=============================================================

**Lab 2 Report**

**FPGA Implementation and Hardware Validation**

**Date \_\_\_\_2/18/17\_\_\_\_\_\_**

**by**

**Name \_\_\_\_\_\_\_Anahit Sarao\_\_\_\_ SID \_\_\_008435583\_\_\_\_**

**Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ SID \_\_\_\_\_\_\_\_\_\_\_\_\_**

**Lab Record**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Performed by (print name)** | **Checked by (print name)** | **Successfully Completed** | **Partially Completed\*** | **Failed or Not Performed\*** |
| Anahit Sarao |  | X |  |  |
|  |  |  |  |  |

**\* Detailed descriptions must be given in the report.**

**San Jose State University**

**Department of Computer Engineering**

**CMPE 125 Spring 2017**

***Introduction***

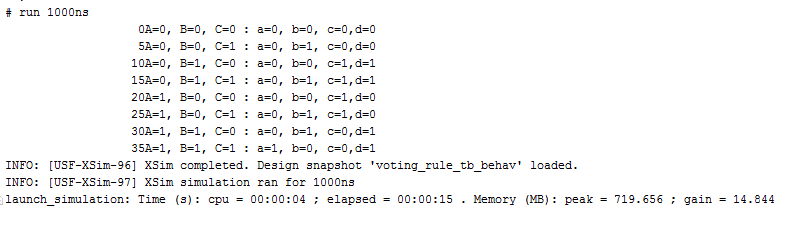
The purpose of this lab involved understanding the design and testing methodologies of developing a system, specifically, a voting machine. The design was used to validate the functionality of the Nexys 4 DDR Artix-7 FPGA. The voting machine consisted of 3 inputs that carried a unique value. The inputs are tied to the right most switches in order of value, the first switch is considered numerical two, second switch is three and third switch is four. The number was added for each combination of the switch which was outputted to the seven segment display. The Xilinx Design Tool was used to generate the source, test and xdc code in the following workflow. The design code was simulated synthesized and lastly a bit stream was generated which allowed the hardware manager to reprogram the FPGA for testing and validation.

***Design Methodology***

The source code involves multiple modules that connect to each other, forming a voting machine configuration. It is a continuation of lab one but introduces new topics on VHDL written and compiled with XDT. The modules sources for this lab contain the code to properly set up the weight for each switch, count the representation of the switches position and display it upon the seven segment display. This includes four modules under the voting\_machine\_fgpa file provided in the Appendix.

The first module voting\_rule which defines the input and output logic. It makes a, b, c as inputs and gives them the proper values while making d as an output. The modules function is to take in any BCD input and pass it to the next module which is the bcd\_to\_7seg module. It is responsible for configuring the BCD input to the 7 segment display using a clock speed defined by the clk\_gen module. The last component is the led\_mux which directely controls the LEDs inside the display. The gluing module voting\_rule can be considered the top level module with two child modules bcd\_t\_seg and led\_mux, with a definition module for clock speeds clk\_gen.

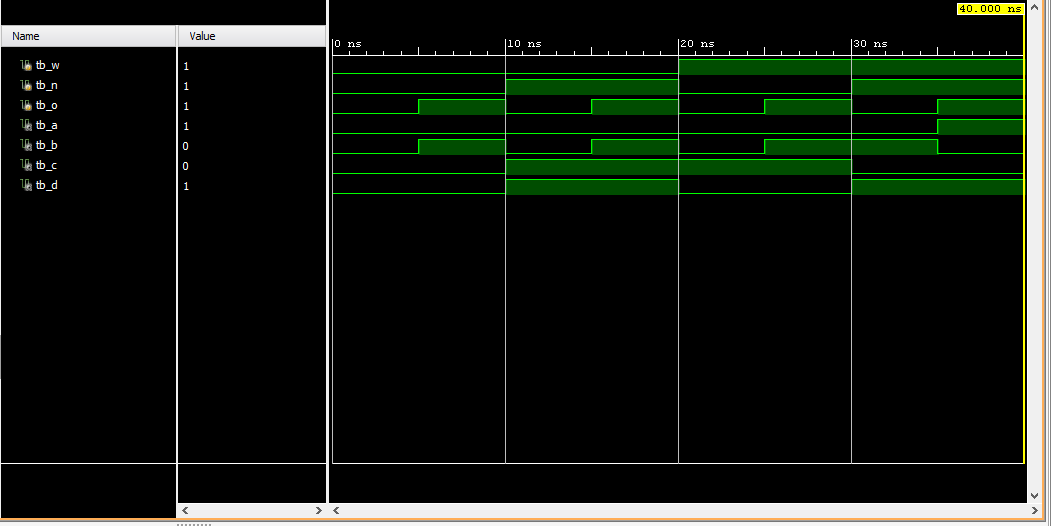
Once the syntax check was completed a constraints file was created to attach the modules to physical parts on the FPGA such as switches and displays. After a synthesis of the constraints was completed a bit stream was fed using the JTAG port on the FPGA to run the code. Below is the sample output of the console during simulation state it generated a truth table of for the modules defined.



