Lab Report 2

***On my honor, I pledge that I have not violated the provisions of the NJIT Student Honor Code***

STUDENT TEAM N**o** \_\_2\_\_

Name Signature

…Arsh Bhamla…………  ***\_\_\_ARSH BHAMLA\_\_\_\_\_\_***  …Christopher Karkenny… ***Christopher Karkenny***.. …Pranay KC…………..  ***\_\_\_PRANAY RAJ KC\_\_\_\_\_***  ..Shreedhar Rajendran. ***SHREEDHAR RAJENDRAN***

# 

Experiment performed on date \_01/ 24-31 / 2025\_\_

Report submitted on date 02/07/2025

Returned for corrections on date \_\_\_\_\_\_\_\_\_ Grade \_\_\_\_\_\_

Returned after corrections on date \_\_\_\_\_\_\_\_\_ Grade \_\_\_\_\_\_

## 

## Introduction

In this lab, we will look at sequential logic circuits, which are different from combinational logic circuits because they include memories which allow the output to be based on both the current inputs and previous states. While combinational circuits are made up of basic logic gates such as NAND, NOR, and inverters, their logic functions are usually expressed by Boolean functions. These functions are usually reduced before making the circuit to reduce the number of gates required. Simplification can be done with methods such as Karnaugh maps or simple Boolean algebra laws. This lab will cover sequential circuits and methods for simplifying Boolean functions.

## Procedure

**Part 1 - 3.1:**

First XOR Gate Diagram:

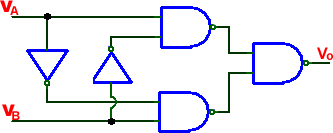


Figure 1. Retrieved from Lab 1: Combinational Circuits website [here](https://ecelabs.njit.edu/ece294/lab1.php).

Second XOR Gate Diagram:

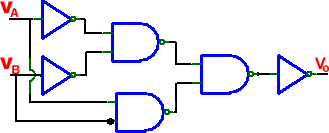


Figure 2. Retrieved from Lab 1: Combinational Circuits website [here](https://ecelabs.njit.edu/ece294/lab1.php).

**Part 2 - 3.2:**

For this part, we take the XOR gate that we had created in part 3.1, and add an AND gate to the circuit to create a half adder, giving us a sum bit and a carry bit through the XOR gate and the AND gate each. (XOR gate will have x XOR y = Sum bit, and AND gate will have xy = Carry bit)

**Part 3 - 3.3:**

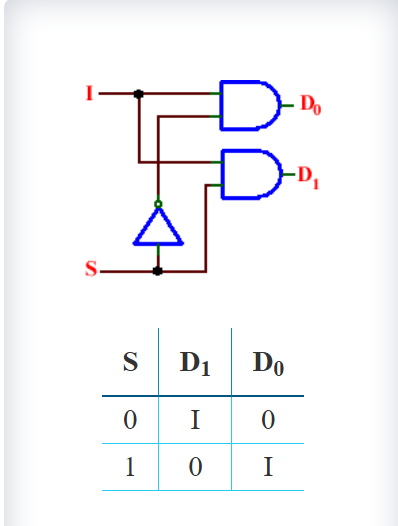
For this part, we take the half adder we built in part 3.2, and now put through the XOR gate the Carry bit that we got from the half adder and the sum bit. Doing this gives us a new sum bit. For the new carry bit, we take the carry bit we got from the half adder, and put it through an OR gate with the AND gate output from Carry bit = xy as well as F=Cin(x XOR y), giving us a new carry bit equation Cin1 = xy+Cin(x XOR y).

