CSE 100/L Logic Design
Fall 2025

Lab Assignment 2

Code Submission deadlines:

- 100% Before the end of your section Thursday, October 9th.
- 95% after the end of your section until 11:59 PM Thursday, October 9th.
- 80% after 11:59PM Thursday, October 9th.
- 60% after 11:59PM Sunday, October 12th.
- 40% after 11:59PM Monday, October 13th.
- 20% after 11:59PM Tuesday, October 14th.
- 0% after 11:59PM Wednesday, October 15th.

Demo deadline:

- Code may be demonstrated until the end of the next lab assignment. You must submit before you demo, and the code you submit must be the code you demo. You may be required to download the code from Canvas to demo.

Write-Up deadline:

- There is no write-up submission for this lab assignment.

Academic Integrity Policy:

The Academic Integrity Policy is stated in the syllabus on Canvas. Students are responsible for reading the syllabus. Not reading the syllabus is not an excuse for academic misconduct.

- Work submitted in CSE 100 must be yours alone. Sharing/sending code, submitting code written by others, and submitting (any) code from generative AI **are all violations of academic integrity**.
- Working together on a whiteboard, on paper, debugging waveforms, explaining concepts and/or examples, is encouraged, **as long as no solution code is shared**.

Verilog Operator Constraints:

In this lab you can use any of the following combinational operators:

- All combinational logic must be implemented using only `assign` statements.
- Bit-wise Operators: &, |, $\sim$, $\wedge$, and $\sim \wedge$
- Concatenation and Replication: {} and {{}}

All other operators will be considered behavioral and will result in a **score of 0**!

Overview

Be sure to complete the prelab before beginning this assignment.
In this lab you will implement:

- an 8-bit adder,
- a 2-to-1 multiplexer for two 8-bit buses,
- a hex to 7-segment display encoder (see below),
- and an 8-bit adder/subtractor.

These components will be used to implement a sign changer. The input will be provided by the switches and pushbutton btnU. The output of the sign changer will be displayed on the right two digits of the 7 segment display.

xs

The sign changer (symbol below) is a combinational circuit that converts a positive 8-bit number to its negative representation and vice versa.

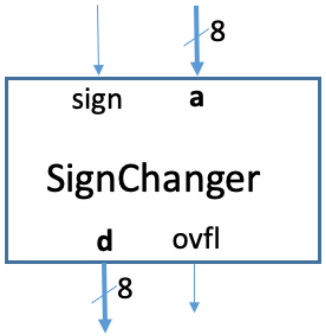

Figure 1: Sign Changer symbol

When `sign` is 0, the output `d` is `a`. When `sign` is 1, the output `d` is `-a`. The 8-bit numbers, `a[7:0]`, will be provided by the switches `sw[7:0]`. Pushbutton `btnU` will provide the input `sign` that determines whether the input `a` has its sign changed.

The result will be displayed on the two rightmost digits of the Basys3 seven segment display. The table below contains several examples:

sign	`a`	`a` (in binary)	`d`	`d` (in hex)	`ovfl`
0	8'd46	8'b00101110	8'd46	8'h2E	0
1	8'd46	8'b00101110	-8'd46	8'hD2	0
0	8'd17	8'b00010001	8'd17	8'h11	0
1	-8'd119	8'b10001001	8'd119	8'h77	0
0	-8'd128	8'b10000000	-8'd128	8'h80	0
1	-8'd128	8'b10000000	8'd128	8'h80	1

Table 1: Sign Changer circuit example inputs/outputs.

Since 8 bits may not be enough for the result, you should illuminate the `dp` segment when there is an overflow. This occurs when provided value has its sign changed to a value outside of the valid range

of 8 bits. 8 bits represents values from -128 to 127. Note that this needs to be detected by Add8 (it is not related to the final carry output).

The figure below contains a (partial) block diagram of the top level of your project. Some of the details have been left for you to complete.

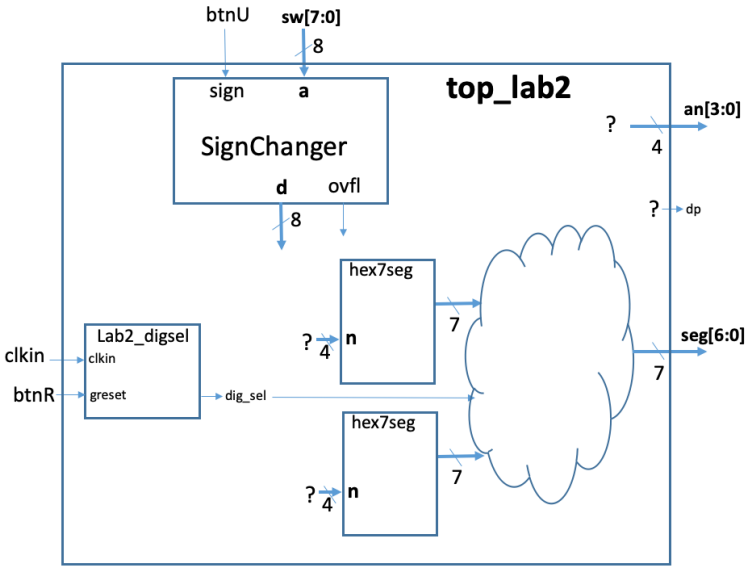

Figure 2: Lab 2 design overview

Multiplexer

Your first task is to implement a 2-to-1, 8-bit multiplexer (Tip: Call this file mux8bit.v). This can be a single assign statement using bit-vector operators.

