LABORATORY MANUAL

SC1005 : Digital Logic

[Location: Hardware Laboratory 3, N4-B1a-05]

Experiment 4: Combinational Design with Behavioural Verilog

COMPUTER ENGINEERING COURSE
COMPUTER SCIENCE COURSE

SCHOOL OF COMPUTER SCIENCE & ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

SC1005 Lab 4: Combination Design with Behavioural Verilog

Objective

To introduce the design approach for combinational circuits using Behavioural Verilog.

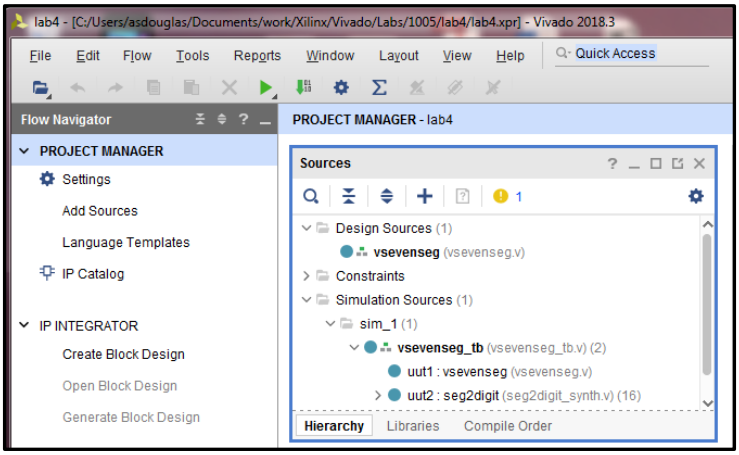
Equipment

As in Lab 3, we will use the Basys 3 FPGA board with an Artix 7 xc7a35tcpg236-1 part. You should also have to hand your Lab 3 manual, the Basys 3 Reference manual and a copy of the lecture notes on Verilog.

Part 1: (Note: Part 1 can be done pre-Lab, using the virtual workstations at vasce.ntu.edu.sg)

First, re-implement the seven segment display decoder from Lab 3 using behavioural Verilog. This will show how much easier a behavioural description is as we do not need to work out any logic equations. Use a **case** statement in a combinational **always** block to implement the decoder, based on the data from Table 1 of Lab 3, except we require active low segments, so these are bit-wise negated to give the required outputs.

1. Start Vivado and from *Quick Start* create a new project (**Quick Start** > **Create Project**). This opens the **New Project Wizard**. In the New Project window, enter a project name (e.g. Lab4). The lab technician will let you know the recommended Project location. Select *RTL Project* and tick the *Do not specify sources at this time* box. Select the **xc7a35tcpg236-1** part corresponding to the Basys 3 FPGA board. You may find it easier to type the front portion of the part number into search window to reduce the number of choices. Click Finish to create the project.
2. Copy all the Verilog files from NTULearn and place them in the same project location specified above (e.g. project_location/Lab4). Check the file extensions (e.g. *filename.v*, *filename.xdc*) are not modified.
3. Create a new Verilog module. Select **File** > **Add Sources** from the menu (or click the + icon in the Sources window). Select **Add or create design sources**, click **Next**, then select **Create File**. Make sure the File type is Verilog, then enter *vsevenseg* as the File name and click **OK**, then **Finish**. Give the module a single input called *x*, select bus, and set the MSB to 3. Add an output called *seg_L*, select bus, and set the MSB to 6. This creates a module with a 4-bit input and a 7-bit output. The input is a 4-bit binary number, and the output represents the seven LED segments. If you double click on *vsevenseg.v* (in the Sources window inside the PROJECT MANAGER window) you should now see a bare module declaration with the inputs and outputs you declared. Note: you may need to expand Design Sources.
4. Then add the simulation source files to the project. Select **File** > **Add Sources** from the menu (or click the + icon in the sources window). Select **Add or create simulation sources**, then click **Next**. Select **Add Files**, and select *vsevenseg_tb.v and seg2digit_synth.v*, then **OK**, then **Finish**. Expand the Design sources and Simulation sources in the Sources window and you should see the following entries.



