Lab 5: Binary Adders/Subtractors

ECE 211: Digital Circuits I — Spring 2025

Abstract. In this lab, we will build SystemVerilog modules at increasing levels of abstraction to realize a circuit that performs addition and subtraction. We will begin by using gate-level modeling to implement a half adder and a full adder circuit, based on the Boolean equations we derived during the lecture. Then, we will connect these building blocks to construct a 6-bit adder—a circuit that adds two 6-bit binary numbers. Using our 6-bit adder and 2's complement representation, we will then implement a circuit that can toggle between addition and subtraction operations, displaying the result of the Nexys A7's seven-segment display.

While we will continue to use gate-level modeling in this lab, we will only be instantiating logic gates within the half/full adder modules. The remaining parts of this lab work at a *higher level of abstraction*, using the half/full adder as a black-box that performs a specific function. This lab takes advantage of the modularity of the half/full adder to construct more interesting circuits.

Objectives.

- Use ***half adders*** and ***full adders*** to implement a circuit that performs 6-bit addition
- Use ***2's complement*** format to implement a circuit that performs 6-bit subtraction
- Implement a logic circuit using gate-level modeling in SystemVerilog
- 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 5 and configure the Nexys A7 board. You can reference Lab 0 for a more detailed description of these steps.

1. Create a Xilinx Vivado project and import the provided constraints file (`*.xdc`) and SystemVerilog modules (`*.sv`).

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

> **Stop and Check:** Toggle the switches switches (`SW[0]-SW[15]`) and check whether the corresponding LEDs (`LED[0]-LED[15]`) turn on and off.

Part 1: Implement Modules for a Half Adder and a Full Adder

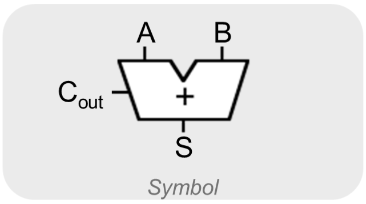
In this part, you will implement a SystemVerilog module for the half-adder and full-adder circuits presented in lecture. A half adder computes the sum (S) of two bits, A and B. The carry out (C_{out}) of a half adder is high when the result of the addition cannot fit into the single sum bit. When chaining adders together, the C_{out} is used as the carry to the next column of addition.

A full adder computes the sum of two bits, A and B, plus a carry from a previous column, termed C_{in}. A full adder also produces a sum bit (S) and a carry out (C_{out}). The meaning of these outputs is identical to that of the half adder. The truth table, equations, and schematics for half and full adders are shown in Figure 1 and Figure 2.

A	*B*	*S*	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth Table

$$S = A \oplus B$$
$$C_{out} = A.B$$



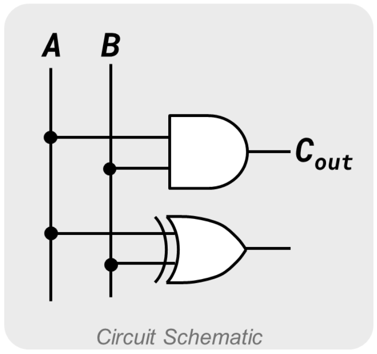
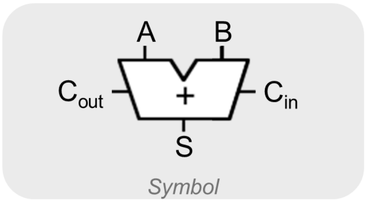


Figure 1. Half Adder Truth Tables, Equations, and Schematic

A	*B*	C_{in}	*S*	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth Table

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = A.B + A.C_{in} + B.C_{in}$$



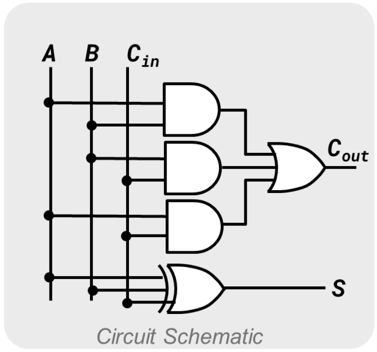


Figure 2. Full Adder Truth Tables, Equations, and Schematic

3. Implement a `half_adder` module:
    a. Add a new source file to your project for `half_adder.sv`
    b. Your module should take two scalar inputs, A and B
    c. Your module should produce two scalar outputs, S and C_{out}
    d. Implement the module body according to the equations for S and C_{out}

4. Implement a `full_adder` module:
    a. Add a new source file to your project for `full_adder.sv`
    b. Your module should take three scalar inputs, A, B, and C_{in}

c. Your module should produce two scalar outputs, S and C_{out}

d. Implement the module body according to the equations for S and C_{out}

5. It is recommended that you test your modules before moving to the next part of the lab. Incrementally testing your modules allows you to more easily identify mistakes and saves you from debugging the *entire* lab at the same time. You can test this part of the lab by instantiating each module in the top-level file and connecting them directly to switches and LEDs on the Nexys board.