Adder

Build an 8-bit adder by first making a full adder module FA (Tip: Call this file fa.v). Your full adder FA will have three input bits: `A_i`, `B_i`, `cin_i`. It will have three outputs, `S_o`, `cout_o`, and `ovfl_o`.

Assemble 8 of your FA modules to make your 8-bit adder:

```
module Add8(
    input [7:0] A_i,
    input [7:0] B_i,
    input cin_i,
    output [7:0] S_o,
    output ovfl_o,
    output cout_o
);
```

Here the `ovfl_o` output should be 1 when adding two positive numbers produces a negative number (overflow!), or adding two negative numbers produces a positive number (overflow!).

Adder/Subtractor

Using your adder and multiplexer, build an AddSub8 module (Tip: Call this file AddSub8.v). The inputs to AddSub8 are the two 8-bit two's complement numbers, `A_i[7:0]` and `B_i[7:0]`, and the input `sub_i`. When `sub_i`== 0, the output, `S_o[7:0]` should be the sum of the two numbers, `A_i+B_i`, in two's complement. When `sub_i`==1, the output should be `A_i-B_i`, in two's complement.

```
module AddSub8(
    input [7:0] A_i,
    input [7:0] B_i,
    input sub_i,
    output [7:0] S_o,
    output ovfl_o
);
```

You will also need to generate logic for the `ovfl_o` output.

Sign Changer

Using all of the previous modules, build a sign changer module (Tip: Call this file SignChanger.v). Below is the ports list of the SignChanger module:

```
module SignChanger(
    input [7:0] A_i,
    input sign_i,
    output [7:0] D_i,
    output ovfl_o
);
```

The inputs into the SignChanger are an 8-bit two's complement number `A_i[7:0]` and 1-bit value `sign_i`. When `sign_i` is 0, the value of the output `d_o[7:0]` should just be `a_i[7:0]`. When `sign_i` is 1, the value of `d_o[7:0]` should be `-a_i[7:0]` if it positive else `a_i[7:0]` if it is negative. **Reminder, you may not use the - operator.**
The value of the output `ovfl_o` needs to be generated. This has likely been generated by a previously created module.

7-Segment Converter

You will create a module hex7seg which takes a 4-bit bus `N_i[3:0]` and produces the signals to control the 7 segment display (`Seg_o[6:0]`) (Tip: Call this module hex7seg.v). This is simlar to Lab 1, however this time your module will use buses for the input and output, and display all 16 hex digits.

(You should have the equations in your notebook from completing the pre-lab.)

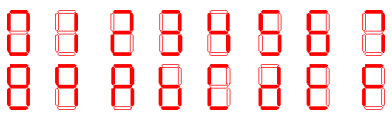

Figure 3: 7-segment display hex digit

Procedure

1. Create a new project for Lab 2

- NOTE: don't forget to add a fresh copy of the constraints file, Basys3_Master.xdc, which can be found on Canvas.

2. Create a top level module with:

   - Inputs `sw[7:0]`, `btnU`, and `clkin` and
   - Outputs `seg[6:0]`, `dp`, `an[3:0]`, and `led[7:0]`

3. In your top level module, you should have:

   - One instance of your **SignChanger** and
   - Two instances of **hex7seg** and
   - One instance of your **mux**

4. Connect the switches to the leds.

   - NOTE: this is to allow you to detect problems with the switches.

5. Connect the **SignChanger** outputs:

   - The lower 4 bits of the output will connect to one instance of **hex7seg** and
   - The upper 4 bits to the output will connect to the other instance of **hex7seg**

6. Add a signal (wire) called **dig_sel**. Using the signal **dig_sel**, and your multiplexer, produce the bit vector **seg[6:0]** by *combining* the outputs of the two **hex7seg** modules.

   - When **dig_sel** is high, **seg[6:0]** is the output of one of the two **hex7seg** and
   - When **dig_sel** is low, **seg[6:0]** is the output of the other **hex7seg**.

7. Using **dig_sel** arrange for the appropriate 7-segment display to be selected (either **an[0]** or **an[1]**) so that the value of your **SignChanger** result is displayed on the two rightmost 7-segment digits in hex.

