San Jose State University
Department of Electrical Engineering
EE178, Fall 2016, Crabill

Laboratory Assignment #2

Objectives

This lab practices logic design using familiar building blocks. From your previous coursework, you should already be familiar with simple counters, multiplexers, decoders, and seven-segment displays. The goal of this lab is for you to implement a circuit that drives the time-multiplexed quad seven-segment display present on the Digilent Basys3 board. When you successfully complete this lab, you will have developed a piece of intellectual property that you might be able to re-use in the future.

Figure 1: A Quad Seven-Segment Display

Now that you are familiar with the tools from Laboratory Assignment #1, you should be able to concern yourself with digital design. Figure 2 shows a symbol of the design you will create. The inputs are shown on the left and the outputs are shown on the right.

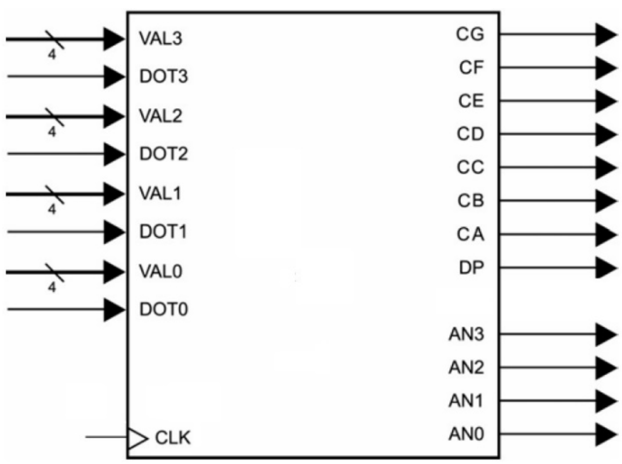

Figure 2: A Symbol of the Design

Bibliography

This lab draws heavily from documents on the Digilent website http://www.digilentinc.com. I would like to thank Digilent for making this material available.

Time-Multiplexed Quad Seven-Segment Display

The Digilent Basys3 board has a time-multiplexed octal seven-segment display. Please read the Basys3 FPGA Board Reference Manual. You will find detailed information about this display in the introductory portion of Chapter 8, “Basic I/O”, and Section 8.1, “Seven-Segment Display”.

Each digit shares eight common control signals to light individual segments. Each individual digit has a separate anode control input. All of these control signals are active low. Think of each digit as having a separate enable; that enable signal is the anode control. To enable any given digit, drive its anode control signal low. Enabled digits will display the segments selected by the eight active low segment controls.

It is important to remember that all digits share the same segment controls. For example, if you were to drive all anode control signals low, and apply values to the eight segment controls, the same pattern would appear on all digits. That's not very useful, because all it does is generate the same pattern on all digits. In order to get a unique pattern on each of the digits, you must apply a technique called time multiplexing:

1. Assert anode control for only digit 0 and apply unique 8-bit segment controls for digit zero.
2. Assert anode control for only digit 1 and apply unique 8-bit segment controls for digit one.
3. Assert anode control for only digit 2 and apply unique 8-bit segment controls for digit two.
4. Assert anode control for only digit 3 and apply unique 8-bit segment controls for digit three.

If you repeat this slowly, you can watch each digit light up in turn. If you increase the rate at which you do this, at some point it will cease to look like the sequential illumination of individual digits, and begin to look like all digits are illuminated at the same time. If you continue to increase the rate, at some point the display intensity will drop due to analog effects as the segments are not given enough time to fully turn on.

Design Description and Requirements

In this design, you are not allowed to use latches. You are allowed to use only one clock. The clock must be the 100 MHz clock signal available from the oscillator on the board. You will receive zero points if you do not follow these requirements.

