Digital Logic

CSC 244L

Laboratory 3

Combinational Building Blocks and Seven Segment Display

1. Objectives

Describe the difference between active high and active low

Use a multi module project

Implement the seven segment displays in preparation for the first project

2. Pre-Lab

Complete sections 4.1, 4.2.1, and 5.1 and bring them to the start of your lab.

Bring your circuit diagrams to lab with you so that your lab instructor can check them before you start working. Lab time is for building and debugging, not for initial design. You should also bring soft copies of all your SV modules to lab to be compiled and loaded onto the FPGA for testing and demonstration. Show your completed diagrams and SV and have your TA sign off on your lab sheet.

For your SV, you must use separate modules with one .txt file per Verilog module for the various logic functions. Do not .zip your pre-lab files, and do not turn in your SV in '.sv' file format. You must turn these in as a '.txt' file to receive pre-lab credit, '.sv' files are not readable in D2L.

3. System Verilog

In all labs (and projects) you should have exactly one SV module per .sv file. The module should have well documented inputs, outputs, and internal logic connections using comments. Use proper indentation using TAB to make your SV easier to read. When connecting multiple modules together using structural SV, you must use explicit port mapping and not locational port mapping. You will lose points if you do not use explicit port mapping, and the TA (and Mr. Galipeau for projects) will not help you debug until you do.

3.1 Active Level Naming Conventions

Pick your variable names to mean something. If a signal has a logic-1 when a door is open, name it something like `doorOpen`. If a signal has a logic-0 when a window is closed, you might name it `windowClosedL` where the "`L`" indicates that this signal is "active-low," or that it has the value of zero when asserted. Obviously this signal could also be named `windowOpen` which would be active-high, but suppose the signal gave a zero when a fuel tank was full. In that case a good name would be `fuelTankFullL` but `fuelTankEmpty` would not be a good way to avoid the active-low signal because "not all the way full" is different from "empty."It should be noted that there is no "`H`" on the end of active-high signal names. Signal namesare assumed to be active-high unless designated as active-low. Also note that we should not try to avoid active-low signals, just label them appropriately. There is nothing inherently bad about an active-low signal. In fact, proper use of active-high and active-low signals can make the logic simpler or circuit faster.

3.2 Behavioral Verilog

(Excerpt from Ch. 4.1.3: Synthesis) One of the most common mistakes when using HDL is to treat it as a computer program, rather than a language for describing digital hardware. If you do not know what the underlying hardware your HDL would create, you will not get what you expected. You may make a circuit that is too complicated, creates errors, or is unable to be created in hardware. Think of your system in terms of combinational blocks connected by wires. Sketch these blocks on paper and show how they are connected before you start writing your HDL. From now on you can stop using the built-in gates if you prefer and move to the behavioral operators. Until after Project 1, you may only make use of structural SV (connect together modules using wires/logic), the built-in logic gates in Table 3, and the behavioral SV operators in Table 2. There are examples of using these in HDL Example 4.3 in your textbook (pg. 179). You can use these operators on any Boolean variables ("logic" types in SV), and `assign` their values to another Boolean variable (`logic`)

Table 1: Structural System Verilog Built-In Modules

Logic Gate	System Verilog (y is always the output)
NOT	not(y,a);
AND	and(y,a,b,…);
OR	or(y,a,b,…);
NAND	nand(y,a,b,…);
NOR	nor(y,a,b,…);
XOR	xor(y,a,b,…);
XNOR	xnor(y,a,b,…);
BUFFER	buf(y,a);

Table 2: Behavioral System Verilog Built-In Modules

Operator	Meaning
~	NOT
&	AND
~&	NAND
^	XOR
~^	XNOR
|	OR
~|	NOR

3.3 Multiple width logic

It is suggested that starting this week you also use `logic` of multiple bit widths. This is accomplished by adding brackets after `logic` with the size of the wire, as shown below for an 8-bitlogic signal named `DATA`.

```
logic [7:0] DATA;
```

Do not put the brackets after the logic variable name. This creates a different type of logic signal that will compile, but cause you issues in the future. You can then access specific logic wires from `DATA` like an array in C. You can also 'slice' multiple wires off of `DATA`. Two examples are provided below for accessing the 6th bit of DATA, and to select the upper-four bits (nibble).

```
DATA[6]
```

```
DATA[7:4]
```

If you need to combine multiple signals together into an input, you can use the concatenate operator as {`logic1,logic2,logic3`}. An example is provided below for connecting two 4-bitdata signals into a single 8-bit MUX input.

logic [3 : 0] DATA1, DATA2;

mux81 MUXexample (. D({DATA2,DATA1}) , . . .) ;

4. Decoders

A decoder is a CBB that decodes a binary input to one-hot output (exactly one output is active out of 2^N outputs) using an N bit input (representing an N bit binary number). Decoders are used to enable one of 2^N circuits and are used extensively in digital design (e.g., processor instruction decoding). The2:4 decoder in Fig. 3 outputsY_3= 1 ifA_1A_0= 11_2= 3_{10} and $Y_2Y_1Y_0$= 000_2(one-hot output). Y_2 will be active if $A_{1:0}$= 2_{10}, etc. This concept can be extended to any N:2^N decoder.