8. Provide appropriate values for the remaining 7-segment controls (**dp**, **an[2]**, **an[3]**).

9. Download **lab2_digsel.v** from Canvas.

10. In the Vivado Project Manager use **Add Sources** to add this file to your project. This Verilog source defines the following module:

```
module lab2_digsel(
    input clkin,   // Basys3 clock
    input greset,  // btnR
    output digsel  // slow signal for 7-segment display controls
);
```

    - The output pin, **digsel** alternates between high and low at a low frequency.
    - Add an instance of **lab2_digsel** to your top level and connect the ports of **lab2_digsel** as follows:

      `.clkin(clkin), .greset(btnR), .digsel(dig_sel)`

11. Now edit your constraints file. Since we used the same names as in the constraints file, you should only need to uncomment the lines corresponding to

    - **sw[7:0]**,
    - **btnU**,
    - **seg[6:0]**,
    - **dp**,
    - **an[3:0]**, and
    - **led[7:0]**.
    - For **clkin**, uncomment the 3 lines below "## Clock" signal near the top of the constraint file and replace **clk** with **clkin** so that these lines are as below:

      ```
      set_property PACKAGE_PIN W5 [get_ports clkin]
      ...
      set_property IOSTANDARD LVCMOS33 [get_ports clkin]
      ...
      create_clock -period 10.000 -name sys_clk_pin \
        -waveform {0.000 5.000} -add [get_ports clkin]
      ```

12. Simulate your design:

    (a) Your TA will provide a short lecture on simulation and a video recording of the lecture will be made available. You will be provided instructions for simulating a design in Vivado and tracking down errors. You can also skim this section of the Vivado simulation user guide, covering the basic commands you will need to control the simulator.

    (b) Download test_signchanger.v from Canvas. If Windows or your browser changes its extension from .v to .txt then you will need to change it back.

    (c) Add this file as a source to your Vivado project. (Make sure you selected the option to copy it into your project.)

    (d) You should now see the file in your Sources panel. Right-click on it and select **Move to Simulation Sources** from the pop-up menu. The file will disappear, but if you expand **sim1** under **Simulation Sources** you'll find it again. This file is intended only for simulation; it should not be implemented.

    (e) Open **test_signchanger.v** in Vivado and examine it. This is a Verilog file in which one module, **test_signchanger**, is defined. This module has no inputs or outputs. Its sole reason for existence is to provide input values (more formally called test vectors) to the top module in your Lab 2 project and allow you to observe the outputs. In **test_signchanger**

    - the inputs/outputs connected to the top level module of the design are defined,
    - the top level module is instantiated, and
    - the values for the inputs are specified by assigning values to the inputs and advancing time (#500;) in increments of 500 nanoseconds, begining at time 0 in

      ```
      initial
      begin
      ...
      end
      ```

    (f) Follow the instructions in the comments to add the test vectors you came up with in the pre-lab to the test bench. You may need to adjust the name of the top level module to match your design.

(g) Click on **Run Simulation** and select **Behavioral Simulation** (the first option in the menu). If there are no errors, the simulator will start up and you will see the same display as in the the guide your skimmed in Step 10.

(h) Your input waveforms must include values that produce every possible output (all 16 hex digits, 0 through F). Currently the simulation includes two values for the switches. Add additional lines to the testbench to generate stimuli to produce more values. Advance time by 500 ns between each value (using #500;).

(i) Simulate your testbench. You must also display the 8-bit bus output of your SignChanger (see the example snapshot below) **without making them pins of your top level module!!!**.

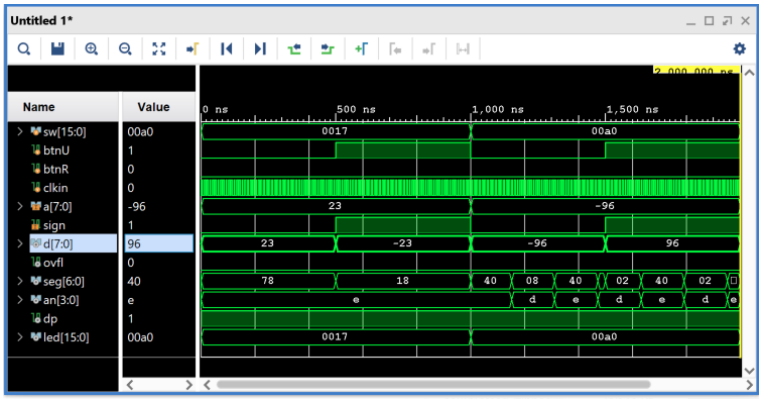

Figure 4: Simulation snapshot

1. ***Implement your design, configure the FPGA and demonstrate your design to one of the TAs. You must show the simulation wave window with the output bus of the SignChanger displayed.***

2. ***Once your working design has been demonstrated, upload your archived project file (zip file) to the "Lab Assignment 2: Demo and Code Submission" assignment in Canvas. (You can continue to improve your project if desired, but you should submit the version you just demonstrated.)***

3. Remember to archive your project even if you are not done. Files left on the PC are not protected. Should it become necessary to re-image that PC, its disks will be wiped clean. The file you submitted in Canvas can serve as a backup, but if you have not submitted please backup the zip file somewhere safe.

4. **Important** Please remember to turn off the power to the Basys 3 board when you are done.

Rubric

There is no partial credit for this lab. All students must completely satisfy the lab description, as many components are used in later labs.