Record: Record the schematics of your half and full adder for inclusion in your lab report.

Part 2: Implement a 6-bit Adder Module

In Part 1 of this lab, we implemented a half adder and a full adder using ***gate-level*** structural descriptions—instantiating gate primitives (*e.g.*, **`and`**, **`or`**, **`not`**) to construct the low-level wiring of a circuit. By encapsulating this low-level functionality in a module, we can take advantage of the module's reusability to build more interesting circuits without having to worry about low-level wiring. In this part, we will instantiate and wire half/full adders to build a 6-bit adder (a circuit that adds two 6-bit binary numbers). Your 6-bit adder module should take in the inputs A and B as ***6-bit vectors*** and return the output Y as a 6-bit vector.

$$\begin{array}{rr} & A_3A_2A_1A_0 \\ + & B_3B_2B_1B_0 \\ \hline & S_3S_2S_1S_0 \end{array}$$

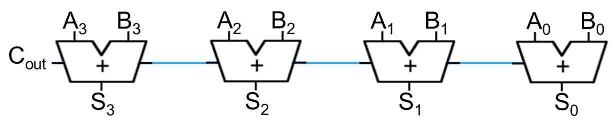


Figure 3. 4-bit Adder

A schematic of a 4-bit adder is shown in Figure 3. We can create wider adders by connecting additional full adders in a similar fashion. This circuit is often referred to as a ***ripple-carry adder*** because the carry out bit produced by each state will "ripple" the left.

6. Implement a `adder_6b` module:

   a. Add a new source file to your project for `adder_6b.sv`
   (*Note that SystemVerilog does not allow names to start with numbers, so* `6b_adder` *is not a valid module name*)

   b. Your module should take two 6-bit vector inputs, A and B

   c. Your module should produce a 6-bit vector output Y

   d. Implement the module body by instantiating half/full adder modules to construct a 6-bit ripple adder.

7. It is recommended that you test your module before moving to the next part of the lab. You can test your 6-bit adder by instantiating a module in the top-level file and connecting it directly to switches and LEDs on the Nexys board.

Record: Record the schematic of your 6-bit adder for inclusion in your lab report.

Part 3: Use 2's Complement Representation to Perform Subtraction

Next, you will use the modules you built in Part 1 and 2 to configure the Nexys A7 board to perform addition and subtraction. The input/output configuration for this step is shown in Figure 4:

- Switches `SW[7]-SW[4]` encode an unsigned 4-bit input **A**.
- Switches `SW[3]-SW[0]` encode an unsigned 4-bit input **B**.
- Switch `SW[15]` is used to toggle between addition and subtraction. Your circuit should perform addition when `SW[15]` is off, and subtraction when `SW[15]` is on.
- The result of the arithmetic, labeled **Y** in Figure 4, is displayed on the 7-segment display.

The inputs and outputs will also be displayed on the 7-segment display, where "-" is used to indicate subtraction and "P" is used to indicate addition (you cannot make a plus sign on a 7-segment display!). Information about how to connect the 7-segment display is given in the steps below.

