San Jose State University
Department of Electrical Engineering
EE178, Spring 2006, Crabill

Laboratory Assignment #2

Objectives

This lab covers simple 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 Spartan-3 Starter Kit 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.

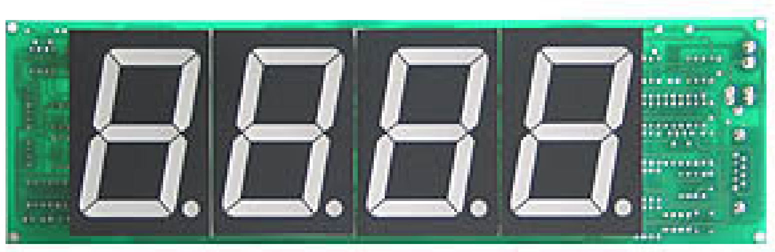


Figure 1: A Quad Seven-Segment Display

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 module you will create. The inputs are shown on the left and the outputs are shown on the right.

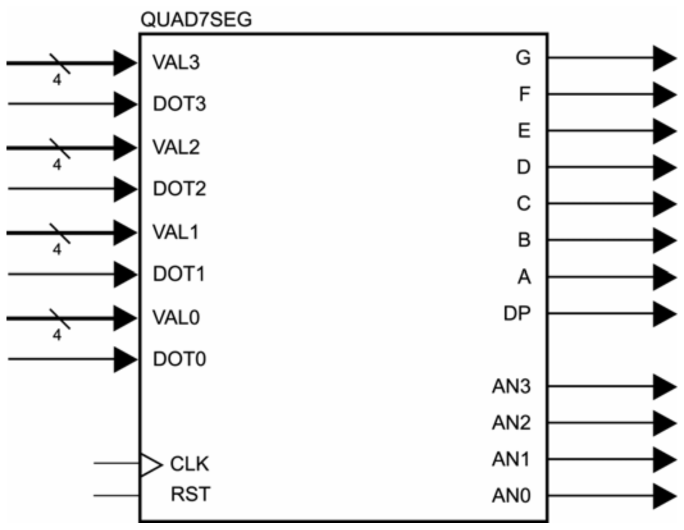


Figure 2: A Symbol of the Module

Bibliography

This lab draws heavily from the *Spartan-3 Starter Kit User Guide*. For convenience, I have reproduced portions of the text and figures related to the use of the seven-segment displays present on the Spartan-3 Starter Kit board.

Time-Multiplexed Quad Seven-Segment Display

The Spartan-3 Starter Kit board has a time-multiplexed quad seven-segment display. This display is controlled by certain pins of the FPGA. Each digit shares eight common control signals to light individual segments. Each individual digit has a separate anode control input. This is illustrated in Figure 3. The pin number for each FPGA pin is shown. All of these control signals are active low.

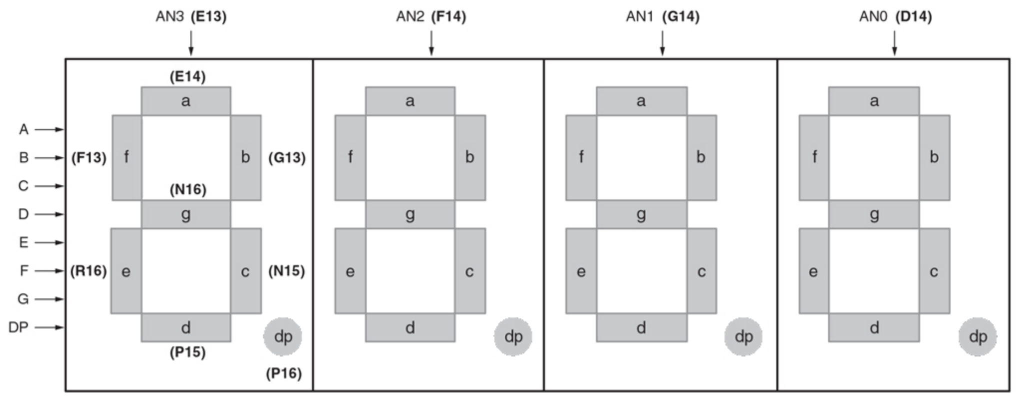


Figure 3: Time-Multiplexed Quad Seven-Segment Display

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. At any given time, you can enable none, one, two, three, or four digits. Digits that are enabled will display the segments selected by the eight active low segment controls. It is important to note that all four digits share the same segment controls.

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

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.
5. Repeat.