**Part 4 - 3.4:**

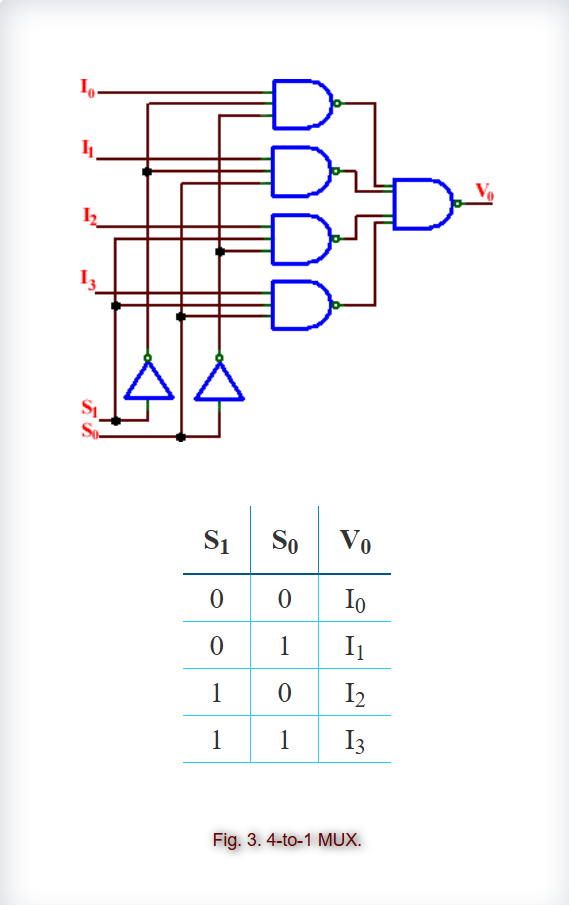
For this part, we create a MUX using the specified instructions of outputting a 1 for months with 31 days and 0 for any other month. To make it easier for ourselves, we create an MEV with VEM = D, and create a K-Map using it. After that, we simply make the MUX as the K-Map shows us, to give us a 1 or 0 as an output depending on the inputs we feed into it. We will be using the 7451 MUX IC that we were provided in order to perform this experiment.

