Lab 1: Verilog and FPGA Refresher

Objectives:

The objective of this lab is to review some Verilog and FPGA basics and to familiarize you with Vivado Design Suite and the Nexys A7 Trainer Board.

This lab is organized as follows:

Introduction

Verilog is a hardware description language used to model digital circuits. Digital circuits described by Verilog are typically simulated and then implemented either on an FPGA or as an ASIC. In this lab we are going to review Verilog through modeling a set of simple circuits and implementing them on the Nexys A7 board. The Nexys A7 is manufactured by Digilent and is based on Xilinx Artix-7 FPGA. The Nexys A7 can be programmed via the Vivado Design Suite which is a full featured software suite designed by Xilinx too. Vivado is the successor of Xilinx ISE and it encapsulates many features including a simulator, a synthesizer, bitstream generator, and an FPGA programmer.

There are many freely-available resources that you can use to learn more about Verilog, Vivado, and Nexys A7. The following resources are particularly recommended:

Verilog:
http://www.asic-world.com/verilog/veritut.html

Vivado:
https://www.xilinx.com/support/documentation/university/Vivado-Teaching/HDL-Design/2013x/Nexys4/Verilog/docs-pdf/Vivado_tutorial.pdf
and
https://reference.digilentinc.com/vivado/getting_started/start

Nexys A7 reference manual:
https://reference.digilentinc.com/reference/programmable-logic/nexys-a7/start
https://reference.digilentinc.com/reference/programmable-logic/nexys-a7/reference-manual

Experiments:

Note: Students should be working in pairs to implement the following experiments

Experiment 1: A simple inverter

a. Create a new RTL project targeting the Nexys A7 board using Vivado. Make sure that you select the right target board when you create the project. If Nexys A7 is not available, it should be installed before proceeding any further.
   - To install Nexys A7, download and extract this Zip file: https://github.com/Digilent/vivado-boards/archive/master.zip
   - Open file called new/board_files.
   - Copy the contents of the board_files folder.
   - Paste into the board_files folder in the Vivado Installation directory (C:\Xilinx\Vivado\2016.4\data\boards\board_files).
   - Restart Vivado.

b. Add a new Verilog module representing a simple inverter. The module should have a 1-bit input port named A and a 1-bit output port named B
   - Filename: inverter.v and module name should be inverter

c. Create a testbench for the inverter following the steps in the Vivado tutorial
   - Filename: inverter_tb.v and module name should be inverter_tb

d. Simulate the testbench

e. Create a constraint file attaching input port A to switch 0 (pin J15) and output port B to Led 0 (pin H17). The constraint file should contain the following 4 lines:

```
set_property package_pin J15 [get_ports A]
set_property iostandard LVCMOS33 [get_ports A]
set_property package_pin H17 [get_ports B]
set_property iostandard LVCMOS33 [get_ports B]
```

   Note: please use the same spacing as above. For example, you must leave a space after `get_ports`

   Note: instead of writing the constraints by yourself, you can use the GUI to assign pins in the IO pin Planning Layout, check Vivado Tutorial for the details.

f. Synthesize, implement, and generate a bit stream for your design.

g. Use the generated bitstream to program the Nexys A7 board and check that the board behaves as expected. Make sure to connect the board to your computer, to power it on, and to adjust the programming jumper to JTAG before this step.

Note: You need to show your instructor the result of this experiment within Lab.

Experiment 2: A 4-digit 7-segment display driver

a. Create a new RTL project targeting the Nexys A7 board using Vivado.
b. Design and implement a 4-digit 7-segment display driver module controlled by the 13 input switches. The Nexys A7 has 16 switches, but we choose to use only 13 switches to specify the binary value we want to visualize on the 7-segment display. Why?
    - File name: `Four_Digit_Seven_Segment_Driver.v`, top level module name: `Four_Digit_Seven_Segment_Driver`

As shown in Figure 1, The Nexys A7 board contains two four-digit common anode seven-segment LED displays, configured to behave like a single eight-digit display. All 7 segments of a digit have a common anode and each segment has individual cathode; however, as shown in Figure 2, all 4 digits share the same cathode pins. So a trick is needed to display 4 different numbers on the 4-digit seven-segment LED display. The idea is to repeatedly activate one digit at a time in a rotational pattern. So we would activate digit1, then digit2, then digit3, then digit4, then digit1 again, and so on. This of course would make each digit flicker quickly, but if it is refreshed quick enough (at a rate between 60 Hz, and 1 KHz), the human eye will not be able to notice that flickering.