5. You should now populate the *vsevenseg.v* source module with a single always @ * block, that contains a case statement. The case statement should switch on *x*, and for each case, make a fixed assignment to the 7 bits of *seg_L*. Remember that the order of the bits is segments "gfedcba" and the segments are active low (that is, they are inverted from Table 1 of Lab 3). For example, the 1st entry in the **case** statement should be "4'd0: seg_L = 7'b100_0000;". Remember your *seg_L* output in the **module**

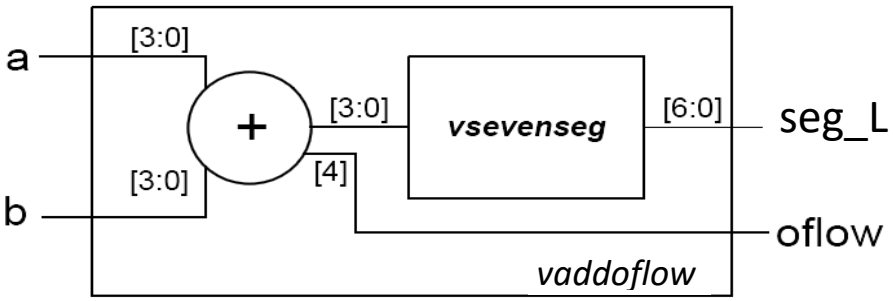
statement will need to be declared as type **reg**. Your case statement should have 16 possible entries for the values 0 to F. Don't forget to add a **default** statement, as "default: seg_L = 7'b111_1111;".

6. Check there are no syntax errors. Syntax errors will show up as a wavy red underline in the source file. Put the cursor over the underlined statement to see the error. Fix all errors.
7. Verify that your design generates the correct segments by running a Behavioural Simulation (from Run Simulation in the Flow Navigator window). The x[3:0] and the digit[3:0] values should be identical. Close the simulation.
8. Run synthesis (from the Flow Navigator window). Correct any errors. Some warnings can be ignored.
9. Open Synthesized Design (from the Flow Navigator window). Click Schematic (in Open Synthesised Design in the Flow Navigator window). Select one of the LUTs. Select the properties tab in the Cell properties window. Note the INIT value. Then select the Truth Table tab and verify that the INIT value matches the truth table (MSB to LSB). Look-up tables (LUTs) are used to implement combinational logic in FPGAs, rather than gates. Close the SYNTHESISED DESIGN window.

Here we have used behavioural modelling to implement this combinational circuit. You can see that even with seven outputs, we only need to write one case statement. The tools took care of all the other steps including determining Boolean logic, simplification, and mapping to the logic available on the FPGA.

Part 2

We will now implement an adder from a high-level description, letting the synthesis tools create the circuit. We will also see how to instantiate modules within our current design.



1. Create a new Verilog source module called *vaddoflow* with two 4-bit inputs, *a* and *b*, a 7-bit output, *seg_L*, and a 1-bit output, *oflow*, as detailed in the figure above.
2. Then open the file and create a simple adder using an **assign** statement. First declare a 5-bit internal wire, *x*, then use an **assign** statement to assign the sum of the two inputs to it. Then, instantiate your *vsevenseg* module from Part 1, using named instantiation. Its output should be connected to the module output. Its input should be connected to the four least significant bits of the adder output. Then using another **assign** statement, connect the *oflow* output to the most significant bit of the adder result. Verify that vaddoflow is the top module (in the Sources window).
3. Add the constraints source file, vaddoflow.xdc. Open the constraints source file and note that the rightmost four switches on the Basys 3 board are connected to the *b* input and the next four switches are connected to the *a* input. The *oflow* output is connected to the rightmost LED, LD0 (pin U16). The *seg_L* output is connected to the 7 segment display (pins W7 to U7).
4. Run Synthesis. Correct any errors and warnings. Then Run Implementation. You should make sure that the project Summary window is visible so that you can see the status of the various processes.
5. Then Generate Bitstream.
6. Make sure the Basys 3 board is connected to the computer using a USB cable, and that the red power LED is lit. Open the hardware Manager. Open Target and select Auto Connect. Verify the Digilent device is shown in the Hardware window. Then Program Device and select xc7a35t_0. Click Program.
7. Test the circuit. You should see the *oflow* LED light whenever the result is more than F. Note all four 7-segment displays are active.
8. **Assessment (Please get the Lab Tutor to assess part 2)**

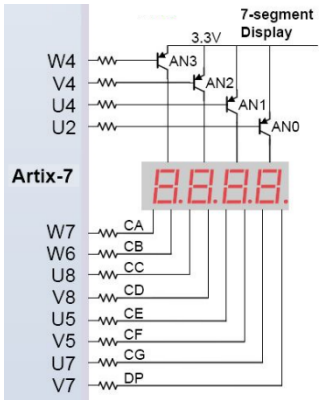
In Part 2 we were able to make use of vector selection to use different bits of a vector for different uses. The adder was also synthesised entirely from a single Verilog arithmetic expression.