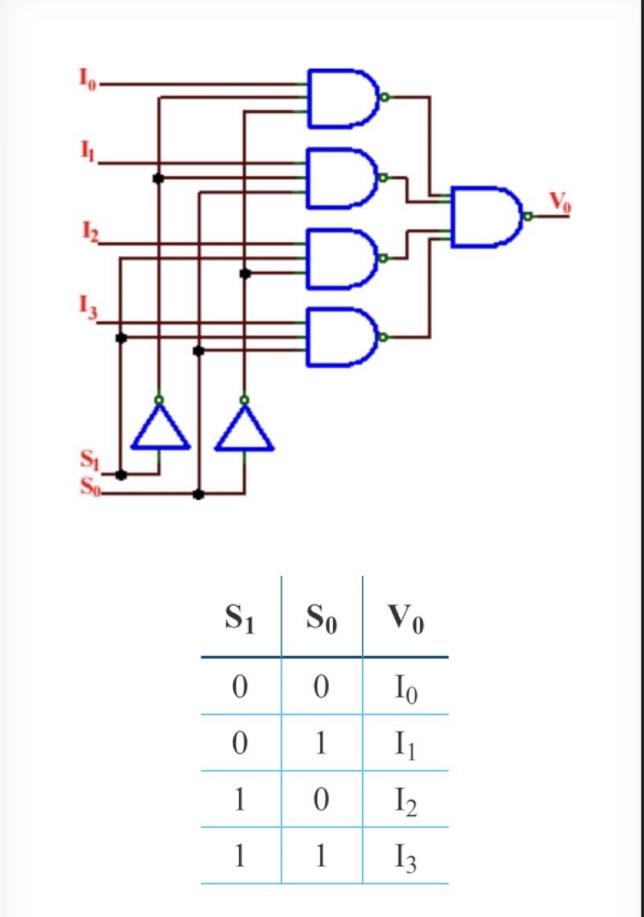
**Part 5 - 3.5:**

A



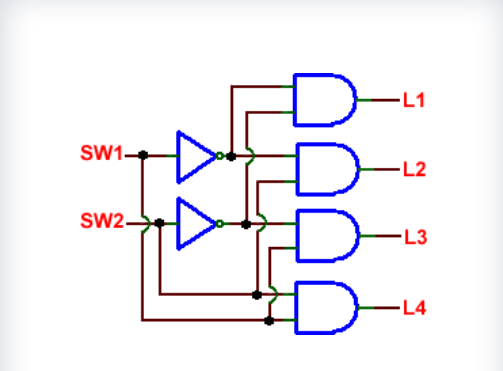
B





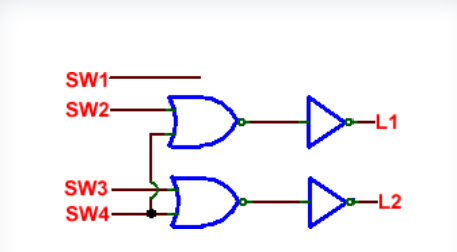
**Part 6:**

A



**Part 7:**

A



(Note: All the circuit diagrams in this section for parts 1-7 were provided from the ECE lab website [here](https://ecelabs.njit.edu/ece294/lab1.php))

## Data and Calculations

**Part 1:**

Figure 1:



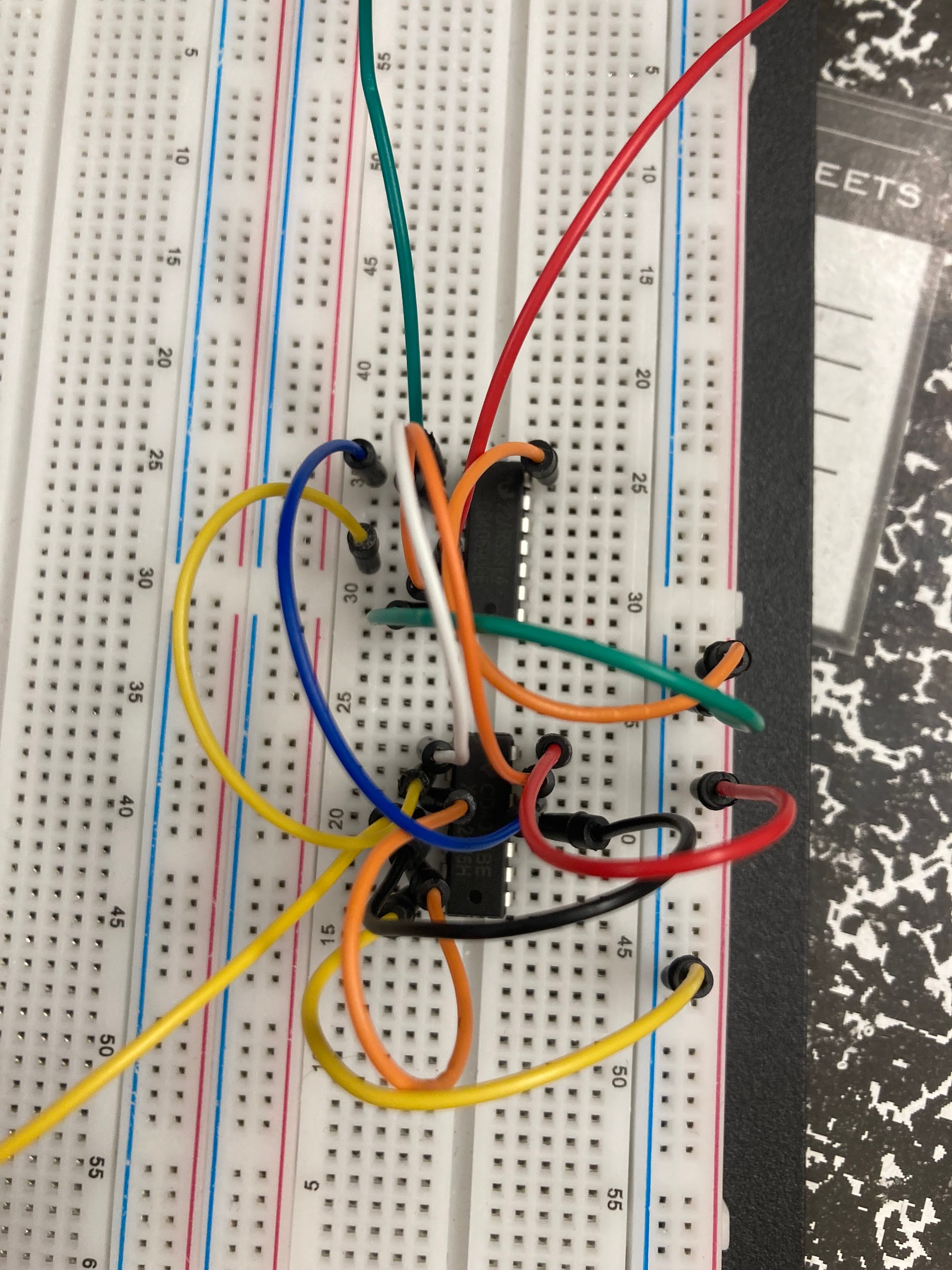
|  |  |
| --- | --- |

Full demonstration done in class.

Truth Table

| Input 1 | Input 2 | Output |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Figure 2:



Truth Table

| Input 1 | Input 2 | Output |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Full demonstration done in class.

**Part 2:**

|  |  |
| --- | --- |
|  |  |

Truth Tables:

Sum

| Input 1 | Input 2 | Sum Output |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Carry

| Input 1 | Input 2 | Sum Output |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Full demonstration done in class.

**Part 3:**

|  |  |
| --- | --- |
|  |  |

Sum:

| Input 1 | Input 2 | Carry Input | Sum Output |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Carry:

| Input 1 | Input 2 | Carry Input | Carry Output |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 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 | 1 |

