ENGR 272 Lab 2:
Combinational Logic (Computer Arithmetic)

2.1 Background Information

Arithmetic Logic Units (ALUs) are a critical component of processors, performing all integer arithmetic operations for the computer, while floating-point operations are carried out in Floating Point Units (FPUs). Adders are used in ALUs to perform addition and subtraction and are often used in other parts of the processor for other simple operations such as incrementing or decrementing numbers.

There are three main ways of implementing multi-bit adders using combinational logic: Ripple-Carry, Carry-Lookahead, and Carry-Save. The simplest method, and the method to be implemented in this section, is the ripple-carry adder. It is significantly slower than the other two options, but it has the benefit of being small, easy to understand, and easy to implement. Carry-Lookahead adders use some additional logic to speed up calculations, at the cost of increased size and complexity. Carry-Save adders are even faster and even bigger than Carry-Lookahead adders.

Integrated circuit design often involves many tradeoffs that the designer must decide between. For instance, increasing speed will improve the circuit's performance, but the tradeoff is that increasing size also increases cost and power consumption. In some situations, a circuit designer may choose to use a slower implementation to keep cost and/or power consumption low.

2.2 Section Overview

Objective: Design and implement a 4-bit, ripple-carry adder using 8 switches for the inputs, and the LEDs on the FPGA to display the 5-bit output.

1. Create a block diagram for your design.
2. Fill out Table 2.1, the truth table for a full-adder.
3. Fill out Table 2.2, the partial truth table for a 4-bit adder.
4. Create a new project for this lab.
5. Create a schematic for a full-adder and generate a symbol.
6. Implement 4 of your full-adder symbols in a new schematic and connect them together to create a 4-bit adder.
7. Simulate the design of your 4-bit adder.
8. Program and test hardware.

2.3 Learning Objectives

In this section, the following items will be covered:

1. Creating block diagrams.
2. Creating and instantiating symbols in Quartus Prime schematics.
3. Simulating digital logic designs.

2.4 Materials

1. Quartus Prime Lite Edition V. 18.0/19.0
2. DE10-Lite kit with MAX10 10M50DAF484C7G FPGA
3. USB to USB-B cable
4. Chapter 5 of the textbook

2.5 Procedure

There are 6 steps to successful digital logic design. This process flow is crucial, and you will see it repeated throughout this course.

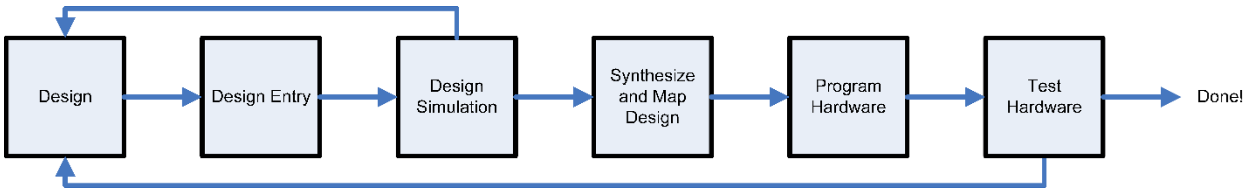

Figure 2.1: This six-step process is used for good digital logic design.

1. **Design:** The context of the design is established in this step. The context involves defining the inputs, desired outputs, and all the logic required in-between. In this step, all the minimizations and layout are planned for the design entry process. While this step is not always the longest, it should involve the most thought and effort. This typically requires a complete block diagram showing all the logic blocks and the connections between them, often with written explanations of specific functions.
2. **Design Entry:** The actual drafting of the digital logic design occurs in this step, translating the design from block diagrams and descriptions into the software. This can be accomplished directly by writing HDL code, or graphically by drawing a schematic that a software tool can convert into HDL code.
3. **Design Simulation:** Before committing to hardware, this step tests the design in a controlled computer simulation. If the design does not function as specified in the "Design" step, it is revised.
4. **Synthesize and Map Design:** When the design simulates correctly, the HDL and schematic source files are synthesized into a design file that can be written to the FPGA. This includes assigning the inputs and outputs of the design to IO pins.
5. **Program Hardware:** After the design file is created, it is used to configure the FPGA. Quartus Prime sends a bit stream over the USB-B cable to configure the DE10-Lite FPGA.
6. **Test Hardware:** Verify hardware operation once the FPGA has been programmed. The FPGA should operate exactly as the design predicted, which was verified by the simulation. Synthesis

problems, timing violations, or incorrect assumptions about the hardware can require the designer to return to the “Design” step.

