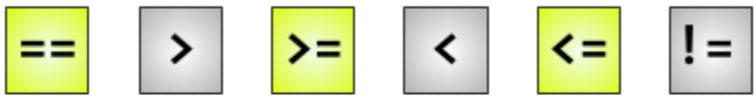
Lab 3: Comparators using SystemVerilog

ECE 211: Digital Circuits I — Spring 2025

Abstract. In the lab, we will implement modules for our two comparison operations that we designed last week—greater than (>) and equal to (==)—in SystemVerilog. We will test the operation of these circuits using the switches and LEDs onboard the Nexys A7. Then, we will use these two modules to configure other, more complex circuits. First, we will use our greater-than and equal-to modules as submodules to display all six comparison operations on the LEDs. Then, we will use them as submodules to display the comparison of two two-bit binary inputs.



Objectives.

- Design a logic circuit that meets a given functional specification
- Implement a logic circuit using SystemVerilog, including defining and instantiating modules
- Test the operation of a circuit using the input and output components provided by the Nexys A7

Materials and Equipment.

- Nexys A7-100T FPGA board with USB cable

Deliverables.

- In-person lab demonstration (per team)
- Lab report (per team)

Part 0: Set Up a Vivado Project

To begin the lab, we will set up a new Vivado project for Lab 3 and configure the Nexys A7 board. You can reference Lab 1 for a more detailed description of these steps.

1. Create a Xilinx Vivado project and import the provided constraints file (`main_constraints_NexysA7.xdc`) and top-level module (`lab03_top.sv`).

2. Connect your FPGA board, generate a bitstream, and program the FPGA.

> **Stop and Check:** Toggle the rightmost six switches (`SW[0]-SW[5]`) and check whether the corresponding LEDs (`LED[0]-LED[5]`) turn on and off.

Part 1: Implement SystemVerilog Modules for Greater-Than (G) and Equal-To (E)

In Lab 2, you designed a circuit that takes two inputs, A and B, and computes greater than (>) and equal to (==). In this section, you will create SystemVerilog modules for both circuits and test your design. Later, you will use these two modules to compose more complex circuits. The Boolean equations for greater (G) than and equal to (E) are given below for your reference:

$$G = A.\overline{B}$$
$$E = A.B + \overline{A}.\overline{B}$$

Each module in SystemVerilog is defined in its own file. The file is given the same name as the module. The following code snippet shows the general syntax for creating a module:

```
module <module_name>(
  input logic  <list_of_input_ports>,
  output logic <list_of_output_ports>);

endmodule
```

The following example defines a module for a NAND gate that takes two inputs, a and b. Note that each line of code is terminated with a semicolon (`;`).

```
module my_nand(
  input logic  a, b,
  output logic y );

  logic w1;
  and(w1, a, b);
  not(y, w1);

endmodule
```

3. Create a new source file for your `greater_than` module:
   a. In the Flow Navigator toolbar, select “Add Sources” listed under “Project Manager” to open the Add Sources wizard.
   b. Select “Add or create design sources.”
   c. Select “Create File.”
   d. Change the file type to SystemVerilog.
   e. Set the file name to `greater_than` (the file name and module name are the same). Then, click “OK” to create the new module.
   f. Click “Finish” to exit the Add Sources wizard.

4. Implement your `greater_than` module to perform the Boolean equation for G:
   a. Set the input and output ports for the module according to your design.
   b. Implement the module body according to your equation for G.