As shown in Figure 2, the design has a number of inputs. There is a clock input, plus a pair of inputs for each digit. Each pair consists of a four-bit binary value with a one-bit decimal point control.

clk	clock signal, 100 MHz from oscillator
val3[3:0]	value for left-most quad display digit, digit 3
dot3	active high control for decimal point for left-most quad display digit, digit 3
val2[3:0]	value for left-center quad display digit, digit 2
dot2	active high control for decimal point for left-center quad display digit, digit 2
val1[3:0]	value for right-center quad display digit, digit 1
dot1	active high control for decimal point for right-center quad display digit, digit 1
val0[3:0]	value for right-most quad display digit, digit 0
dot0	active high control for decimal point for right-most quad display digit, digit 0

Also shown in Figure 2 are the two groups of output signals: the four anode control signals and the eight segment control signals. All of these signals are active low.

an3...an0	anode control for left-most display digit (digit 3), thru right-most display digit (digit 0)
ca...cg	control for segment a, through segment g
dp	control for segment dp

The design must drive the segment and anode control signals to generate a display that represents the values applied to the inputs. Each digit must be capable of displaying the 16 possible values: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The display must be bright and not exhibit excessive flickering. Decimal points are to be illuminated when the control input is high – pressing a button illuminates a decimal point.

Describing the Design

Before you begin writing any code, you must sit down with scratch paper and draw a block diagram of a circuit that will satisfy the design requirements. Once you have a possible solution, write a description of it in Verilog-HDL and proceed to test it in simulation. Use the following template for your design. You may change the declared port data types if you need to suit your design description:

```
// File: quad_seven_seg.v
// This is the top level design for EE178 Lab #2.

// The `timescale directive specifies what the
// simulation time units are (1 ns here) and what
// the simulator time step should be (1 ps here).

`timescale 1 ns / 1 ps

// Declare the module and its ports. This is
// using Verilog-2001 syntax.

module quad_seven_seg (
  input wire clk,
  input wire [3:0] val3,
  input wire dot3,
  input wire [3:0] val2,
  input wire dot2,
  input wire [3:0] val1,
  input wire dot1,
  input wire [3:0] val0,
  input wire dot0,
  output reg an3,
  output reg an2,
  output reg an1,
  output reg an0,
  output reg ca,
  output reg cb,
  output reg cc,
  output reg cd,
  output reg ce,
  output reg cf,
  output reg cg,
  output reg dp
  );

  // Describe the actual circuit for the assignment.

endmodule
```

To facilitate re-use of your completed design, you must implement it in this single module – you are not allowed to use hierarchical design with sub-modules for this assignment. If you have further questions, or need clarification, consult the instructor.

Testing the Design

You must perform some minimal functional simulation of the design. This is important for two reasons. First, it will give you confidence your design is working properly before you implement it. Second, if the design does not behave as expected when you download it, you will have a mechanism to quickly create additional test cases to help debug the problem. The instructor will not help you debug logic problems (incorrect design behavior) unless you have a block diagram and are able to run a simulation.