2.6 Design

2.6.1 Make a block diagram

Review section 1.5.2 for an overview of block diagrams.

Two examples of block diagrams are shown below in Figure 2.2. The diagram on the left is a high-level description of the adder, showing the entire design as a single block with two 4-bit inputs and one 5-bit output, connected to LEDs. The diagram on the right goes one level deeper, showing that the 4-bit Adder block shown on the left is made up of 4 full-adders connected together. Block diagrams with different levels of abstraction are useful in different situations, for instance, the high-level diagram on the left is better for understanding the overall design, but the diagram on the right is more useful for actually implementing the design.

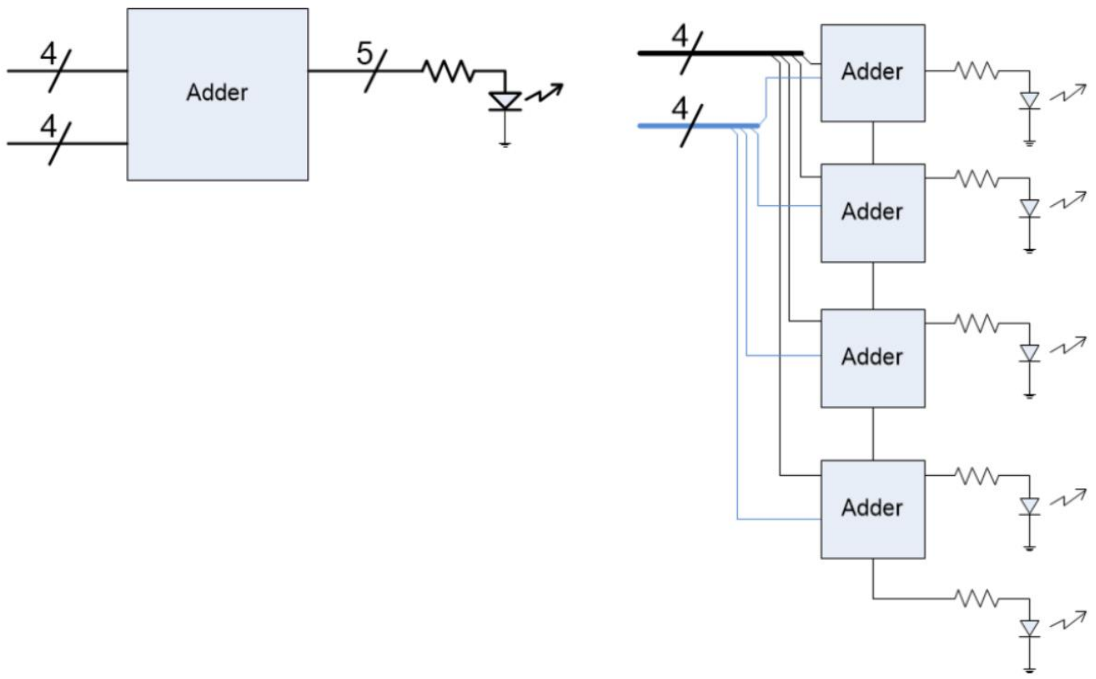

Figure 2.2: Two Block Diagram Options

Follow these steps to create a complete block diagram from Figure 2.2:

1. This design will have one main block (Top Level) for the FPGA
2. The design will include a sub-block (1-bit Full-Adder)
3. You may use the Quartus to make these block diagrams.

2.6.2 Make a functional truth table

Fill out the truth table below to describe the functionality of a Full-Adder. Four of these will be linked together to create a 4-bit, ripple-carry adder. A description of a full adder is shown at the beginning of Chapter 5 in the textbook.