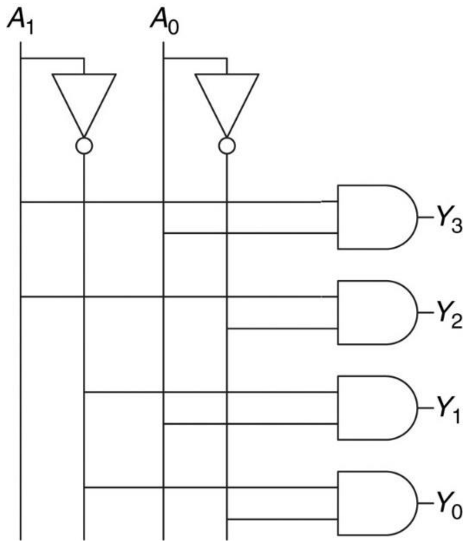


Figure 3: A 2:4 decoder that decodes a 2-bit (A) input to activate exactly one (one-hot) of the 4-bit (Y) outputs. For example, if $A_{1:0}$= 00_2, Y_0 will be active and $Y_{3:1}$ will be inactive

4.1 Pre Lab

Draw the gates representing the first four outputs of a 4:16 decoder.

4.2 System Verilog

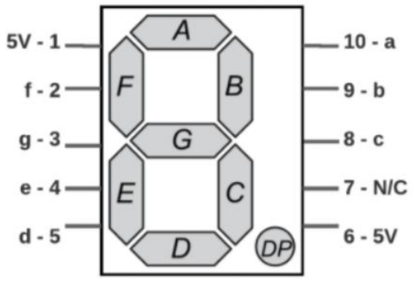
4.2.1 Create a SV file named "dec416.sv" that contains one SV `module` named dec416. Write structural or behavioral SV to describe the operation of a 4:16 decoder with active-high outputs. You may choose to at an enable input (it is your choice whether this enable is active-high or active-low). Remember to follow our active-level naming convention.

4.2.2 Create a SV project named "test_decode". Create a top level SV file that implements the decoder module you wrote for part 4.1.1.

4.2.3 Compile the decoder test module using Quartus Prime Light. Assign switches SW[3:0] to the address inputs (SW[3] as MSB, SW[0] as LSB), switch SW[9] to the enable, and LEDR[8:0]to monitor the output.

4.2.4 Load the decoder test module to your FPGA. Verify the operation of your SV module by checking every possible input combination. Show your working 4:16 decoder and truth table to your TA, and have them sign off on your lab sheet.

5. Seven Segment



A seven segment display is a relatively simple display element that is used for displaying numerical information such as on clocks, meters, and basic calculators. You will be using seven segment displays to for your upcoming projects. Each segment of the display is an LED connected to a pin on the DE-10. while it is possible to display 128 unique symbols with the seven segment display we're only going to implement 16 hex digits.

Digit	Display
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
b	
C	
d	
E	
F	

5.1 Prelab

5.1.1 Fill out the truth table for the seven segment display on page 6.

5.1.2 Draw a block diagram using decoders and any necessary logic gates to drive a seven segment display.

5.2 System Verilog

5.2.1 Create an SV file named "seven_seg.sv" that contains one SV module name seven_seg. Write structural or behavioral SV to describe the operation of seven segment display driver. The description should include at least one "dec416" sub module, and should allow use of the decimal point. The module should have 4 input wires and have outputs so that when it is connected to a seven segment display the hex digits 0-F are displayed.

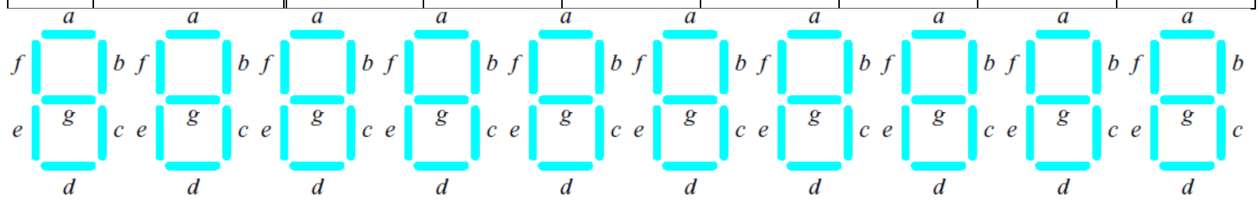
5.2.2 Create a SV project name “test7seg”. Create a top level SV file that implements the “seven_seg” module.

5.2.3 Compile the test module and assign switches SW[4:0] to the address inputs and the right most seven segment display to the outputs

5.2.4 Load the test module to your FPGA. Verify the operation of your SV module by checking every possible input combination. Show your working seven segment decoder to your TA, and have them sign off on your lab sheet.

Pro tip, Implement one segment and verify that it works the way you think it should. Once one segment is working the rest are basically just copy and paste with minor adjustments.

Digit	Binary $D_{3:0}$	S_a	S_b	S_c	S_d	S_e	S_f	S_g
0	0000							
1	0001							
2	0010							
3	0011							
4	0100							
5	0101							
6	0110							
7	0111							
8	1000							
9	1001							
A	1010							
b	1011							
C	1100							
d	1101							
E	1110							
F	1111							



Section	Points	Sign off
Pre Lab		
4.1	5	
4.2.1	5	
5.1.1	5	
5.1.2	5	
Lab		
4.2.4	15	
5.2.4	15	