ECE 385

EXPERIMENT #8

SOC with USB and VGA Interface in SystemVerilog

I. OBJECTIVE

In this experiment you will write a protocol to interface a keyboard and a monitor with the DE2 board using the on-board USB and VGA ports.

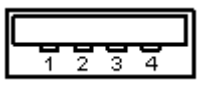
II. INTRODUCTION

You will connect the monitor to the VGA port and the keyboard to the USB port and depending on the key pressed on the keyboard, a small red ball will move and bounce in either the X or Y direction on the monitor screen

How the USB Keyboard Works

The Universal Serial Bus (USB) standard defines the connection and communication protocols between computers and electronic devices. It also provides power supply to the connected devices. Due to the compatibility with a wide variety of devices, the USB standard has become prevalent since its introduction in the 1990s.

A USB cable has either a type A or a type B port on its ends. A USB port consists of four pins: VDD, D-, D+, and GND. Figure 1 shows the configuration. The VDD and GND pins are power lines, and D- and D+ are data lines. When data is being transmitted, D- and D+ take opposite voltage levels in a single time frame to represent one bit of data. On low and full speed devices, a differential '1' is transmitted by pulling D+ high and D- low, while a differential '0' is a D- pulled high a D+ pulled low.



Type A

Type B

Pin	Name
1	VDD (5V)
2	D-
3	D+
4	GND

Figure 1

The USB port on the DE2 board is equipped with the Cypress EZ-OTG (CY7C67200) USB Controller, which handles all the data transmission via the physical USB port and manages structured information to and from the DE2 board. CY7C67200 can act as either a Host Controller, which controls the connected devices, or a Device Controller, which makes the DE2 board itself a USB device. In this lab, we will be using CY7C67200 as a Host Controller.

A USB keyboard is a Human Interface Device (HID). HIDs usually do not require massive data transfers at any given moment, so a low speed transmission would suffice. (Other USB devices such as a camera or a mass storage device would often need to send large files, which would require bulk transfers, a topic not covered in this lab.) Unlike earlier standards such as PS/2, a USB keyboard does not send key press signals on its own. All USB devices send information only when requested by the host. In order to receive key press signals promptly, the host needs to constantly poll information from the keyboard. In this lab, after proper configuration, ISP1362 will constantly send *interrupt requests* to the keyboard, and the keyboard will respond with key press information in *report descriptors*. A descriptor simply means a data structure in which the information is stored.

Table 1 shows the keyboard input report format (8 bytes). In this format, a maximum of 6 simultaneous key presses can be handled, but here we will assume only one key is pressed at a time, which means we only need to look at the first key code. Each key code is an 8-bit hex number. For example, the character A is represented by 0x04, B by 0x05, and so on. When the key is not pressed, or is released, the key code will be 0x00 (No Event).

Byte	Description
0	Modifiers Keys
1	Reserved
2	Keycode 1
3	Keycode 2
4	Keycode 3
5	Keycode 4
6	Keycode 5
7	Keycode 6

Table 1

A more detailed explanation of how the keyboard works (and all the key code combinations) can be found in the INTRODUCTION TO USB AND EZ-OTG ON NIOS II (IUQ. 1-13). In particular, the USB 2.0 Specification and the HID Device Class Definition (refer to the ECE 385 course website) are two documents that define all the behavior of a USB keyboard.

How the VGA Monitor works

For detailed explanation on how the VGA monitor works, please refer to the lecture slides and section 4.10 of the DE2-115 User Guide available on the ECE 385 website. Additional information can also be found by doing a web search.

Some sample codes are given on the ECE 385 website which generates the horizontal sync, vertical sync, horizontal pixel and vertical pixel location.

Instructions for the lab

The goal of this circuit is to make a small red ball move on the white VGA monitor screen. The red ball can either move in the X (horizontal) direction or the Y (vertical) direction. (Remember that on the monitor, Y=0 is the top and Y=479 is the bottom!)

When the program starts, a stationary red ball should be displayed in the center of the screen. The ball should be waiting for a direction signal from the keyboard. As soon as a direction key (W-A-S-D) is pressed on the keyboard, the ball will start moving in the direction specified by the key.

W - Up
S - Down
A - Left
D - Right

When the ball reaches the edge of the screen, it should bounce back and start moving in the opposite direction.

The ball will keep moving and bouncing until another command is received from the keyboard. When a different direction key is pressed, the ball should start moving in the newly specified direction immediately, without returning to the center of the screen. NOTE: The ball should never move diagonally, and once set into motion by the initial key press, should never come to a stop.

Sample SystemVerilog code for the ball is given on ECE 385 web site. The sample code only implements the bouncing of the ball in the Y direction. You have to add support for motion in the X direction and response to keyboard input.

Please do not take this lab lightly! Working with the VGA and Keyboard is a big time sink.