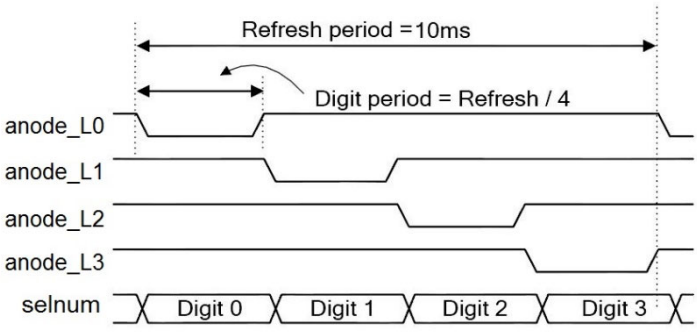
Part 3

In lab 3, you built a circuit that allows you to select which digit of the seven segment display to use. Here, we will create the circuit that enables us to use both at the same time. This is done by switching the segments at a high enough rate for them all to appear lit at the same time. The four digits on the seven segment module cannot be controlled at the same time. Instead, we send four (or fewer) numbers to display separately at high speed in a repeating manner, and control each segment's anode connection. So long as this happens at above 40Hz or 50Hz, the digits won't blink. See the figure below for more detail.

1. Remove all the Design and Simulation source files from the project by right clicking each and then selecting Remove File from Project. Alternatively, you could open a completely new project.
2. Add the *segtoggle.v* file as a design source and *segtoggle.xdc* as a constrains file.
3. The circuit in *segtoggle.v* generates the product of inputs *a* and *b*, and displays the 8-bit result on the 2 rightmost 7-segment displays. Open *segtoggle.v*.
   a. At the bottom of *segtoggle.v* (**below** the "Do NOT modify the code above" comment) you will see declarations for the ten's and one's digit of the bench number. **Enter your bench number THERE.**
   b. Generate an 8-bit product (*res*) from *a* and *b*. (e.g. delete the "//" from the **assign** statement).
   c. Below this is another comment, where you should copy your seven segment decoder always block from Part 1. You only need to copy the always block. Modify the signal that controls the case statement to be *numsel*, as indicated.
4. Open the *segtoggle.xdc* constraints file. Examine the constraints file and note the following:
   a. Lines 2 to 4 enable the 100MHz system clock and connect it to *clk*.
   b. Lines 8 to 15 and 19 to 26 connect a and b to the Basys 3 slider switches.
   c. Lines 70-71 connect the reset (rst) to the center pushbutton.
   d. Lines 73-74 connect sel to the upper pushbutton. Sel is used to display the bench number
   e. Lines 84 to 97 connect the active low segments (*seg_L*[6:0]) to the 7-segment display.
   f. Lines 104 to 111 connect the 4 active low anodes (*anode_L*[3:0]) to the 7-segment display.
5. Run Synthesis. Correct any errors and critical warnings. Then Run Implementation. Again, you should make the project Summary window active so that you can see the status of the various processes.
6. Generate the bitstream and program the device. Test your circuit. It should now display two digits, each controlled by the *a* (W13-W15) and *b* (W17-V17) switches on the board. What happens when you press the *rst* button (BTNC).
7. **Assessment (Please get the Lab Tutor to assess part 3)**

What does the segtoggle module do: Firstly it implements a 20-bit counter and uses the most significant two bits (seg7_clk) to generate the 7-segment anode drivers. The board clock runs at 100MHz, so the 20th bit of the counter runs at 95Hz, resulting in a refresh period of 10.5ms. The top ***always @ ** block then generates the anode driver signals (U2-W4) and the digit to be displayed. Note* ***anode_L[3:2] =1111****, meaning these 2 displays are* ***OFF****. So the result of a multiplied by b (= x) is displayed on the two rightmost digits. You do not need to worry about all of the details inside the module yet, as this includes synchronous design techniques, that you have not yet covered.*





Note: The following system warnings can be safely ignored

Synthesis (Part 1):

WARNING: [Constraints 18-5210] No constraints selected for write.

Synthesis (Part 2):

WARNING: [Constraints 18-5210] No constraints selected for write.

Implementation (Part 2):

Place Design: [Place 46-29] place_design is not in timing mode. Skip physical synthesis in placer.
Route Design: [Power 33-232] No user defined clocks were found in the design.
Route Design: [Timing 38-313] There are no user specified timing constraints.

Bitstream (Part 2):

There should be NO warnings.

Synthesis (Part 3):

WARNING: [Synth 8-3917] design segtoggle has port anode_L[3] driven by constant 1 (See note below).
WARNING: [Synth 8-3917] design segtoggle has port anode_L[2] driven by constant 1 (See note below).
WARNING: [Constraints 18-5210] No constraints selected for write.

Implementation (Part 3):

[Place 46-29] place_design is not in timing mode. Skip physical synthesis in placer

Bitstream (Part 3):

There should be NO warnings

Notes:

The two MSBs of anode_L (in seven_seg.v) are not used. They are fixed High, so you get a warning.

The other warnings (above) are system warnings and can be ignored.

ANY other errors or warnings probably mean that you have made a coding/syntax mistake, and you will need to fix these.