	SDRAM Controller Intel FPGA IP
clk	Clock Input
reset	Reset Input
s1	Avalon Memory Mapped Slave
wire	Conduit

This module is our SDRAM that we are interfacing with. We use SDRAM to store the software program because the on-chip memory is limited. We have to use an SDRAM controller to interface with the bus since we have row/column addressing and constantly needs to refresh in order to retain data.

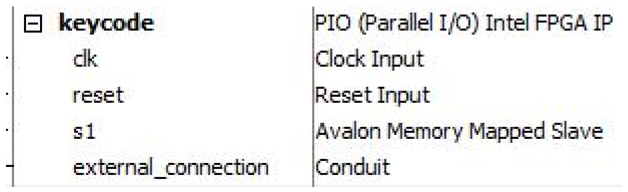


sdram_pll	ALTPLL Intel FPGA IP
inclk_interface	Clock Input
inclk_interface_reset	Reset Input
pll_slave	Avalon Memory Mapped Slave
c0	Clock Output
c1	Clock Output

This module generates the clock that goes into the SDRAM. The PLL allows us to account for delays, specifically 3ns in order to have the SDRAM wait for the outputs to stabilize.

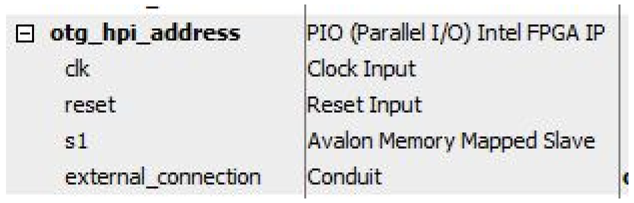


jtag_uart_0	JTAG UART Intel FPGA IP
clk	Clock Input
reset	Reset Input
avalon_jtag_slave	Avalon Memory Mapped Slave
irq	Interrupt Sender



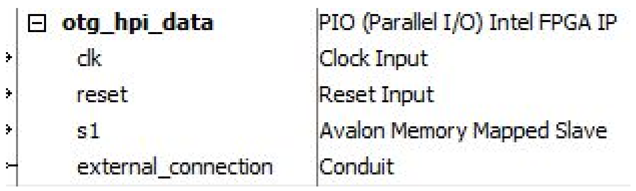
keycode	PIO (Parallel I/O) Intel FPGA IP
clk	Clock Input
reset	Reset Input
s1	Avalon Memory Mapped Slave
external_connection	Conduit

This module is a simple PIO block, which outputs the keycode from the IO_READ (keyboard), 8 bits wide



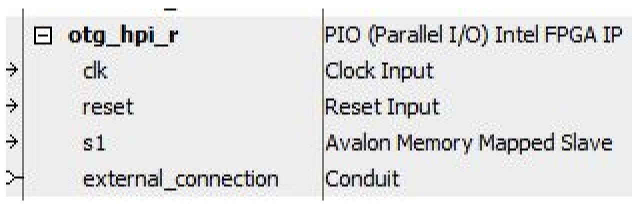
otg_hpi_address	PIO (Parallel I/O) Intel FPGA IP
clk	Clock Input
reset	Reset Input
s1	Avalon Memory Mapped Slave
external_connection	Conduit

This module is a simple PIO block, which outputs the 2 bit value corresponding to the specific HPI register



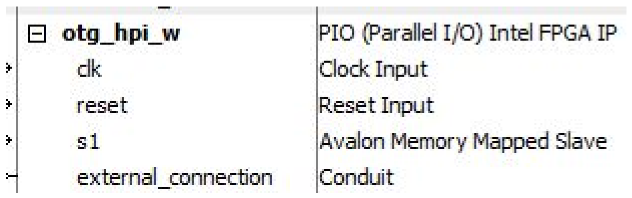
otg_hpi_data	PIO (Parallel I/O) Intel FPGA IP
clk	Clock Input
reset	Reset Input
s1	Avalon Memory Mapped Slave
external_connection	Conduit

This module is a simple PIO block, which is inout because data is both read from and written to here. This block is 32 bits wide.