**Part 4:**

|  |  |
| --- | --- |

4x1 MUX

|  |  |
| --- | --- |
|  |  |

**Part 5:**

|  |  |
| --- | --- |
| B: | |

**Part 6:**

|  |  |
| --- | --- |
|  |  |

**Part 7:**

|  |  |
| --- | --- |
|  |  |

## Discussion

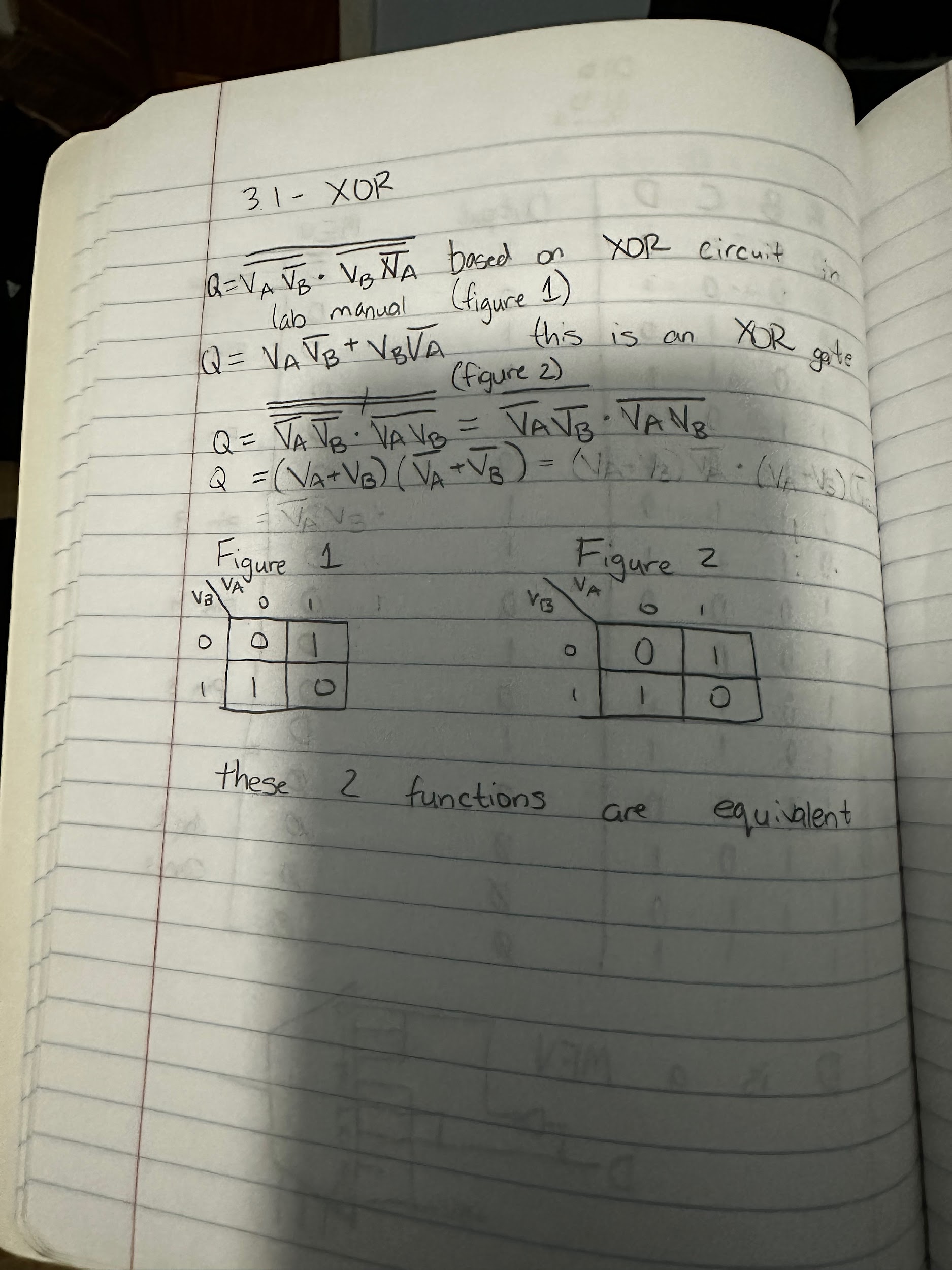
**Part 1:**

The XOR gate is one of the most foundational ICs that’s used in circuit design. It’s functionality is simple, it outputs a 1 when the two inputs coming in are different from one another. In this lab, we construct the XOR circuit in two different ways, both achieving the same logic function. The boolean expression representing the XOR function is:

And is represented by the following expression:

To implement this circuit, we use a 7404 inverter and 7300 NAND gates. Inverters are used to generate the complement of the input signals, while the NAND gates help combine and process the signals to produce the final XOR output. The connections involve wiring the NAND gates in a specific configuration to match the Boolean expression. After building the circuit, testing it with different input combinations confirms that the output matches the XOR truth table.