The first four steps are shown graphically in Figure 4 using a waveform. If you do 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. This is thanks to the persistence of vision. 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. Reasonable refresh rates for a good display quality are between 50 Hz and 2 KHz.

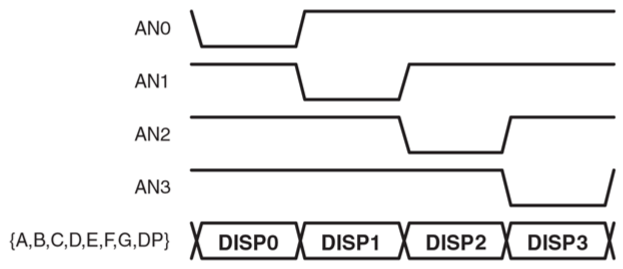


Figure 4: Time Multiplexing the Display Digits

Module Description and Requirements

In this design, you are not allowed to use latches. You are allowed to use only one clock and only one asynchronous reset signal. The clock must be the 50 MHz clock signal available from the oscillator on the Spartan-3 Starter Kit board. **You will receive zero points if you do not follow these requirements.**

As shown in Figure 2, the module has a number of inputs. There is a clock and a reset 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, 50 MHz from oscillator
rst	reset signal
val3[3:0]	value for left-most display digit, digit 3
dot3	active high control for decimal point for left-most display digit, digit 3
val2[3:0]	value for left-center display digit, digit 2
dot2	active high control for decimal point for left-center display digit, digit 2
val1[3:0]	value for right-center display digit, digit 1
dot1	active high control for decimal point for right-center display digit, digit 1
val0[3:0]	value for right-most display digit, digit 0
dot0	active high control for decimal point for right-most display digit, digit 0

Also shown in Figure 2 are the two groups of output signals. The first group is the eight segment control signals. The second group is the four anode control signals. All of these signals are active low.

a	control for segment a
b	control for segment b
c	control for segment c
d	control for segment d
e	control for segment e
f	control for segment f
g	control for segment g
an3	anode control for left-most display digit, digit 3
an2	anode control for left-center display digit, digit 2
an1	anode control for right-center display digit, digit 1
an0	anode control for right-most display digit, digit 0

The module must drive the segment and anode control signals to generate a display that represents the values applied to the inputs. The display must be bright and not exhibit excessive flickering. When the reset signal is asserted, the display must be completely blank (that is, nothing illuminated). Decimal points are to be illuminated when the control input is high – pressing a button illuminates a decimal point.

Module 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. As a hint, consider that one possible implementation of this module can be realized using a small counter, a decoder, some multiplexers, a hexadecimal to seven-segment decoder, and a handful of gates. Once you have a possible solution, write a description of it in Verilog-HDL and proceed to test it in simulation.

To facilitate re-use of your completed design, you must implement it in a 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.

Module Verification

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 module you are designing. This can be used for functional simulation as well as timing simulation after the implementation step. Feel free to enhance this basic test bench as you see fit.

```
// File:  testbench.v
// Date:  01/01/2005
// Name:  Eric Crabill
//
// This is a top level testbench for the
// quad7seg 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 timestep should be
// (1 ps here).

`timescale 1 ns / 1 ps

module quad7seg_testbench_v_tf();

  // 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 a, b, c, d, e, f, g, dp;

  reg [3:0] val3, val2, val1, val0;
  reg       dot3, dot2, dot1, dot0;
  reg       clk, rst;

  quad7seg my_quad7seg (
    .val3(val3), .val2(val2), .val1(val1), .val0(val0),
    .dot3(dot3), .dot2(dot2), .dot1(dot1), .dot0(dot0),
    .clk(clk), .rst(rst),
    .an3(an3), .an2(an2), .an1(an1), .an0(an0),
    .a(a), .b(b), .c(c), .d(d), .e(e), .f(f), .g(g), .dp(dp)
    );

  // Describe two processes that generate clock
  // and reset signals.  The clock is 50 MHz, and
  // let's assume the reset button is pressed at
  // time zero for a short period of time, then
  // released.
  always
  begin
    clk = 1'b1;
    #10;
    clk = 1'b0;
    #10;
  end

  initial
  begin
    rst = 1'b1;
    #110;
    rst = 1'b0;
  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 prematurely, restart");
    $display("using 'run -all' on the command line.");
    // This should get "3 2.1 0." on the display.
    val3 <= 4'h3;
    dot3 <= 1'b0;
    val2 <= 4'h2;
    dot2 <= 1'b1;
    val1 <= 4'h1;
    dot1 <= 1'b0;
    val0 <= 4'h0;
    dot0 <= 1'b1;
    // Wait until reset is deasserted.
    @(negedge rst);
    $display("Reset is deasserted...");
    $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
```

