LC Lab Project
Smart ALU

January 2024

1 Introduction

In this project your job is to design and implement a circuit that uses both combinational and sequential logic. To put it simply, you have to design an ALU (Arithmetic logic unit), and a tester that uses a random sequence to test the functionality of the ALU.

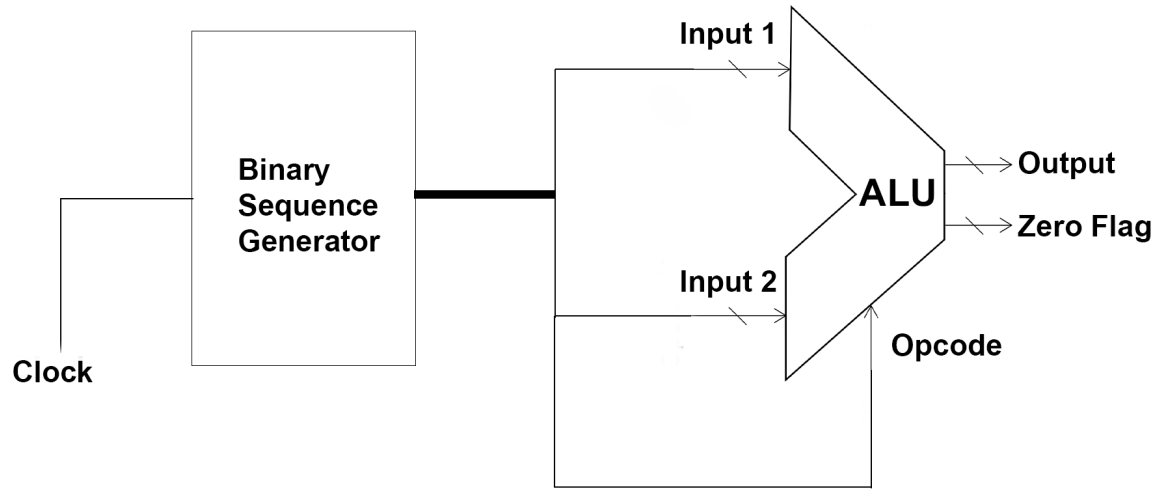

Figure 1: A basic schematic of the project

2 Phases

Here the parts you need to implement are listed. We suggest that you implement these parts in the same order that they're listed in the document.

Table 1: ALU opcodes and operations.

Opcode	Operation	Output
000	Add	Output = Input1 + Input2
001	Sub	Output = Input1 - Input2
010	Mult	Output = Input1 * Input2
011	Div	Output = Input1 / Input2
100	Bitwise And	Output = Input1 & Input2
101	Bitwise Xor	Output = Input1 ^Input2
110	Pass Input2	Output = Input2
111	Modulo	Output = Input1 %Input2

2.1 ALU

Here we want to implement an ALU that gets two 2-bit inputs, and produces a 4-bit output based on the value of the opcode.
What is opcode? It's the operation code, or the code that determines the mode of the ALU. for example if the opcode is 000, then ALU should add the two inputs. Table 1 shows the full list of opcodes and ALU operations. You can implement a 4-bit ALU to get bonus points. The differences are explained in in section 3.2 .

- All operations are unsigned.
- For division, use Q3 (MSB) and Q2 to show remainder, and Q1 and Q0 (LSB) to show quotient.
- If Input2 is 0 in division or modulo operations (resulting in division by 0), the output should be 1111.
- You should also implement zero flag, which is 1 if the output of the ALU is 0.

2.2 Binary Sequence Generator

In order to test the ALU, design a module that uses the sequence shown in Table 2 to test the ALU. In other words, you have to design a counter that counts the sequence shown in Figure 2.

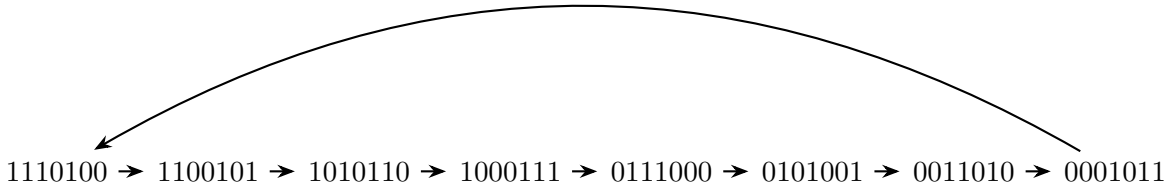

Figure 2: Binary Sequence for 2-bit ALU.