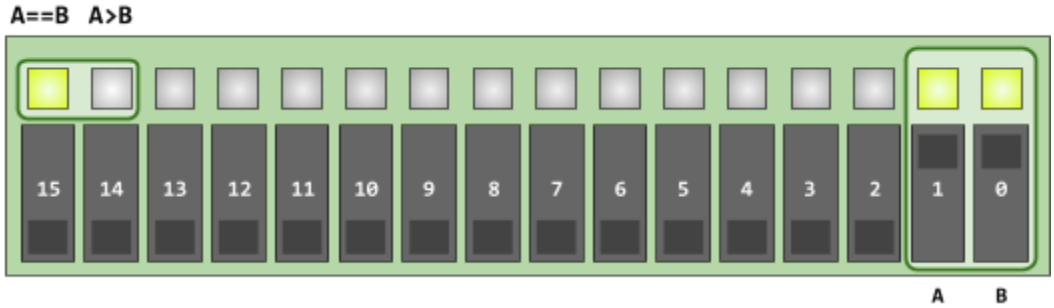


Figure 1. Input/Output Configuration for Part 1

5. Repeat Steps 3 and 4 to make an `equal_to` module that performs the Boolean equation for E.

6. Instantiate the `greater_than` and `equal_to` modules in your top-level module.
   a. To test our module implementation, we will instantiate each module inside of `lab03_top.sv` with the connections illustrated in Figure 1. Namely, we will use SW[1] for input A and SW[0] for input B. We will display the result of G on LED[14], and E on LED[15].
   b. Using named instantiation, create an instance of the `greater_than` module, connect the input ports to SW[0] and SW[1], and connect the output port to LED[14]. **The syntax for named instantiation is shown below.**
   c. Using named instantiation, create an instance of the `equal_to` module, connect the input ports to SW[0] and SW[1], and connect the output port to LED[15].

```
// General syntax
<module_name> <instance name>( .port(connected_wire), .port(connected_wire) );

// Instantiating my_nand
logic output_wire;
my_nand D0( .a(SW[0]), .b(SW[1]), .y(output_wire));
```

7. Check the circuit schematic under “RTL Analysis/Open Elaborated Design/Schematic” in the Flow Navigator. This circuit schematic is a visual representation of the connections created by `lab03_top.sv`. Each submodule is shown in blue. Click the **+** symbol to expand the submodules and look at their contents. How do they relate to the SystemVerilog code you have written?

8. Generate a bitstream, program the FPGA, and test that your implementations of G and E function correctly.

Stop and Check: Toggle the switches for inputs A and B to ensure that the implementation of G and E are working correctly.

Record: Screenshot the circuit schematic (produced by Vivado) for this step.

Part 2: Implement All Six Comparison Operators

Our FPGA now computes G (>) and E (==) and displays these results on the two leftmost LEDs. In this section, we will use these two signals to derive the remaining four relational operators: not-equal-to (!=), greater-than-or-equal-to (>=), less-than-or-equal-to (<=), and less-than (<). All six comparison operators will be displayed on the Nexys board, as shown in Figure 2.

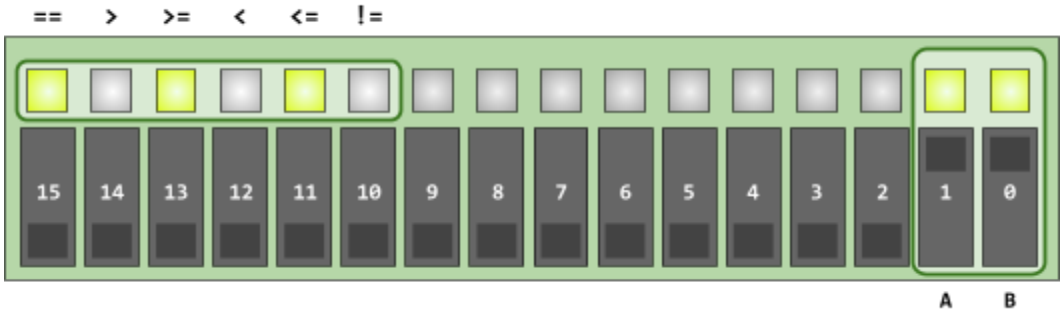


Figure 2. Input/Output Configuration for Part 2

To implement Part 2, you may need to modify `lab03_top.sv` to reuse the output for the greater-than and equal-to modules. Instead of connecting the module's output port directly to an LED (left), connect the port to a named wire (right). This change will allow us to use the named wire for other operations!

```
equal_to  D0(.a(SW[1]), .b(SW[0]),
             .y(LED[15]));
```

Connect output port directly to LED

```
logic e;

equal_to D0 (.a(SW[1]), .b(SW[0]),
             .y(e));

assign LED[15] = e;
```

Connect output port to named wire (e), and then to the LED

Let's begin by determining how we can derive all operators from the signals G and E.