Summarizing, complete working code for a ball, moving and bouncing in the Y direction, can be found on the ECE 385 website. You have to add the following features:

- A keyboard entity that outputs the code of the last received key (Completed USB tutorial with additional logic)
- Motion and bouncing in the X and Y direction
- Immediately changing the ball's motion using the direction keys (W,A,S,D) (The ball should respond to the scan code)
- All of these functions should work in any sequence without having to reset the circuit

As this can be a daunting task, you may take the following steps to receive partial credit:

- Display the last received key (scan code) from the keyboard (1 points)
- Somehow show that the ball can move in X as well as Y direction without reprogramming the FPGA (perhaps by using switch inputs as stimulus) (1 points)
- Somehow show that your keyboard code can identify the 4 different directions and light a different LED for each direction (1 points)
- Somehow show that the ball can bounce when moving in the X as well as Y direction without reprogramming the FPGA. Ball does not move diagonally (1 points)

Your top-level circuit should have **at least** the following inputs and outputs:

General Interface:

Inputs

```
Reset          : logic          -- For any initialization purposes
Clk            : logic          -- 50 MHz clock input
HEX0, HEX1     : logic [6:0]    -- Makecode output in HEX
```

VGA Interface:

Outputs

```
Red            : logic [7:0]    -- Red color output to the VGA
Green          : logic [7:0]    -- Green color output to the VGA
Blue           : logic [7:0]    -- Blue color output to the VGA
VGA_clk        : logic;         -- 25 MHz pixel clock for DAC chip
sync           : logic;         -- Sync signal for DAC chip
blank          : logic;         -- Blanking signal for DAC chip
vs             : logic;         -- Vertical Sync Signal to the VGA
hs             : logic;         -- Horizontal Sync Signal to the VGA
```

CY7C67200 Interface:

Inputs

```
OTG_INT        : logic          -- CY7C67200 Interrupt
```

Bidirectional ports (inout)

```
OTG_DATA       : logic [15:0]   -- CY7C67200 Data Bus 16 Bits
```

Outputs

```
OTG_ADDR       : logic [1:0]    -- CY7C67200 Address 2 Bits
OTG_CS_N       : logic          -- CY7C67200 Chip Select
OTG_RD_N       : logic          -- CY7C67200 Read
OTG_WR_N       : logic          -- CY7C67200 Write
```

OTG_RST_N : logic -- CY7C67200 Reset

SDRAM Interface for Nios II Software:

Bidirectional ports (inout)

sdram_wire_dq : logic [31:0] -- SDRAM Data 32 Bits

Outputs

sdram_wire_addr : logic [12:0] -- SDRAM Address 13 Bits
sdram_wire_ba : logic [1:0] -- SDRAM Bank Address 2 Bits
sdram_wire_dqm : logic [3:0] -- SDRAM Data Mast 4 Bits
sdram_wire_ras_n : logic; -- SDRAM Row Address Strobe
sdram_wire_cas_n : logic; -- SDRAM Column Address Strobe
sdram_wire_cke : logic; -- SDRAM Clock Enable
sdram_wire_we_n : logic; -- SDRAM Write Enable
sdram_wire_cs_n : logic; -- SDRAM Chip Select
sdram_clk : logic; -- SDRAM Clock

NOTE: For the partial credit, you may add LEDs, hex displays, switches, and/or buttons to the above lists.

III. PRE-LAB

A. Download the incomplete Quartus project for Lab 8 on the ECE 385 course website. Follow the IUQ tutorial to complete the software USB protocol by performing the tasks as described in the tutorial. Follow the VGA tutorial and the provided codes on the ECE 385 course website to complete the VGA display.

You will need to bring the following to the lab:

1. Your code for the Lab. You can bring the code to the lab on a USB storage device, floppy disk, CD, FTP or using any other method.

2. Block diagram of your design with components, ports, and interconnections labeled.

IV. LAB

Follow the Lab 8 demo information on the course website.

V. POST-LAB

1.) Refer to the Design Resources and Statistics in IQT.29-31 and complete the following design statistics table.

LUT	
DSP	
Memory (BRAM)	
Flip-Flop	
Frequency	
Static Power	
Dynamic Power	
Total Power	

Document any problems you encountered and your solutions to them, and write a short conclusion.

Before you leave from your lab session submit your latest project code including both the .sv files and the usb_software code to yout TA on his/her USB drive. TAs are under no obligation to accept late code, code that doesn't compile (unless you got 0 demo points) or code files that are intermixed with other project files.

VI. REPORT

In your lab report, you should include but not limited to the following:

- An introduction;
- Written description of the operation of your circuit;
- Written purpose and operation of each module, including the inputs/outputs of the modules;
- Written description of the USB protocol and the required changes to the provided files.
- Block diagram with components, ports, and interconnections labeled;
- Answers to post-lab questions;
- A conclusion regarding what worked and what didn't, with explanations of any possible causes and the potential remedies.