Table 2: The sequence and the outputs.

Opcode	Input1	Input2	Sequence	Output
111	01	00	111_01_00	1111
110	01	01	110_01_01	0001
101	01	10	101_01_10	0011
100	01	11	100_01_11	0001
011	10	00	011_10_00	1111 (Div by 0)
010	10	01	010_10_01	0010
001	10	10	001_10_10	0000
000	10	11	000_10_11	0101

2.3 Multiplexer

You also need to implement a 2-to-1 multiplexer that uses a manual select signal. When the signal is set to 0, the output of the ALU should come from the Binary Sequence Generator, when it is set to 1, it should come from the user input (Use logic state device in Proteus for manual input).

3 Bonuses

If you have missed any lab classes, doing these bonus parts can be a good way to get closer to that sweat, absolutely worthless perfect score :).

3.1 7-Segment

Implement binary to 7-segment decoder, and show the result of the ALU using that. Use 7SEG-DIGITAL in Proteus if you wish to implement this part. Using devices like 7SEG-BCD that convert binary to BCD on their own will not give you any bonus.

3.2 4-bit everything

Everything is the same as the 2-bit ALU, but you have to connect more wires, and there are more points of failure. Do this part if you absolutely have to. Naturally instead of a 7-bit random sequence, you now have to generate a 11-bit sequence. And you have to implement a 2-to-1 , 11-bit MUX as well. In division, use Q7 (MSB), Q6, Q5, and Q4 to show remainder, and Q3, Q2, Q1, and Q0 (LSB) to show quotient. For divison by 0, the output of the ALU shoud be 11111111. The sequence you have to implement is shown in Figure 3. Again,
DO NOT DO THIS PART UNLESS YOU REALLY REALLY HAVE TO. IT IS NOT WORTH IT.

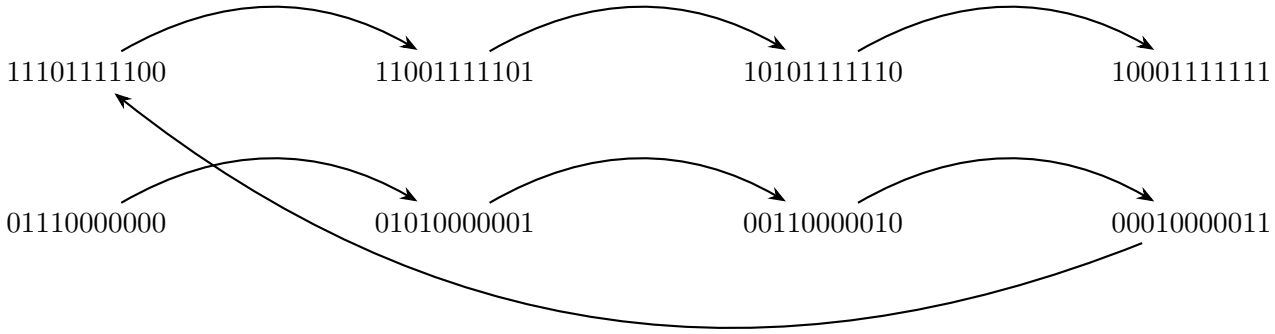

Figure 3: Binary Sequence for 4-bit ALU.

3.3 Implementing FlipFlops

You can use devices like 7473 IC (JK flip flop) that are available, in Proteus, but implementing these modules will give you bonus points. If you decide to implement the flipflops yourself, be sure to add a reset pin, or you module might not work.

4 Other important stuff

- Give **EVERYTHING** you use in this project a unique name, every gate, input, output,... should be named. Also avoid similar names at all if you can even for completely different modules, because Proteus is as buggy as Xilinx ISE and using similar names can lead to a lot of problems.
- While you can use ICs like 7473 in Proteus for Flipflops, any other module like MUXes should be implemented by yourself.
- To generate the sequence, you can use more than one counter.
- If you are not comfortable implementing a 7-bit 2-to-1 multiplexer, you can implement several 2-to-1 multiplexers that have the same select signal.
- Do **NOT** copy from your classmates' answers :)).
- Feel free to ask any questions.
- The deadline of the project will be determined later, but we'll give you as much time as we can.

Good Luck! And never stop fighting.
MohammadReza Naziri, Koosha Zandparsa, Ghazal Nikseresht
Mehdi Zokaie, Reza Hesami, Arman Alavi