9. Plan your design. Using a truth table, determine how you can derive each operator from your E and G functions. Sketch the circuit schematic.

10. Implement your circuit in the top-level module.
    a. If not done already, connect your submodules for G and E to named wires (shown above).
    b. Using those named wires and additional logic gates, derive the signals for the other four comparison operators: less-than, less-than-or-equal-to, greater-than-or-equal-to, and not-equal-to.
    c. Connect the signals to the corresponding LEDs, as shown in Figure 2.

11. Verify that the circuit schematic matches your proposed design. You can view the circuit schematic under "RTL Analysis/Open Elaborated Design/Schematic" in the Flow Navigator.

12. Generate a bitstream, program the FPGA, and test that your implementation functions correctly.

Stop and Check: Toggle the switches for inputs A and B to ensure that the implementation of each operator is working correctly.

Record: Screenshot the circuit schematic (produced by Vivado) for this step. Record your test cases (i.e., the combinations of A and B and the six outputs) to include in the Results section of your lab report.

Part 3: Implement a 2-bit Comparator

In digital design, we often use simple modules to build more complex designs. This is called *modular design*. In Part 2, we used our `greater_than` and `equal_to` modules to derive other comparator operators. In this part, we will scale up our `greater_than` and `equal_to` modules to compare two 2-bit numbers, instead of comparing just A and B.

Previously, we used our modules to compare A and B, which can have the value 0 or 1. Binary is a number system where each digit is a Boolean value (0 or 1). By stringing together multiple binary digits, we can represent larger numbers (*e.g.,* 2, 3, 4, *etc.*). As shown in Table 1, we can string together two binary digits to represent the decimal numbers 0-3. Because these numbers have two digits of binary, they are referred to as 2-bit numbers.

Decimal	Binary	Switch Positions
0	00	OFF-OFF
1	01	OFF-ON
2	10	ON-OFF
3	11	ON-ON

Table 1. 2-Bit Binary Numbers

In this part, we will use our 1-bit binary comparators to build a 2-bit comparator. Adjacent switches on the Nexys A7 board will encode each 2-bit binary number, as shown in Table 1 and Figure 3. For example, if switch SW[3] is ON and SW[2] is OFF, then the value of the signal `M` is 2 in decimal. If switch SW[5] is OFF and SW[4] is OFF, then the value of the signal `N` is 0 in decimal. For this set of inputs, both G and E will be false since `N!=M` and `N<M`.

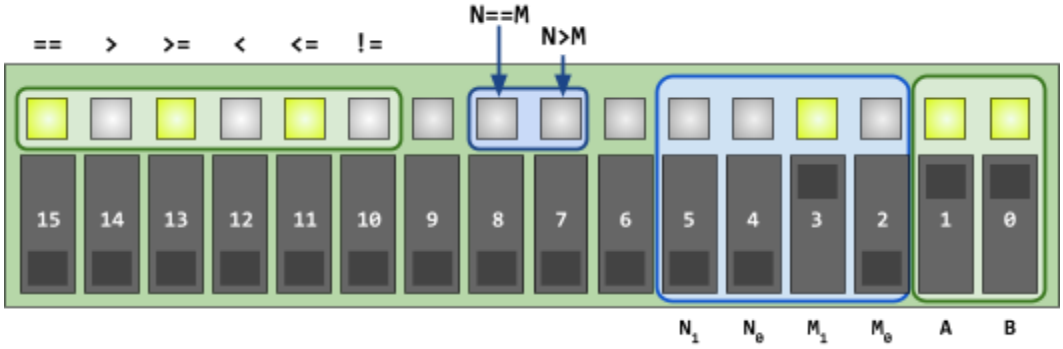


Figure 3. Input/Output Configuration for Part 3

13. Plan your design to compute 2-bit equality. Determine how you can test the equality of two 2-bit numbers using your 1-bit `equal-to` module. *(Hint: Check the equality of each individual bit, then combine them with logic gates*)

14. Implement 2-bit equality in the top-level module. You can choose to make your 2-bit comparator its own submodule (this will improve the organization of your code), but it is not necessary for this assignment.
    a. Do not remove your code for Parts 1-2! Implement Part 3 simultaneously on the board using different switches and LEDs. This layout is highlighted in blue in Figure 3.
    b. Instantiate one or multiple submodules and connect the input ports to the appropriate switches.
    c. Connect the output of your 2-bit equality comparator to LED[8], as shown in Figure 3.