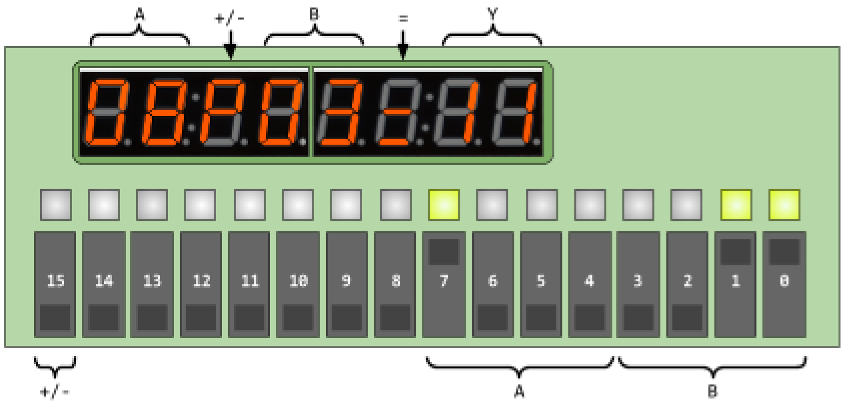


Figure 4. Input/Output Configuration for Lab 5

Implementing Subtraction. There are several ways you can configure your addition/subtraction circuit to complete this task. For example, you can instantiate one module to perform addition and one module to perform subtraction, then select which result to display based on the control bit `SW[15]`. This approach would require a multiplexor (MUX) to select which output to display on the seven-segment display. Alternatively, you could instantiate one addition module and select whether B or the 2's complement form of B is sent to the adder (*i.e.,* if `SW[15]` indicates subtraction, the 2's complement form of B is sent to the adder). This approach can be done using either a MUX or an XOR gate. Both approaches will produce a functionally-correct design. However, the latter may require fewer logic gates, and thus can be viewed as the "more efficient" approach from the perspective of hardware cost.

8. Plan your design. Sketch a high-level circuit schematic that implements both addition and subtraction using your 6-bit adder module(s). Your schematic should take two 6-bit inputs and produce a 6-bit output.

9. Implement your design in the top-level module `lab5_top`, or by making and instantiating a new module if appropriate.

   If you use a MUX in your implementation, you can use the following SystemVerilog ***ternary operator*** (`?:`) to represent a MUX in your program:

```
// The SV ternary operator (?:) synthesizes to a multiplexor (MUX).
// It assigns the output, f, based on the value of sel. Specifically, if
// sel is zero, f = d0. If sel is one, f = d1.
assign f = sel ? d1 : d0;
// The ternary operator works with scalars as well as vectors of any
// size! The selection bit, sel, must be a scalar.
```

10. Connect the inputs of your design to the switches `SW[7:4]`, `SW[3:0]` and `SW[15]`.

11. Connect the output of your design to the seven-segment display via the `calculator_display` module. The ports of this module that are relevant to this step are highlighted in the following code snippet:

```
    calculator_display DISP(
        .a( ),                  /* Connect input A here */
        .b( ),                  /* Connect input B here */
        .operation(SW[15]),     /* Indicates add or subtract */
        .y( ),                  /* Connect result Y here */
        .y_sign(1'b0),          /* Replace connection for Part 4*/
        .clk(clk), .seg_n(SEG), .an(AN));
```

12. 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.

13. Generate a bitstream and program the FPGA.

14. It is not feasible to test all possible cases. Instead, select a set of 8-10 representative test cases to check whether your circuit functions correctly. Record these test cases and the circuit’s output for inclusion in your lab report. Note any instances of overflow. Note that your circuit is performing signed addition and subtraction. **The outputs are 6-bit 2’s complement numbers, which may at first appear misleading!**

Stop and Check: Select 8-10 test cases for A +/- B. Administer these test cases and record the results of each test in a table.

Record: Screenshot the circuit schematic (produced by Vivado) for this step. Record your test cases and results to include in the Results section of your lab report.

Part 4: Improve Your Design

After administering your test cases, you may have noticed a flaw with our design. Negative numbers are displayed in 2's complement format. Because of this, it is hard to interpret the output. For example, the input 5 - 6 will display the result 63 instead of -1. This result may appear misleading to a regular calculator user! As a digital circuit designer, you can use your knowledge of 2's complement to understand that the result "63" is actually correct.

In this final part of the lab assignment, you are tasked with improving your circuit to display the sign and magnitude of your final result, allowing the human user to see "-1" instead of "63".

15. Display the sign and magnitude of the result, **Y**, on the seven-segment display.
    a. The most significant bit of the result, **Y**, indicates the sign of the number (1 for negative, 0 for positive). Connect this value to the `y_sign` pin of the `calculator_display` module.
    b. Add logic that calculates the magnitude of the result, **Y**. Connect this value to the y pin of the `calculator_display` module.
    c. The calculator display module is implemented to indicate the sign of the result. The following code snippet shows where you can connect signals from Step 15:

```
calculator_display DISP(
    .a( ),                  /* Connect input A here */
    .b( ),                  /* Connect input B here */
    .operation(SW[15]),     /* Indicates add or subtract */
    .y( ),                  /* Connect result Y here */
    .y_sign(1'b0),          /* Replace connection for Part 4*/
    .clk(clk), .seg_n(SEG), .an(AN));
```

Record: Record the schematics of update and a list of test cases for inclusion in your lab report.

Part 5: Wrap-Up

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

16. ***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.

17. 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.