Operand1	+	Operand2	+	Carry In	=	Value (2-bit Binary)	Value (Unsigned Decimal)
0b0	+	0b0	+	0b0	=		
0b0	+	0b0	+	0b1	=		
0b0	+	0b1	+	0b0	=		
0b0	+	0b1	+	0b1	=		
0b1	+	0b0	+	0b0	=		
0b1	+	0b0	+	0b1	=		
0b1	+	0b1	+	0b0	=		
0b1	+	0b1	+	0b1	=		

Table 2.1: Full-Adder Truth Table

Fill out the partial truth table below to describe the functionality of a 4-bit adder. Later in this section, you will compare this truth table to your simulation results to verify your design.

Operand1	+	Operand2	=	Value (5-bit Binary)	Value (Unsigned Decimal)	Value (Hexadecimal)
0b0000	+	0b0000	=			
0b0000	+	0b0001	=			
0b0000	+	0b0010	=			
0b0000	+	0b0011	=			
0b0000	+	0b0100	=			
0b0000	+	0b0101	=			
0b1000	+	0b0000	=			
0b1000	+	0b0001	=			
0b1000	+	0b0010	=			
0b1000	+	0b0011	=			
0b1111	+	0b0011	=			
0b1111	+	0b1000	=			
0b1111	+	0b1010	=			
0b1111	+	0b1011	=			

Table 2.2: Partial 4-bit Adder Truth Table

This table is incomplete; there are actually 256 (28) possible combinations! Complex designs can often be broken into several simple components like you're doing here by building a 4-bit adder (256 IO combinations) out of 4 full-adders (8 IO combinations each).

2.6.3 Design the Logic

In the next lab, you will learn to go from a truth table to logic gates using Karnaugh Maps (Section 2.7.2 of Textbook). For this lab, the Full-Adder Logic is given to you in the Sum of Products Form (Section 2.2.2 of Textbook)

Read the Textbook for more information about this form of writing logic and how it is generated. In short, things added together are OR gated together, things multiplied are AND gated together. $\oplus$ is XOR. Follow the Order of operations.

SUM = A $\oplus$ B $\oplus$ Cin

Cout = AB + ACin + BCin

2.7 Design Entry

1. Create a new Project in Quartus, make two schematic files, and enter the Full-Adder design in the schematic file that is not the top level. Be sure to label the inputs and outputs in this schematic! If needed, refer back to Section 1.6 for instructions on how to create a new project and add schematic files.