15. Before moving on to implement 2-bit greater-than, you may opt to test your design.
    a. Verify that the circuit schematic matches your proposed design. You can view the circuit schematic under "RTL Analysis/Open Elaborated Design/Schematic" in the Flow Navigator.
    b. Generate a bitstream, program the FPGA, and test that your implementation functions correctly based on the values of SW[5]-SW[2].

> **Stop and Check:** Toggle the switches for inputs N and M to ensure that the implementation of equality is working correctly.

16. Plan your design to compute 2-bit greater than (>). Determine how you can compare two 2-bit numbers using your 1-bit `greater-than` module and your 1-bit `equal-to` module.

17. Implement 2-bit greater than in the top-level module.
    a. Instantiate one or multiple submodules and connect the input ports to the appropriate switches.
    b. Connect the output of your 2-bit greater-than comparator to LED[7], as shown in Figure 3.

18. Verify that the circuit schematic matches your proposed design. You can view the circuit schematic under “RTL Analysis/Open Elaborated Design/Schematic” in the Flow Navigator.

19. Generate a bitstream, program the FPGA, and test that your implementation functions correctly based on the values of SW[5]-SW[2].

Stop and Check: Toggle the switches for inputs N and M to ensure that the implementation of each operator is working correctly.

Record: Screenshot the circuit schematic (produced by Vivado) for this step. Record your test cases (i.e., the combinations of N and M and the two outputs) to include in the Results section of your lab report.

Part 4: Wrap-Up

Before leaving the lab, make sure to complete the following steps!

20. ***Demonstrate the operation of your circuit to the instructor.*** A portion of your lab grade will come from the lab report in addition to this demonstration.

21. Turn the Nexys board OFF and disconnect it from the computer. Make sure to save your Vivado project and sign out of your machine!

Lab Report *(Due at beginning of next lab session)*

Lab Report. In collaboration with your lab partner, submit one lab report on Moodle that describes the lab activity and your team's circuit design. Your written report is worth 60% of your grade for each lab assignment. Your lab report should include the following sections:

- **Title Page:** Your lab report should begin with a title block that includes the course number, the title of the lab assignment, the names of each team member, and the date of the lab assignment. The title page should also include a "Statement of Collaboration" (detailed below) and an estimate of the total amount of time spent on the lab assignment. The inclusion of these components are graded, but the content of each statement is not used in determining your grade.

- **Statement of Collaboration:** Describe the contributions of each team member to the lab assignment, including writing of the lab report. What is specified in this section will not affect your grade, however it is expected that you periodically rotate roles within your lab group if you do divide work, including writing the report.

- **Introduction:** Include a brief paragraph summarizing the objectives of the lab and what your team achieved. This section should be written in your own words; it is not acceptable to copy text for this from the lab handout.

- **Design:** Include a concise yet descriptive explanation of the circuit design, following all steps in the process—from the problem specification and initial truth table, to equations and circuit schematics. Tables and equations should be typeset and clearly labeled. Technical information should be accompanied by some text narrating each step or aspect of the design.

- **Testing & Results:** Describe how you tested your circuit, including rationale for why you chose specific test cases if the circuit is not tested exhaustively. If you tested your circuit at multiple points (*i.e.,* using a testbench and then on the Nexys board), this section should include all test cases at each point. Test cases can be included in the form of tables or simulation waveforms. Results should be neatly formatted and labeled, indicating what step is being tested. This section should also include any recorded measurements, tests, diagrams, or images specified by the lab document.

- **Discussion/Conclusion:** Your discussion/conclusion section should comment on any observations about the technique you employed to get your results. Briefly describe any difficulties that you encountered and how you resolved them. Then, summarize what you concluded from your work. (It may also be helpful to think negatively, *i.e.* what your results did not show. Remember that a negative result is often as valuable as a positive result!)

  A number of technical questions may be posed in the assignment sheet. If questions are given, include your answers to them in this section.

Your lab report should be typed and submitted as a .pdf document. Only one member per team needs to submit the lab report.