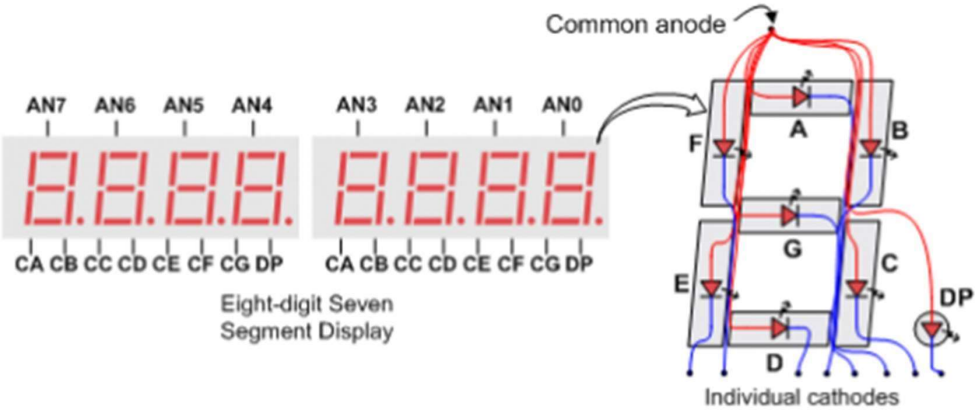

Figure 1 Common anode circuit node (reproduced from the Nexys A7 reference manual)

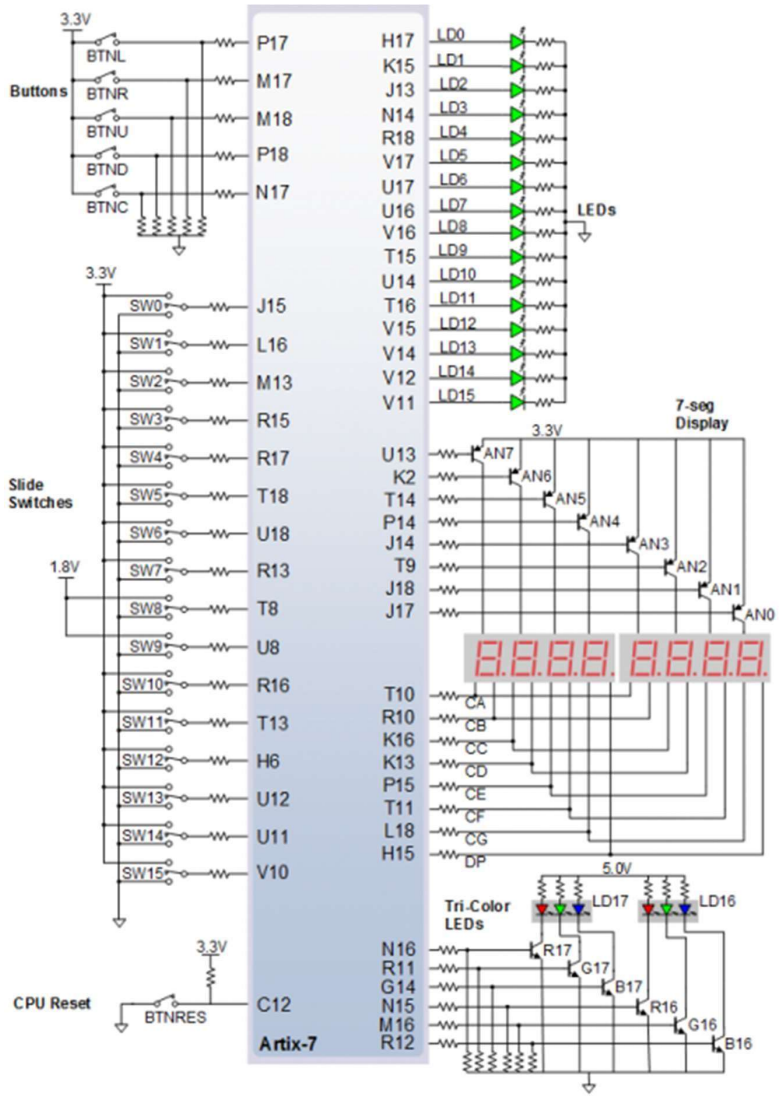

Figure 2 General purpose I/O devices on the Nexys A7 (reproduced from the Nexys A7 reference manual)

In order to do so, we make use of the built-in 100 MHz clock available on pin **E3** of the Nexys A7 package whose frequency we divide by 2^{20} using a 20-bit counter. The result will be a clock of 95.37 Hz (a period of 10.49 ms). We can further split this period into 4 equal time slots of 2.6 ms each such that each digit is activated for 2.6 ms every 10.49 ms. In order to do this final splitting we consider the most significant 2 bits of the 20 bit counter and change the active digit as soon as they change.