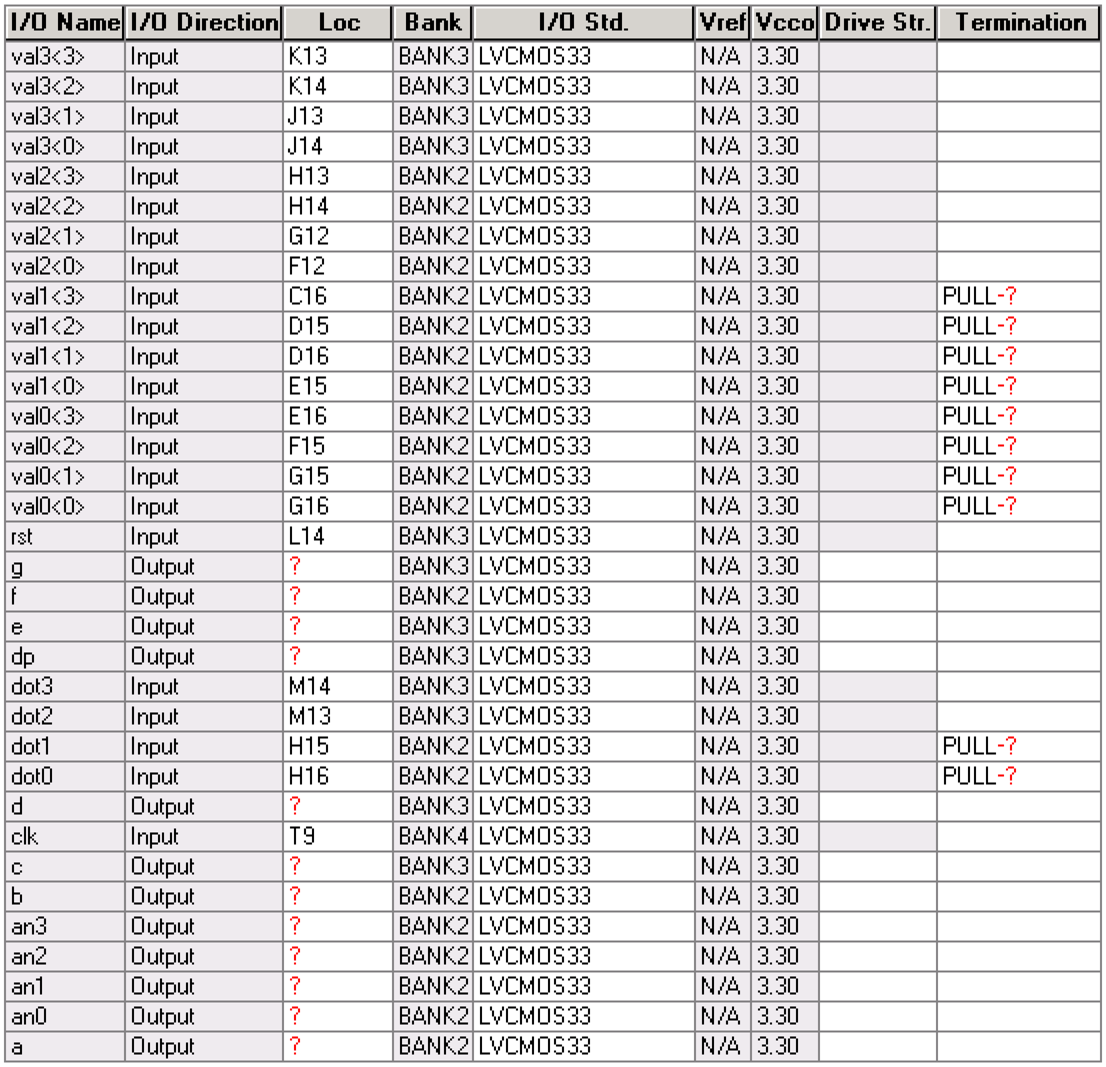
Module Synthesis

Synthesize your design exactly as you did in the tutorial. Do not forget to check the synthesis report. This report will tell you how many clocks exist in your design, under the "clock information" summary. If you have more than one clock, you need to go back and correct your design. Also check to see if any latches were used, you should not have any. These can be found in the "cell usage" summary. If you see anything starting with LD (Latch, D-type) then you need to go back and correct your design.

Module Implementation

Before you implement your design, you will need to add a constraints file and edit the I/O locations and properties. The first thing you might realize is that the Spartan-3 Starter Kit board only has eight switches and four buttons, but this design has 21 inputs, excluding the clock.