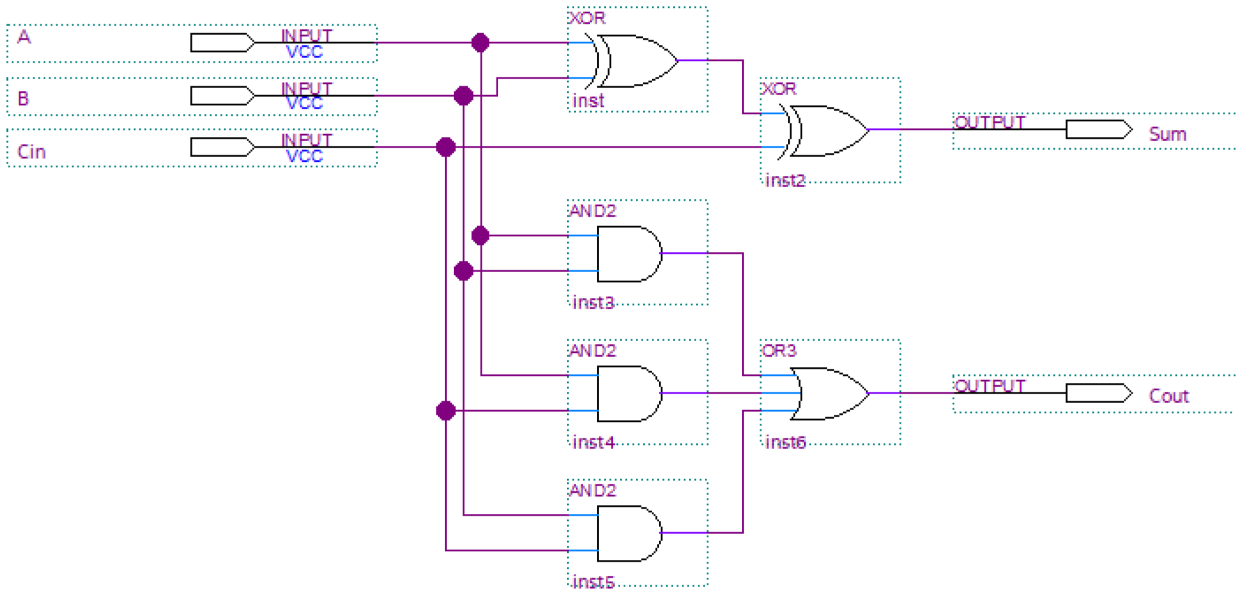

Figure 2.3: FullAdder schematic from Sum of Products Equations

2. Generate a symbol from the full-adder schematic by selecting File→ Create/Update →Create Symbol file for the current file. This will add a new symbol into the [Project] library of the Symbol menu in the schematic editor.

3. In the Top-level file instantiate that symbol 4 times, then connect those full-adders together to create a 4-bit adder.
4. Label the 4-bit number inputs A0-A3 and B0-B3 and outputs Z0-Z4, with 0 being the least-significant bit, and 3 being the most-significant.
5. Label the carry-in input to the least-significant Full-Adder. This won't be connected to a button, but it will need to be assigned to a pin so that you can use a pull resistor to keep that input HIGH or LOW.