*Figure 1. Truth table output during simulation stage*

***Simulation Test Plan***

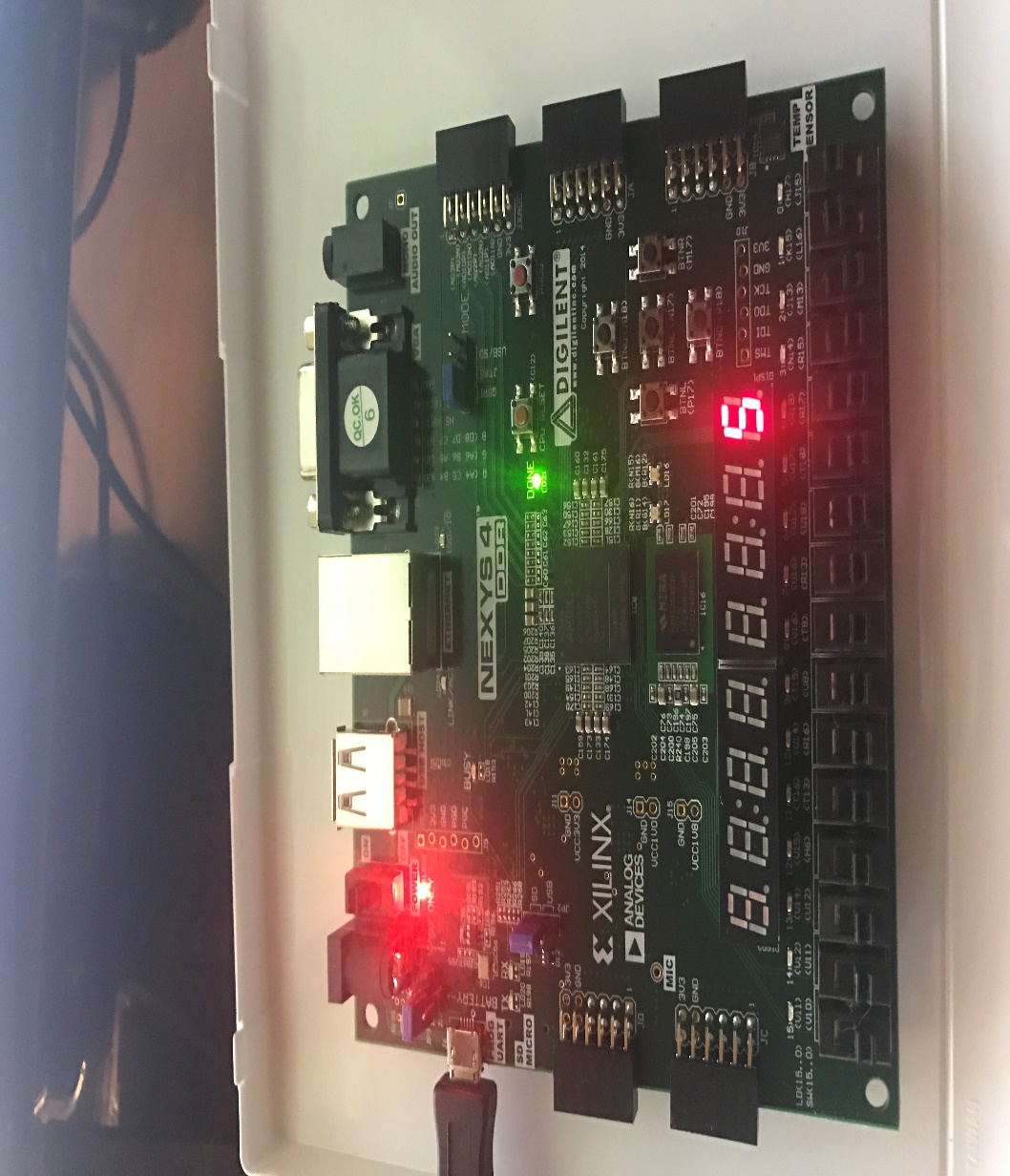
To validate the simulations generated based of the truth table certain min terms were used such as if the inputs were 0,1,1 the output would be assumed to be 0,1,1,1. This simple test can be defined using the truth table and validated against the waveforms. The waveforms come from the simulation stage which shows zeros and ones as highs and lows per input per output. Figure two is the actual waveform output which was validated using the truth table in figure one.