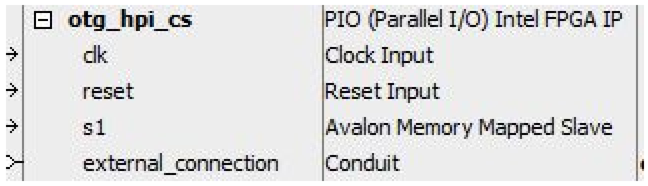
⊟ **otg_hpi_r**	PIO (Parallel I/O) Intel FPGA IP
clk	Clock Input
reset	Reset Input
s1	Avalon Memory Mapped Slave
external_connection	Conduit

This module is a simple PIO block, which is a 1bit output corresponding to a "read" enable signal



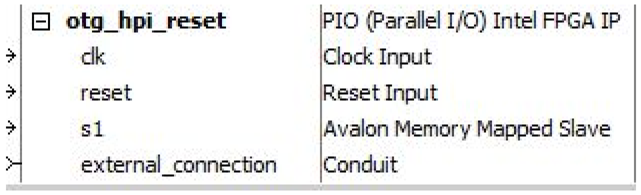
⊟ **otg_hpi_w**	PIO (Parallel I/O) Intel FPGA IP
clk	Clock Input
reset	Reset Input
s1	Avalon Memory Mapped Slave
external_connection	Conduit

This module is a simple PIO block, which is a 1bit output corresponding to a "write" enable signal



⊟ **otg_hpi_cs**	PIO (Parallel I/O) Intel FPGA IP
clk	Clock Input
reset	Reset Input
s1	Avalon Memory Mapped Slave
external_connection	Conduit

This module is a simple PIO block, which is a 1bit output corresponding to a "chip enable" signal



⊟ **otg_hpi_reset**	PIO (Parallel I/O) Intel FPGA IP
clk	Clock Input
reset	Reset Input
s1	Avalon Memory Mapped Slave
external_connection	Conduit

This module is a simple PIO block, which is a 1bit output corresponding to a "reset" signal

<u>8.3 Questions</u>

8.3.1 Hidden Questions

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.

If you want to use PS/2 mouse or keyboard, you need to shutdown the system, plug it in then reboot it. While for USB you can just plug it into the system and after a second or so you can use it. However, PS/2 has faster response than USB. This is due to the fact that PS/2 directly connects to the MOBO while USB connects to the BUS which is then makes contact with the MOBO. Thus USB is “further” from the processor. You can send files over USB but not over PS/2 because PS/2 is always an HID which is good and bad. In some sense PS/2 is more secure because you can not be sure that cheap USB mouse off Amazon isn’t transmitting files.

2. 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?

   When signals are being assigned on the same clock edge, we used parallel assignments which causes the old value to be utilized rather than the new one. This means that when the ball is “bouncing” it is actually would continue moving into the “wall”. This happens for only 1 frame.

   If the key press didn’t take into account the edge case, the ball would move in the wrong direction. Again this would happen in 1 frame. The fix to all these issues was accounting for them with conditional (if, elseif, and else statements).

8.3.2 Post Lab Questions

1.

LUT	**2689**
DSP	**0**
BRAM	**55,296**
Flip-Flop	**2234**

Frequency	**130.28 MHz**
Static Power	**101.15mW**
Dynamic Power	**0.74mW**
Total Power	**174.84mW**

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?

Recall that an integer is 32 bits while a char is 8 characters. We need the integer pointer because our data is 32 bits wide. Meanwhile, we only need a char pointer to point to the HPI registers because we only have 4 registers and the system cannot allocate less than a byte.

3. What is the difference between VGA_CLK and Clk?

VGA_Clk runs at 25Mhz where as Clk runs at 50 Mhz. The VGA_Clk is used to update the monitor frames while the Clk is used for the FPGA.

8.4 Conclusion

In conclusion, there were no issues during the demo. The ball was able to move in all 4 directions, but not diagonally. The key presses showed up correctly on the FPGA board on the hex display. While this lab at first seemed very intimidating and difficult, the parts started falling altogether as we understood the different aspects and worked on it. The most confusing part and the biggest issue we had was the fact that 1 keyboard did not work with our board, and caused a lot of confusion and added about 40 mins to our debugging time. However, it was an easy fix. Over all this lab was a great introduction for implementing I/O and how we can implement it in future labs and projects.