In order to help you get started, here is a template for a test bench that works with the design. Feel free to enhance this as you see fit:

```
// File: testbench.v
// This is a top level testbench for the
// quad_seven_seg design, which is part of
// the EE178 Lab #2 assignment.

// The `timescale directive specifies what the
// simulation time units are (1 ns here) and what
// the simulator time step should be (1 ps here).

`timescale 1 ns / 1 ps

module testbench;

  // Declare wires to be driven by the outputs
  // of the design, and regs to drive the inputs.
  // The testbench will be in control of inputs
  // to the design, and will check the outputs.
  // Then, instantiate the design to be tested.

  wire an3, an2, an1, an0;
  wire ca, cb, cc, cd, ce, cf, cg, dp;
  reg [3:0] val3, val2, val1, val0;
  reg dot3, dot2, dot1, dot0;
  reg clk;

  // Instantiate the quad_seven_seg module.

  quad_seven_seg my_quad (
    .clk(clk),
    .val3(val3),
    .dot3(dot3),
    .val2(val2),
    .dot2(dot2),
    .val1(val1),
    .dot1(dot1),
    .val0(val0),
    .dot0(dot0),
    .an3(an3),
    .an2(an2),
    .an1(an1),
    .an0(an0),
    .ca(ca),
    .cb(cb),
    .cc(cc),
    .cd(cd),
    .ce(ce),
    .cf(cf),
    .cg(cg),
    .dp(dp)
  );

  // Describe a process that generates a clock
  // signal. The clock is 100 MHz.
```

```
  always
  begin
    clk = 1'b0;
    #5;
    clk = 1'b1;
    #5;
  end

  // Assign values to the input signals and
  // check the output results. This template
  // is meant to get you started, you can modify
  // it as you see fit. If you simply run it as
  // provided, you will need to visually inspect
  // the output waveforms to see if they make
  // sense...

  initial
  begin
    $display("If simulation ends before the testbench");
    $display("completes, use the menu option to run all.");
    // This should get "0 1.2 3." on the display.
    val3 <= 4'h0;
    dot3 <= 1'b0;
    val2 <= 4'h1;
    dot2 <= 1'b1;
    val1 <= 4'h2;
    dot1 <= 1'b0;
    val0 <= 4'h3;
    dot0 <= 1'b1;
    $display("Prepare to wait a long time...");
    #5000000;
    $display("Checkpoint, simulation time is %t",$time);
    #5000000;
    $display("Checkpoint, simulation time is %t",$time);
    #5000000;
    $display("Checkpoint, simulation time is %t",$time);
    #5000000;
    $display("Checkpoint, simulation time is %t",$time);
    #5000000;
    $display("Checkpoint, simulation time is %t",$time);
    // End the simulation.
    $display("Simulation is over, check the waveforms.");
    $stop;
  end

endmodule
```

Synthesizing the Design

Synthesize your design exactly as you did in the tutorial. Do not forget to check the reports. As a general practice, you will want to review all errors and warnings. These point to areas of concern that you should either address or justify.