*Figure 2. Simulation waveforms used to test code.*

***FPGA Test Plan***

After the hardware manager uploaded the code onto the FPGA a similar testing method was used to test if the constraints file and modules worked as expected. As explained above the right outer most switches were utilized, that means any switch that changes its state will trigger its value upon of display. If one or more switches are enabled the result will be properly adjusted for each unique combination. For a simple test the switches were set to replicate 0,1,1. Which according to the waveform and truth table should display five. The figure three attached below shows this test being conducted and being successful.



*Figure 3. FPGA testing of sequence 0-1-1*

***Conclusion***

The function of this lab was to explain the workflow and importance of major step using XDE. From creating sources and test benches all the way to generating the bit stream and being able to implement the logical design virtually to a physical device within seconds is very complex process which needs to be understood to be able to fully recreate modules from scratch. The lab was demo was conducted by the lab instructor after showing the results a successfully completed notion was received. Some warnings came up during the execution of the lab, the instructor was made aware in which it was advised to restart the computer as it may have to do with boards connection to the computer. However, the warning can be deemed ignorable as it did not affect the lab results.

***Appendix***

C:\Users\india\Downloads\Chrome-DL\Untitled Diagram (2).png

*Figure 4. Block Diagram of lab modules*

1. Source Code: voting\_machine\_fpga

module voting\_machine\_fpga (

input A,

input B,

input C,

input clk100MHz,

input rst,

output [7:0] LEDOUT,

output [7:0] LEDSEL);

supply1[7:0] vcc;

wire s0, s1, s2, s3, s4, s5, s6, s7;

wire [3:0] i;

wire DONT\_USE, clk\_5KHz;

assign s7 = 1'b1;

// instantiation and connect the sub modules

voting\_rule u0(A,B,C,i[0],i[1],i[2],i[3]);

bcd\_to\_7seg u1({i[0],i[1],i[2],i[3]},s0,s1,s2,s3,s4,s5,s6);

led\_mux u2(clk\_5KHz, rst, vcc, vcc, vcc, vcc, vcc, vcc, vcc,

{s7, s6, s5, s4, s3, s2, s1, s0}, LEDOUT, LEDSEL);

clk\_gen u3 (.clk100MHz(clk100MHz), .rst(rst),

.clk\_4sec(DONT\_USE), .clk\_5KHz(clk\_5KHz));

// Signal DONT\_USE will cause some warnings during Synthesis.

// You can ignore the warnings.

endmodule // end voting\_machine\_fpga

1. Source Code: voting\_rule

module voting\_rule(w,n,o,a,b,c,d);

input w,n,o;

output a,b,c,d;

assign a = w&n&o;

assign b = ~w&~n&o | ~w&n&o | w&~n&o | w&n&~o;

assign c = ~w&n&~o | ~w&n&o | w&~n&~o | w&~n&o;

assign d = ~w&n&~o | ~w&n&o | w&n&~o | w&n&o;

endmodule // end voting\_rule

1. Source Code: clk\_gen

module clk\_gen(clk100MHz, rst, clk\_4sec, clk\_5KHz);

input clk100MHz, rst;

output clk\_4sec, clk\_5KHz;

reg clk\_4sec, clk\_5KHz;

integer count, count1;

always@(posedge clk100MHz)

begin

if(rst)

begin

count = 0;

count1 = 0;

clk\_4sec = 0;

clk\_5KHz =0;

end

else

begin

if(count == 200000000)

begin

clk\_4sec = ~clk\_4sec;

count = 0;

end

if(count1 == 10000)

begin

clk\_5KHz = ~clk\_5KHz;

count1 = 0;

end

count = count + 1;

count1 = count1 + 1;

end

end

endmodule // end clk\_gen

1. Source Code: led\_mux

module led\_mux (

input wire clk,

input wire rst,

input wire [7:0] LED0, // leftmost digit

input wire [7:0] LED1,

input wire [7:0] LED2,

input wire [7:0] LED3,

input wire [7:0] LED4,

input wire [7:0] LED5,

input wire [7:0] LED6,

input wire [7:0] LED7, // rightmost digit

output wire [7:0] LEDSEL,

output wire [7:0] LEDOUT

);

reg [2:0] index;

reg [15:0] led\_ctrl;

assign {LEDOUT, LEDSEL} = led\_ctrl;

always@(posedge clk)

begin

index <= (rst) ? 3'd0 : (index + 3'd1);

end

always @(index, LED0, LED1, LED2, LED3, LED4, LED5, LED6,

LED7)

begin

case(index)

