CS 223 Digital Design Laboratory Assignment 2

Digital Building Blocks: Decoders and Multiplexers

Impotant Note: Preliminary Report Due: November 2, 2025. At 23:59
Lab Report Due: Immediately after the Lab session.

Location: EA-Z04 (in the EA building, straight ahead past the elevators)
Groups: Each student will do the lab individually. Group size = 1
Scoring: The number inside brackets [ ] for each section denotes the point obtained from that section (i.e. **[15]** means that part has a maximum of 15 scores).

Preliminary Work [35]
IMPORTANT NOTES:

- Include your code as plain text, not image!
- You should not upload reports with handwriting. Use computers to write your reports, for circuit schematics, you can use online tools such as Draw.io (`https://app.diagrams.net/`) or Vivado schematics.
- Name your reports in the format: `Name_Surname_ID.pdf`
- Your report should have a cover page including: course code, course name and section, the number of the lab, your name-surname, student ID, date.
- Upload your report in the PDF format.
- **MENTIONED CRITERIA ABOVE ARE MANDATORY, IF ANY OF THEM ARE NOT FOLLOWED, THE PRELIMINARY WORK WILL NOT BE ACCEPTED.**

This section must be finished **before** you arrive at the lab. Set up a clearly structured report of your work and upload it to Moodle prior to the lab's start time. Be sure to include your code as **plain text** rather than images!

Today's exercise requires substantial advance preparation. Prepare your preliminary designs and SystemVerilog modules ahead of time, and assemble them into a tidy Preliminary Report that includes a printed cover page as well as printed pages for your SystemVerilog source. Give each part an appropriate heading. The cover page must list: the course name and code, the lab number, your full name and student ID, and the date.

Multiplexer

[3] First, write the Boolean equation of a 3-to-1 multiplexer (clearly define data inputs and select lines). Then, rewrite the *same* Boolean equation *without using the AND operation* (e.g., express it using only OR and NOT via De Morgan's laws). Finally, provide the graphical symbol, a logic diagram (constructed with NOR, OR, and NOT gates), and the truth table for the 3-to-1 multiplexer.

[2] Write a behavioral SystemVerilog module for a 3-to-1 multiplexer and supply a corresponding testbench.

[1] Provide the graphical symbol and truth table for a 4-to-1 multiplexer. In the symbol, indicate which data input maps to each select-bit combination. For example, label the first input pin with "00".

[2] Write a behavioral SystemVerilog module for a 4-to-1 multiplexer built from three 2-to-1 multiplexers, and include a testbench for it.

• Consider the truth table of the three-input function given in Table 1a.

Table 1: Three-input logic functions.

(a) Three-input logic function f.

w_1	w_2	w_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

(b) Three-input logic function g.

w_1	w_2	w_3	g
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

[1] Which operation is realized by the function f given in Table 1a?

[2] Implement the function f given in Table 1a using a single 4-to-1 multiplexer and a minimal number of inverters. Describe the procedure with a few short sentences. Draw its logic diagram.

[2] Implement the function f given in Table 1a using a minimal number of 2-to-1 multiplexers and inverters. Describe the procedure with a few short sentences. Draw its logic diagram.

• Consider the truth table of the three-input function given in Table 1b. Unless otherwise stated, use the 4-to-1 MUX select-line convention $s_1 = w_2$ and $s_0 = w_1$.

[2] Implement the function g given in Table 1b using a single 4-to-1 multiplexer. Assume the select signals are $s_1 = w_2$ and $s_0 = w_1$. Describe the procedure with a few short sentences. Draw the logic diagram and the truth table of the MUX in terms of the select signals (s_1, s_0), clearly indicating which data inputs $D_0, \ldots, D_3$ correspond to each select combination.

[2] Implement the function g given in Table 1b using a single 2-to-1 multiplexer and a minimal number of gates. Clearly state which input bit is used as the MUX select signal (e.g., $s = w_k$). Describe the procedure with a few short sentences. Draw the logic diagram and the truth table of the MUX in terms of the select signal s.

[2] Write a behavioral SystemVerilog module for the function g given in Table 1b that you implemented with a 2-to-1 multiplexer in the previous part, and provide a corresponding testbench for simulation.