Implementing the Design

Before you implement your design, you will need to add a constraints file. The following constraints are similar in nature to those used in the tutorial, with one new type of constraint for the clock input:

```
# Constraints for CLK
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -name external_clock -period 10.00 [get_ports clk]

# Constraints for SW0
set_property PACKAGE_PIN V17 [get_ports {val0[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val0[0]}]

# Constraints for SW1
set_property PACKAGE_PIN V16 [get_ports {val0[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val0[1]}]

# Constraints for SW2
set_property PACKAGE_PIN W16 [get_ports {val0[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val0[2]}]

# Constraints for SW3
set_property PACKAGE_PIN W17 [get_ports {val0[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val0[3]}]

# Constraints for BTNU
set_property PACKAGE_PIN T18 [get_ports dot0]
set_property IOSTANDARD LVCMOS33 [get_ports dot0]

# Constraints for SW4
set_property PACKAGE_PIN W15 [get_ports {val1[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val1[0]}]

# Constraints for SW5
set_property PACKAGE_PIN V15 [get_ports {val1[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val1[1]}]

# Constraints for SW6
set_property PACKAGE_PIN W14 [get_ports {val1[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val1[2]}]

# Constraints for SW7
set_property PACKAGE_PIN W13 [get_ports {val1[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val1[3]}]

# Constraints for BTNR
set_property PACKAGE_PIN T17 [get_ports dot1]
set_property IOSTANDARD LVCMOS33 [get_ports dot1]

# Constraints for SW8
set_property PACKAGE_PIN V2 [get_ports {val2[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val2[0]}]

# Constraints for SW9
set_property PACKAGE_PIN T3 [get_ports {val2[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val2[1]}]

# Constraints for SW10
```

```
set_property PACKAGE_PIN T2 [get_ports {val2[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val2[2]}]

# Constraints for SW11
set_property PACKAGE_PIN R3 [get_ports {val2[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val2[3]}]

# Constraints for BTND
set_property PACKAGE_PIN U17 [get_ports dot2]
set_property IOSTANDARD LVCMOS33 [get_ports dot2]

# Constraints for SW12
set_property PACKAGE_PIN W2 [get_ports {val3[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val3[0]}]

# Constraints for SW13
set_property PACKAGE_PIN U1 [get_ports {val3[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val3[1]}]

# Constraints for SW14
set_property PACKAGE_PIN T1 [get_ports {val3[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val3[2]}]

# Constraints for SW15
set_property PACKAGE_PIN R2 [get_ports {val3[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {val3[3]}]

# Constraints for BTNL
set_property PACKAGE_PIN W19 [get_ports dot3]
set_property IOSTANDARD LVCMOS33 [get_ports dot3]

# Constraints for CA
set_property PACKAGE_PIN W7 [get_ports {ca}]
set_property IOSTANDARD LVCMOS33 [get_ports {ca}]

# Constraints for CB
set_property PACKAGE_PIN W6 [get_ports {cb}]
set_property IOSTANDARD LVCMOS33 [get_ports {cb}]

# Constraints for CC
set_property PACKAGE_PIN U8 [get_ports {cc}]
set_property IOSTANDARD LVCMOS33 [get_ports {cc}]

# Constraints for CD
set_property PACKAGE_PIN V8 [get_ports {cd}]
set_property IOSTANDARD LVCMOS33 [get_ports {cd}]

# Constraints for CE
set_property PACKAGE_PIN U5 [get_ports {ce}]
set_property IOSTANDARD LVCMOS33 [get_ports {ce}]

# Constraints for CF
set_property PACKAGE_PIN V5 [get_ports {cf}]
set_property IOSTANDARD LVCMOS33 [get_ports {cf}]

# Constraints for CG
set_property PACKAGE_PIN U7 [get_ports {cg}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {cg}]

# Constraints for DP
set_property PACKAGE_PIN V7 [get_ports dp]
set_property IOSTANDARD LVCMOS33 [get_ports dp]

# Constraints for AN0
set_property PACKAGE_PIN U2 [get_ports {an0}]
set_property IOSTANDARD LVCMOS33 [get_ports {an0}]

# Constraints for AN1
set_property PACKAGE_PIN U4 [get_ports {an1}]
set_property IOSTANDARD LVCMOS33 [get_ports {an1}]

# Constraints for AN2
set_property PACKAGE_PIN V4 [get_ports {an2}]
set_property IOSTANDARD LVCMOS33 [get_ports {an2}]

# Constraints for AN3
set_property PACKAGE_PIN W4 [get_ports {an3}]
set_property IOSTANDARD LVCMOS33 [get_ports {an3}]

# Constraints for CFGBVS
set_property CFGBVS VCCO [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
```

The additional constraint for the clock input gives a "name" to the clock and tells the implementation tools the period of the clock – in this case 10.00 ns, the period of a 100 MHz clock. The implementation tools are timing driven, meaning they will make choices in placement and routing in an effort to achieve a result that will run at or above the specified clock frequency.

Do not forget to check the reports. As a general practice, you will want to review all errors and warnings. If the design fails one or more timing specifications the reports will indicate this is the case.

Test your design in hardware. Does the circuit behave as you expect? If it does not, seek assistance. Once you are confident it works properly, demonstrate your final result to the instructor.

Laboratory Hand-In Requirements

Once you have completed a working design, prepare for the submission process. You are required to demonstrate a working design. Within four hours of your demonstration, you are required to submit your entire project directory in the form of a compressed ZIP archive. Use WinZIP to archive the entire project directory, and name the archive lab2_yourlastname_yourfirstname.zip. For example, if I were to make a submission, it would be lab2_crabill_eric.zip. Then email the archive to the instructor. Only WinZIP archives will be accepted.

Demonstrations must be made on or before the due date. If your circuit is not completely functional by the due date, you should turn in what you have to receive partial credit.