One possible implementation in Verilog (adapted from www.fpga4student.com) is shown next.

```
module Four_Digit_Seven_Segment_Driver (
   input clk,
   input [12:0] num,
   output reg [3:0] Anode,
   output reg [6:0] LED_out
   );

   reg [3:0] LED_BCD;
   reg [19:0] refresh_counter = 0; // 20-bit counter
   wire [1:0] LED_activating_counter;

   always @(posedge clk)
      begin
         refresh_counter <= refresh_counter + 1;
      end

   assign LED_activating_counter = refresh_counter[19:18];

   always @(*)
      begin
         case(LED_activating_counter)
         2'b00: begin
            Anode = 4'b0111;
            LED_BCD = num/1000;
            end
         2'b01: begin
            Anode = 4'b1011;
            LED_BCD = (num % 1000)/100;
            end
         2'b10: begin
            Anode = 4'b1101;
            LED_BCD = ((num % 1000)%100)/10;
            end
         2'b11: begin
            Anode = 4'b1110;
            LED_BCD = ((num % 1000)%100)%10;
            end
         endcase
      end

    always @(*)
       begin
          case(LED_BCD)
             4'b0000: LED_out = 7'b0000001; // "0"
             4'b0001: LED_out = 7'b1001111; // "1"
             4'b0010: LED_out = 7'b0010010; // "2"
             4'b0011: LED_out = 7'b0000110; // "3"
             4'b0100: LED_out = 7'b1001100; // "4"
             4'b0101: LED_out = 7'b0100100; // "5"
             4'b0110: LED_out = 7'b0100000; // "6"
             4'b0111: LED_out = 7'b0001111; // "7"
             4'b1000: LED_out = 7'b0000000; // "8"
             4'b1001: LED_out = 7'b0000100; // "9"
             default: LED_out = 7'b0000001; // "0"
          endcase
       end
 endmodule
```

c. Create a constraint file attaching:
   - The inputs of the module to switches 0 to 12 according to Figure 2
   - The outputs of the module to the cathodes and anodes of the 7-segment LEDs according to Figure 2
   - The 100 MHz clock input to pin E3
   - File name: `FDSSD_constraint.xdc`
d. Program the Nexys A7 board and test it
e. Record the utilization (number of LUTs, FFs, and IO ports) consumed by the driver circuit you designed. You can find this information after you have completed the implementation step in the utilization report.

Experiment 3: A 4-digit 7-segment display driver with optimized Divisor

Note: Make a New project, don't override Exp 2 files.
Repeat Experiment 2 but change the division and remainder methods to an optimized algorithm.

- File name: `Four_Digit_Seven_Segment_Driver_Optimized.v`, top level module name: `Four_Digit_Seven_Segment_Driver_Optimized`

The division (/) and remainder (%) operators are very expensive. Some tools are not even able to synthesize them (that is not the case with Vivado).
Here is a more efficient design to convert binary to BCD (called **Shift and Add-3 Algorithm**):
https://pubweb.eng.utah.edu/~nmcdonal/Tutorials/BCDTutorial/BCDConversion.html

You will need to modify the algorithm to handle thousands as well.

```
module BCD (
       input [7:0] num,
       output reg [3:0] Hundreds,
       output reg [3:0] Tens,
       output reg [3:0] Ones
       );
integer i;
always @(num)
begin
       //initialization
       Hundreds = 4'd0;
       Tens = 4'd0;
       Ones = 4'd0;
       for (i = 7; i >= 0 ; i = i-1 )
       begin
              if(Hundreds >= 5 )
                     Hundreds = Hundreds + 3;
              if (Tens >= 5 )
                     Tens = Tens + 3;
              if (Ones >= 5)
                     Ones = Ones +3;

              //shift  left  one
              Hundreds = Hundreds << 1;
              Hundreds [0] = Tens [3];
              Tens = Tens << 1;
              Tens [0] = Ones[3];
              Ones = Ones << 1;
              Ones[0] = num[i];
       end
end
endmodule
```

Deliverables:

General Report Submission Notes:

- Each student is required to submit an individual report.
- For Verilog descriptions or constraint files, please attach the .v and .xdc files (not the whole project) or copy and paste the code. No photos or screenshots accepted.
- The submission should be a zipped folder with the following Structure
    a. YourID_Report.pdf
    b. Exp1 : Folder containing your codes for experiment#1
    c. Exp2 : Folder containing your codes for experiment#2
    d. Exp3 : Folder containing your codes for experiment#3
- **Submission method**: On Blackboard

Lab Report [10 pts]

Report of Lab1 should include:

1. [0 pts] Your names, student IDs, Team Name.
2. [2 pts] A technical summary of experiments conducted in the lab (Steps, Results, components (a screen shot with Schematic design might help), code functionality, etc...]
3. [4 pts] Any Verilog descriptions or constraint files you wrote for the lab.
4. [2 pts] Compare the utilization and delay of experiment 2 vs experiment 3, please add your comments about the results.
5. [2 pts] Results recorded in the last step of experiment 2 & 3 (Photos or screen shots)