Decoder

[3] Graphical symbol, logic diagram, and truth table of a 2-to-4 decoder. Include the "Enable" signal.

[2] Behavioral SystemVerilog module for 2-to-4 decoder and a testbench for it.

[2] Logic diagram of 3-to-8 decoder using 2-to-4 decoders and minimal amount of gates. You can draw 2-to-4 decoders as a block. Include the "Enable" signal.

[2] Structural SystemVerilog module for 3-to-8 decoder using 2-to-4 decoders you implemented in the previous part and a testbench for it.

Logic Function

[2] Schematic (block diagram) and SystemVerilog module of 8-to-1 MUX by using two 4-to-1 MUX modules, two AND gates, an INVERTER, and an OR gate. Prepare a test bench for it.

[5] Schematic (block diagram) and SystemVerilog module for $F(A, B, C, D, E) = \sum(0, 1, 2, 3, 4, 10, 11, 16, 17, 18, 19, 25, 28, 29)$ function (note that there are 14 terms), using one (not two) 8-to-1 multiplexer and one 2-to-4 decoder. Do not use any other gates (AND, OR, XOR, etc.). Your design must contain only one 8-to-1 multiplexer and one 2-to-4 decoder. Complements of signals are available.

Important notes on signal assignments on FPGA:

- In this lab, you are going to name multi-bit input and output signals. If you choose to name a bits individually in your SystemVerilog code, take care to index the most significant bit with the most significant index number. For example, if you have a four bit signal Y, represent it as $y_3y_2y_1y_0$ where y_3 is the most significant bit.
- Be consistent with the ordering, otherwise you may confuse yourself into thinking that your circuit is faulty where in reality you only misconnected your correctly functioning modules.
- Follow the same convention while assigning your inputs to switches and outputs to LEDs. While experimenting, you are more likely to place the board where the switches are closer to you. In this orientation, try to physically assign the most significant bits of a signal on the left hand side of switches and LEDs. For example, if you want to input decimal "10" in binary format with switches, you should input "1010" to switches where the leftmost switch is 1. If you hold the board in the other orientation, simply flip the physical pin assignment order. Just be consistent with the way you assign your pins.
- Representing and assigning signals with this convention is more intuitive and following it will help both you while debugging and TAs while grading your circuits.

Part 1: Multiplexer [20]

A multiplexer ("MUX") is a common higher-level module in digital design. An M-to-1 MUX accepts M data inputs and produces one data output, forwarding exactly one input to the output. A set of select lines chooses which input is passed through. Larger multiplexers can be built hierarchically by composing smaller ones, which you will practice in this part.

See Table 1b for the truth table of function g. In the preliminary work, you realized g using a single 2-to-1 MUX together with a few logic gates.

[5] Simulation: Simulate your implementation of g built with a 2-to-1 MUX plus gates, and present the results to your TA.

[5] Test: Using the Xilinx design flow, synthesize, generate the programming file, and download the bitstream of your g design to the BASYS-3 board. With your assigned switches and LEDs on BASYS-3, verify operation on hardware. Once you are sure with the results, demonstrate the running circuit to the TA and be ready to answer questions.

Design: Write structural SystemVerilog for an 8-to-1 MUX constructed from two 4-to-1 MUX blocks, two AND gates, one OR gate, and one inverter.

[5] Simulation: Simulate your 8-to-1 MUX and show the results to your TA.

[5] Test: Again follow the Xilinx flow to synthesize, create the programming file, and program your 8-to-1 MUX onto BASYS-3. Using the previously assigned switches and LEDs, validate the hardware behavior. When you are satisfied it works as intended, demonstrate the physical implementation to the TA and be prepared for potential questions.

Part 2: Decoder [20]

Decoders are fundamental components in digital design. While they can be implemented directly with logic gates, their behavior is typically specified (and modeled in SystemVerilog) rather than described structurally.

A 2-to-4 decoder decodes a 2-bit input binary number by setting exactly one of the decoder's 4 outputs to 1. Unless it has an enable signal, one and only one output of a decoder will ever be 1 at the same time, corresponding to the current value of the inputs. With an enable signal, it is possible to make all the outputs be 0, when the decoder is disabled. When enabled, it behaves as described above. Decoder outputs are mutually exclusive, and in fact are the minterms of the inputs.

