Lab 3

Combinational-Circuit Building Blocks

The purpose of this lab is to build useful digital circuits by combining different combinational-circuit building blocks, such as multiplexers, decoders, priority encoders, and seven-segment code converters. In all parts, you will be displaying a numerical output using one of the available seven-segment digits.

Part 1 (Displaying the output of a priority encoder)

In this part, you will explore the functionality of a priority encoder by connecting its inputs to switches and displaying the hex number of the highest asserted switch. The number will be displayed on a single seven-segment digit.

The following steps should guide you through the implementation:

1. Import the `priority_encoder_generic.v` from the starter code. The module describes a generic priority encoder where the number of inputs is parameterized.
2. Import the `hex2sseg.v` from the starer code. The module describes a hexadecimal to seven-segment coder converter.
3. Design a test system for the priority encoder. The system should use a priority encoder of size 16x4 and a hexadecimal to seven-segment code converter to display its output on one of the seven-segment digits.
4. Draw a schematic of the test system and include it with your submission.
5. Write a verilog module (call it `prior_encoder_test.v`) that describes the test system you designed.
6. Verify the functionality of the test system by implementing it on the FPGA board using the following IO specifications:
    a. SW15 ← SW0 connected to the 16 inputs of the priority encoder. Assume SW15 has the highest priority.
    b. Activate a single seven-segment digit by setting its `AN` to 0
    c. Connect the output of the hex2sseg code converter to the seven-segment display LEDs
    d. Turn off the decimal point (`DP`)

Part 2 (Building a rudimentary seven-segment display driver)

The objective in this part is to explore an application of binary decoders and introduce you to a very rudimentary seven-segment display driver. You will also build a test circuit that takes a 3-bit input `X` and display the value of `X` on a seven-segment digit. The location of the activated digit will be the same as the value of `X`. In other words, if `X` is set to `3'b101`, then the seven-segment digit located in the 5th place will be activated and it will display the number 5.

Assuming the very first seven-segment digit location is 0, then changing `X` from 0 to 8 should activate the digits in order and display the appropriate number on each digit.

The following steps should guide you through the implementation:

1. Import the `decoder_generic.v` from the starter code. The module describes a generic decoder where the number of inputs is parameterized.
2. Import the `hex2sseg.v` from the starer code. The module describes a hexadecimal to seven-segment coder converter. (If you are using the same project as in Part 1, you should have this module imported already)
3. Use the 2 imported modules to design a rudementary seven-segment driver (call it `first_sseg_driver`). The driver should have the following specfications:
    a. 3-bit input `active_digit` to select one of eight seven-segment digits to activate
    b. 4-bit input `num` containing a binary number to be displayed on the activated digit
    c. 1-bit input `DP_ctrl` to turn on/off of the decimal point
    d. Outputs to set the appropriate seven-segment control signals (i.e. `sseg`, `AN`, `DP`)
4. Draw a schematic of the driver and include it with your submission.
5. Design a test system that that utilizes the driver and implements the functionality described above. In other words, connect a 3-bit input `X` to both the `active_digit` and `num`. (Call this system `first_sseg_driver_test`). The system should output appropriate seven-segment control signals (i.e. `sseg`, `AN`, `DP`).
6. Verify the functionality of the test system by implementing it on the FPGA board using the following IO specifications:
    a. SW3 ← SW0 connected to the input `X`.
    b. Connect the seven-segment control signals to appropriate output from the system.
7. In your video demo, change `X` from 0 to 8.

Question: If you were able to change the input `X` very quickly, say in the order of 30 times per second or more, what would happen?

Part 3 (Using the rudimentary seven-segment display driver at the output of a simple calculator)

The objective of this part is to use the rudimentary seven-segment display driver to show the result obtained from the modified simple calculator you built in lab2. The digits can only be displayed one at a time using some selector switches. You can assume all numbers are unsigned for this part.

Recall, the calculator should perform 4-bit addition, subtraction, multiplication, and should utilize a binary-to-BCD converter to generate a 3 digit BCD (or 12-bit) result.

1. Write a verilog module (call it `simple_calc_first_sseg`) that will contain the whole system for this part.
2. Import `simple_calc` along with all necessary dependencies (i.e. the adder/subtractor, csa multiplier,etc.)
   As a reminder
   a. The module should accept two 4-bit inputs `X, Y`
   b. The module should accept a 2-bit operator select input (`op_sel`).
      i. `op_sel` = 00 (add)
      ii. `op_sel` = 01 (subtract)
      iii. `op_sel` = 1x (multiply)
3. Import `bin2bcd` (from lab2) to convert the 8-bit binary output of the calculator to its equivalent 12-bit BCD.
4. Import `mux_4x1_nbit` from the starter code. The module describes a generic 4x1 multiplexer where the number of bits for each input is parameterized.
5. Separate the 12-bit BCD result into 3 BCD digits (4-bit each) and connect each of the digits to one of the inputs of the 4x1 MUX.
6. Use the `first_sseg_driver` module to display the 3-digit BCD result on the seven-segment digits (4, 5, 6)
7. Draw a schematic of the whole system and include it with your submission. You do not need to show the content of the seven-segment driver or the simple calculator, just include them as boxes with inputs and outputs.
8. Verify the functionality of your simple calculator by implementing in on the FPGA board using the following IO specifications:
   i. SW3 ← SW0 for the input `X`
   ii. SW7 ← SW4 for the input `Y`
   iii. SW9 ← SW8 to control which seven-segment digit is activated (these switches will select digits 4, 5, 6, 7)
   iv. SW15 ← SW14 for the `op_sel`
   v. Seven-Segment7 ← Seven-Segment4 for the output BCD `result`
   vi. LED14 to display the `carry_out` of the adder/subtractor
   vii. LED15 to display the `overflow` of the adder/subtractor

Submission check list:

[ ] All Verilog code you generated or modified
[ ] All testbenches written
[ ] Embed all screenshot of your testbench output in your README.md
[ ] Embed all block diagram generated in your README.md
[ ] 3 Short videos demonstrating each of the parts