Use the details in Figure 5 as a starting point when entering your constraints. You will notice that the clock input is properly constrained, and button 3 is used as the reset. Then, button 1 and button 0, plus the eight switches, are used to control the two digits on the left. The ten remaining inputs which control the two digits on the right are assigned to I/O pins that connect to an expansion port.



I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination
val3<3>	Input	K13	BANK3	LVCMOS33	N/A	3.30		
val3<2>	Input	K14	BANK3	LVCMOS33	N/A	3.30		
val3<1>	Input	J13	BANK3	LVCMOS33	N/A	3.30		
val3<0>	Input	J14	BANK3	LVCMOS33	N/A	3.30		
val2<3>	Input	H13	BANK2	LVCMOS33	N/A	3.30		
val2<2>	Input	H14	BANK2	LVCMOS33	N/A	3.30		
val2<1>	Input	G12	BANK2	LVCMOS33	N/A	3.30		
val2<0>	Input	F12	BANK2	LVCMOS33	N/A	3.30		
val1<3>	Input	C16	BANK2	LVCMOS33	N/A	3.30		PULL-?
val1<2>	Input	D15	BANK2	LVCMOS33	N/A	3.30		PULL-?
val1<1>	Input	D16	BANK2	LVCMOS33	N/A	3.30		PULL-?
val1<0>	Input	E15	BANK2	LVCMOS33	N/A	3.30		PULL-?
val0<3>	Input	E16	BANK2	LVCMOS33	N/A	3.30		PULL-?
val0<2>	Input	F15	BANK2	LVCMOS33	N/A	3.30		PULL-?
val0<1>	Input	G15	BANK2	LVCMOS33	N/A	3.30		PULL-?
val0<0>	Input	G16	BANK2	LVCMOS33	N/A	3.30		PULL-?
rst	Input	L14	BANK3	LVCMOS33	N/A	3.30		
g	Output	?	BANK3	LVCMOS33	N/A	3.30		
f	Output	?	BANK2	LVCMOS33	N/A	3.30		
e	Output	?	BANK3	LVCMOS33	N/A	3.30		
dp	Output	?	BANK3	LVCMOS33	N/A	3.30		
dot3	Input	M14	BANK3	LVCMOS33	N/A	3.30		
dot2	Input	M13	BANK3	LVCMOS33	N/A	3.30		
dot1	Input	H15	BANK2	LVCMOS33	N/A	3.30		PULL-?
dot0	Input	H16	BANK2	LVCMOS33	N/A	3.30		PULL-?
d	Output	?	BANK3	LVCMOS33	N/A	3.30		
clk	Input	T9	BANK4	LVCMOS33	N/A	3.30		
c	Output	?	BANK3	LVCMOS33	N/A	3.30		
b	Output	?	BANK2	LVCMOS33	N/A	3.30		
an3	Output	?	BANK2	LVCMOS33	N/A	3.30		
an2	Output	?	BANK2	LVCMOS33	N/A	3.30		
an1	Output	?	BANK2	LVCMOS33	N/A	3.30		
an0	Output	?	BANK2	LVCMOS33	N/A	3.30		
a	Output	?	BANK2	LVCMOS33	N/A	3.30		

Figure 5: Details, Details…

You need to fill in the remaining details. These are shown as red question marks in Figure 5. Assign pin locations to all of the output pins. This information is available in Figure 3. Finally, you must deal with the ten inputs that are connected to the I/O pins on the expansion port. These pins are not driven by switches or buttons. In fact, they are not driven by anything.

In the termination column, select either a PULLUP or a PULLDOWN for each of the ten pins. This will cause the FPGA itself to weakly pull the pin voltage up, or down, based on the selection. You must select the PULLUP and PULLDOWN options such that the two digits on the right will display the last two digits of your student identification number and the decimal points will be off if you are a graduate student, but on if you are an undergraduate student.

Verify your design in hardware. When you are satisfied with the results, generate a programming file for the PROM and then load your design into the PROM.

Laboratory Hand-In Requirements

Once you have completed a working design, prepare for the submission process. You are required to demonstrate a working design which has been programmed into the PROM. Within six 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 l2_yourlastname.zip. For example, if I were to make a submission, it would be l2_crabill.zip. Then email the archive to the instructor. Only WinZIP archives will be accepted. If your archive is too large, you may remove:

- The xst subdirectory (temporary synthesis files)
- The work subdirectory (temporary simulation files)
- Any file with a .wlf extension (simulation waveform files)

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.