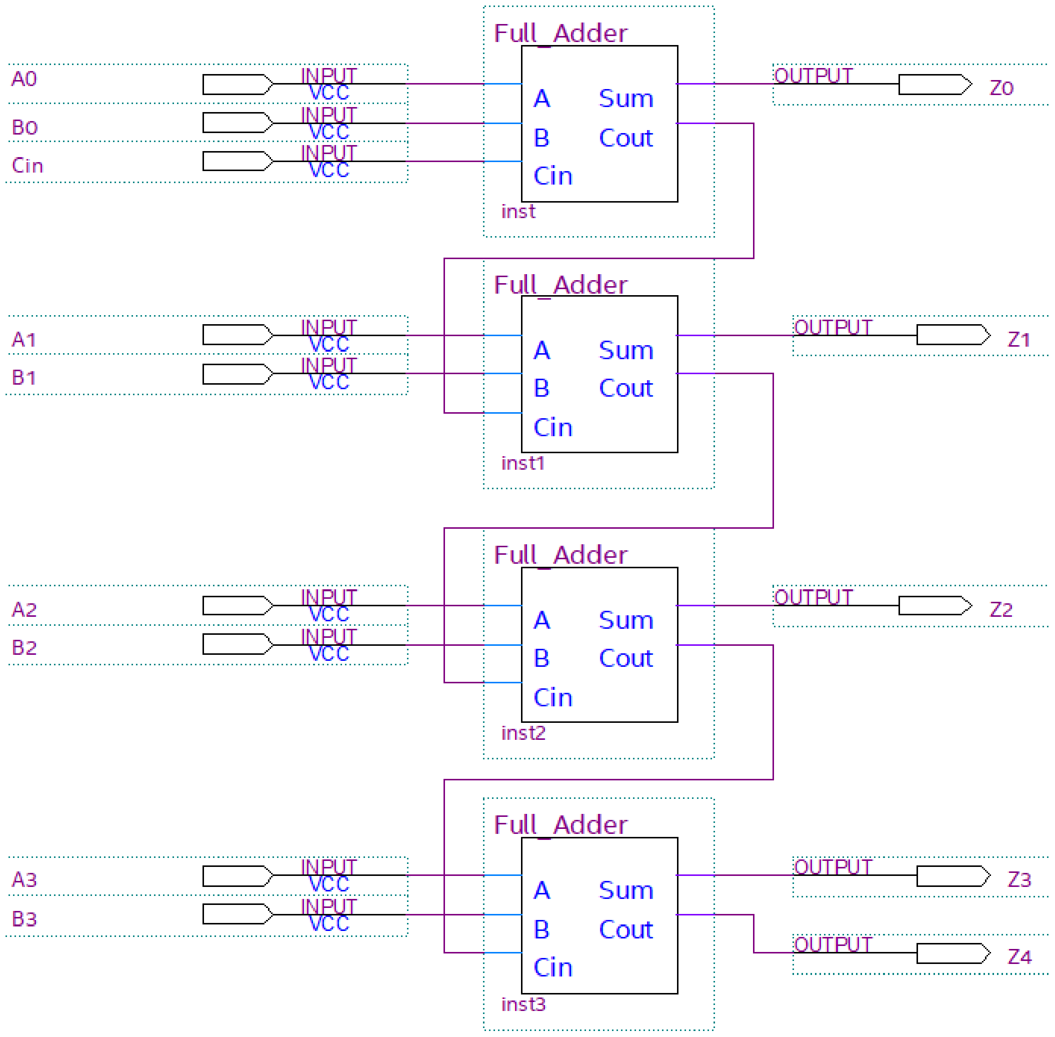

Figure 2.4: Top Level schematic of 4-bit adder circuit

2.8 Design Simulation

Simulation is done with Modelsim in this course using the same process as in Section 1.7. See the links in Moodle for information on the software. You do not need to simulate every combination, but you must at least simulate the different combinations from Table 2.2

To generate the Verilog Hardware Description Language file for use in Modelsim, go to File -> create/update -> create HDL Design File for current file -> Verilog (You will need to do this for each module).

Find and open the **.v** files in Modelsim and simulate your design. This will need to be done for both the Full-Adder schematic and the Top-level 4-bit adder. You will need to open both files in ModelSim for the simulation to work correctly.

2.9 Synthesize and Map Design

Follow the same process for using the Assignment Editor and compile your design as in Section 1.8.

2.10 Program Hardware

Program the DE10-Lite using the same process as in Section 1.9.

2.11 Test Hardware

Validate that the hardware performs according to Table 2.2.

2.12 Checkoff

1. Verify that Table 2.1 is filled out
2. Verify that Table 2.2 is filled out
3. Verify that the Hardware output matches the tables
4. Verify the simulation in ModelSim matches the expected Results

2.13 Study Questions

The study questions go at the end of the Lab Report

1. Explain how you would convert your 4-bit adder to a 4-bit adder/subtractor.
2. In your own words, explain what pull resistors do in the FPGA.
3. Explain your selection for the pull mode for the pin connected to the least-significant full-adder's carry-in.

2.14 Challenge - Extra Credit

This section designed, built, and tested (simulation and hardware validation) a simple 4-bit adder. Now, modify your design to include a 9th input control pin (Information on ALUs can be found in chapter 5 in your textbook). Full credit will be given for the design, simulation, and implementation of the FPGA.

1. Control pin low: (Operand 1) + (Operand 2)
2. Control high: (Operand 1) - (Operand 2)

2.15 Report Rubric

The reports for the labs are to be formal reports and are to be completed using a word processing application. Please turn in the report to Moodle in a PDF or MS Word format (No Google Docs share links).

Your report should contain at a minimum the following items:

1. **Title**
2. **Date**
3. **Name**
4. **Objectives**
5. **Equipment/Parts/Materials**
6. **Procedure**
    a. **Design**
    b. **Design Entry**
        i. **Block Diagrams (like the ones shown in Figure 2.2)**
        ii. **All schematics you created in Quartus Prime (Screen Captures).**
        iii. **A screen capture of what pin assignments you made in the Assignment Editor in Quartus.**
    c. **Design Simulation**
        i. **Table 2.1: Full-Adder Truth Table completed.**
        ii. **Table 2.2: Partial 4-bit Adder Truth Table completed.**
        iii. **ModelSim simulation figures (Screen Captures).**
7. **Complete the study questions in section 2.13**
8. **Challenge - Extra Credit (If this was done)**
9. **Observations**
10. **Conclusion**
11. **References**
12. **Appendix (Source code of Verilog files used for your simulations).**

All images need to have a title and a figure number. For more detailed information about what goes into each of the different sections, see “ENGR 272 Guidelines for Technical Lab Reports” in Moodle.

For an example of what your reports should look like see “ENGR 272 Example Report” in Moodle.

References:

1. Oregon State University ECE 272: Section 2: Adders on an FPGA (2019) https://eecs.oregonstate.edu/tekbots/courses/ece272/section2

2. Harris, S. L. & Harris, D. H. (2016) Digital Design and Computer Architecture: ARM Edition 1st Edition, Waltham, MA: Elsevier.