3'd0: led\_ctrl <= {8'b11111110, LED7};

3'd1: led\_ctrl <= {8'b11111101, LED6};

3'd2: led\_ctrl <= {8'b11111011, LED5};

3'd3: led\_ctrl <= {8'b11110111, LED4};

3'd4: led\_ctrl <= {8'b11101111, LED3};

3'd5: led\_ctrl <= {8'b11011111, LED2};

3'd6: led\_ctrl <= {8'b10111111, LED1};

3'd7: led\_ctrl <= {8'b01111111, LED0};

default: led\_ctrl <= {8'b11111111, 8'hFF};

endcase

end

endmodule

1. Source Code: bcd\_to\_7seg

module bcd\_to\_7seg(BCD, s0, s1, s2, s3, s4, s5, s6);

output s0, s1, s2, s3, s4, s5, s6;

input [3:0] BCD;

reg s0, s1, s2, s3, s4, s5, s6;

always @ (BCD)

begin // BCD to 7-segment decoding

case (BCD) // s0 – s6 are active low

4'b0000: begin s0=0; s1=0; s2=0; s3=1; s4=0; s5=0; s6=0; end

4'b0001: begin s0=1; s1=0; s2=1; s3=1; s4=0; s5=1; s6=1; end

4'b0010: begin s0=0; s1=1; s2=0; s3=0; s4=0; s5=1; s6=0; end

4'b0011: begin s0=0; s1=0; s2=1; s3=0; s4=0; s5=1; s6=0; end

4'b0100: begin s0=1; s1=0; s2=1; s3=0; s4=0; s5=0; s6=1; end

4'b0101: begin s0=0; s1=0; s2=1; s3=0; s4=1; s5=0; s6=0; end

4'b0110: begin s0=0; s1=0; s2=0; s3=0; s4=1; s5=0; s6=0; end

4'b0111: begin s0=1; s1=0; s2=1; s3=1; s4=0; s5=1; s6=0; end

4'b1000: begin s0=0; s1=0; s2=0; s3=0; s4=0; s5=0; s6=0; end

4'b1001: begin s0=0; s1=0; s2=1; s3=0; s4=0; s5=0; s6=0; end

default: begin s0=1; s1=1; s2=1; s3=1; s4=1; s5=1; s6=1; end

endcase

end

endmodule // end bcd\_to\_7seg

1. Test Bench

// testbench for module voting\_rule

module voting\_rule\_tb;

reg tb\_w, tb\_n, tb\_o;

wire tb\_a, tb\_b, tb\_c, tb\_d;

//instantiate the module voting\_rule

voting\_rule DUT (tb\_w, tb\_n, tb\_o, tb\_a, tb\_b, tb\_c, tb\_d);

//all initial blocks begin at time=0 concurrently

initial

begin

tb\_o = 0;

forever #5 tb\_o = ~tb\_o;

end

initial

begin

tb\_n = 0;

forever #10 tb\_n = ~tb\_n;

end

initial

begin

tb\_w = 0;

forever #20 tb\_w = ~tb\_w;

end

//stop at time=40

initial

#40 $stop;

initial

$monitor($time, "A=%b, B=%b, C=%b : a=%b, b=%b, c=%b,d=%b",tb\_w, tb\_n, tb\_o, tb\_a, tb\_b, tb\_c, tb\_d);

endmodule

1. Constraints File

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk100MHz }];

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk100MHz}];

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { A }];

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { B }];

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { C }];

set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[0] }];

set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[1] }];

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[2] }];

set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[3] }];

set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[4] }];

set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[5] }];

set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[6] }];

set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[7] }];

set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[0] }];

set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[1] }];

set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[2] }];

set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[3] }];

set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[4] }];

set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[5] }];

set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[6] }];

set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[7] }];

set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports { rst }];