Design: Provide System Verilog code that models a 2-to-4 decoder in behavioral style (i.e., Boolean equations with continuous assignments).

[10] Simulation: Using System Verilog testbench code, verify in simulation that your enabled 2-to-4 decoder operates correctly. (Ensure the port order in the module and its instantiation match one-to-one.)

Implementation: Next, follow the Xilinx design flow to synthesize, generate the programming file, and download the 2-to-4 decoder bitstream to your BASYS-3 FPGA board.

[10] Test: Using your assigned BASYS-3 switches and LEDs, test the decoder. Once satisfied it works correctly, demonstrate the hardware results to the TA and be prepared to answer questions the TA might ask.

Part 3: Logic Function [15]

Set up the circuit you designed in the preliminary work for

$$F(A, B, C, D, E) = \sum(0, 1, 2, 3, 4, 10, 11, 16, 17, 18, 19, 25, 28, 29)$$

in a new module, using a single 8-to-1 multiplexer and a 2-to-4 decoder. Using inverses of signals is allowed.

[5] Simulation: Simulate your implementation and show it to your TA.

[10] Test: Implement your circuit on FPGA using switches as inputs and a LED as output. Show your circuit to TA. Be prepared to answer questions that you may be asked.

Part 4: Bit Shifter [10]

In many digital designs, it is necessary to use circuits that shift the bits of a vector left or right by one or more positions. Design a circuit that, when commanded, shifts a four-bit vector $W = w_3w_2w_1w_0$ exactly one position to the right whenever the control input *Shift* equals 1. Let the outputs be a four-bit vector $Y = y_3y_2y_1y_0$ together with a signal k, defined so that if *Shift* = 1 then $y_3 = 0$, $y_2 = w_3$, $y_1 = w_2$, $y_0 = w_1$, and $k = w_0$. If *Shift* = 0, the circuit must pass the input through unchanged, i.e., $Y = W$, and set $k = 0$.

Design and implement the bit-shifter circuit exactly as specified above.

[5] Simulation: Simulate the bit-shifter circuit and present the results to your TA.

[5] Test: Follow the Xilinx design flow to synthesize, generate the programming file, and download the bitstream of your bit-shifter to the BASYS-3 FPGA board. Use four switches to provide the input vector and a separate switch for the *Shift* control. Drive four LEDs with the

output vector and an additional LED with k. After confirming correct behavior, demonstrate the hardware implementation to the TA and be ready to answer questions.

Hint: Consider which building block is appropriate for shifting the bits of a vector.

Clean Up

1. Clean up your lab station, and return all the parts, wires, the Beti trainer board, etc. Leave your lab workstation for others the way you would like to find it.
2. CONGRATULATIONS! You are finished with the lab.

Notes

- Advance work on this lab, and all labs, is strongly suggested.
- Be sure to read and follow the Policies and other related material for CS223 labs, posted in Moodle.

Lab Policies

1. DO NOT CHAT OR SIT AROUND WASTING TIME. OTHERWISE, YOU WILL BE WASTING THE TIME OF YOURSELF, YOUR FRIENDS, AND YOUR TA's.
2. There are three computers in each row in the lab. Don't use middle computers, unless you are allowed by lab coordinator.
3. You borrow a lab-board containing the development board, connectors, etc. in the beginning. The lab coordinator takes your signature. When you are done, return it to his/her, otherwise you will be responsible and lose points.
4. Each lab-board has a number. You must always use the same board throughout the semester.
5. You must be in the lab, working on the lab, from the time lab starts until you finish and leave. (bathroom and snack breaks are the exception to this rule). Absence from the lab, at any time, is counted as absence from the whole lab that day.
6. No cell phone usage during lab. Tell friends not to call during the lab hours–you are busy learning how digital circuits work!
7. Internet usage is permitted only to lab-related technical sites.
8. If you come to lab later than 30 minutes, you will lose that session completely.
9. When you are done, DO NOT return IC parts into the IC boxes where you've taken them first. Just put them inside your Lab-board box. Lab coordinator will check and return them later.