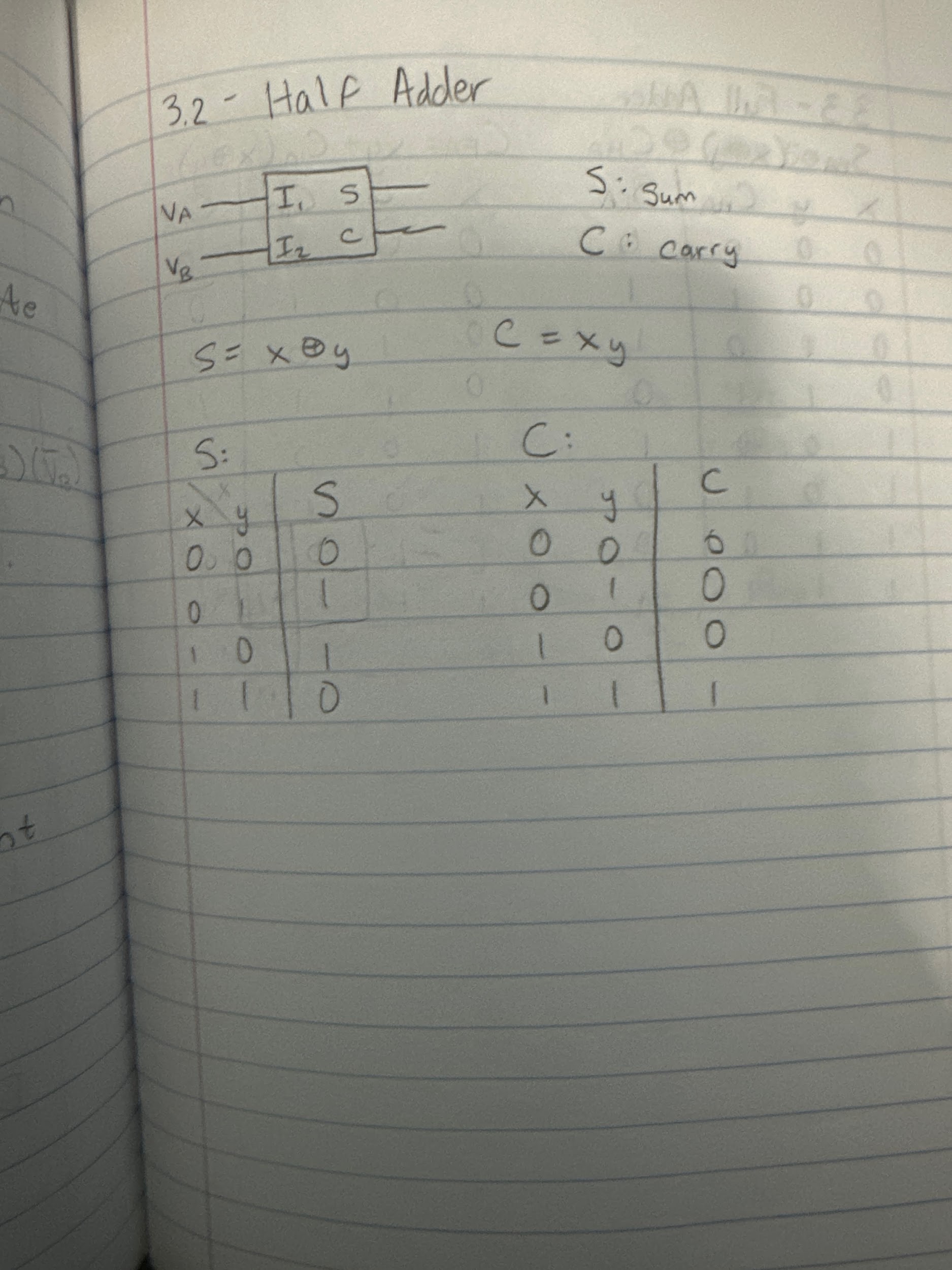
We were given two separate designs for the circuit, and we had to run separate tests. Both ended up giving the same answer, meaning that there is more than one way to implement the exclusive OR function.



**Part 2:**

A half-adder is a simple combinational circuit used to add two single-bit binary numbers. It produces two outputs, being a sum and a carry. The sum is generated by an XOR gate, while the carry is produced by an AND gate. The Boolean expressions for these outputs are:

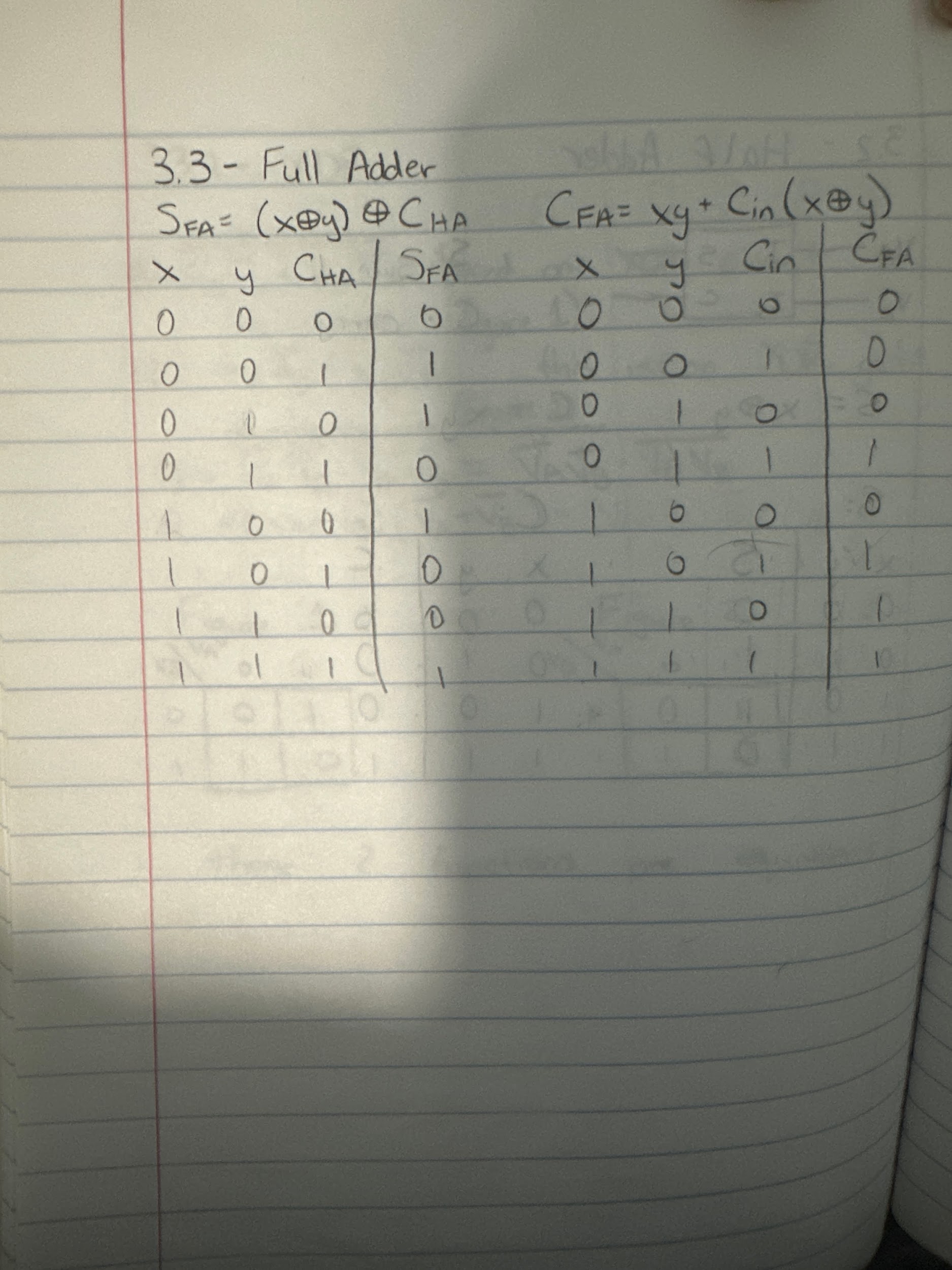
In terms of components, the XOR gate is responsible for calculating the sum, which indicates whether the total number of 1s in the inputs is odd. The AND gate determines the carry, which is high only when both inputs are 1. These gates are connected directly to the two inputs A and B. The output of the XOR gate becomes the sum, and the output of the AND gate is the carry.



**Part 3:**

The full-adder has one notable addition on top of the half-adder by introducing a carry-in input from a previous half or full adder. When we made adders that are multiple bits long, the first bit uses a half adder, and any other bits after that (whether it be one or fifteen) will use full adders because they have to take in the input of the previous carry. Full adders need to accommodate three numbers, two regular data inputs, and the carry. The boolean expressions are as follows:

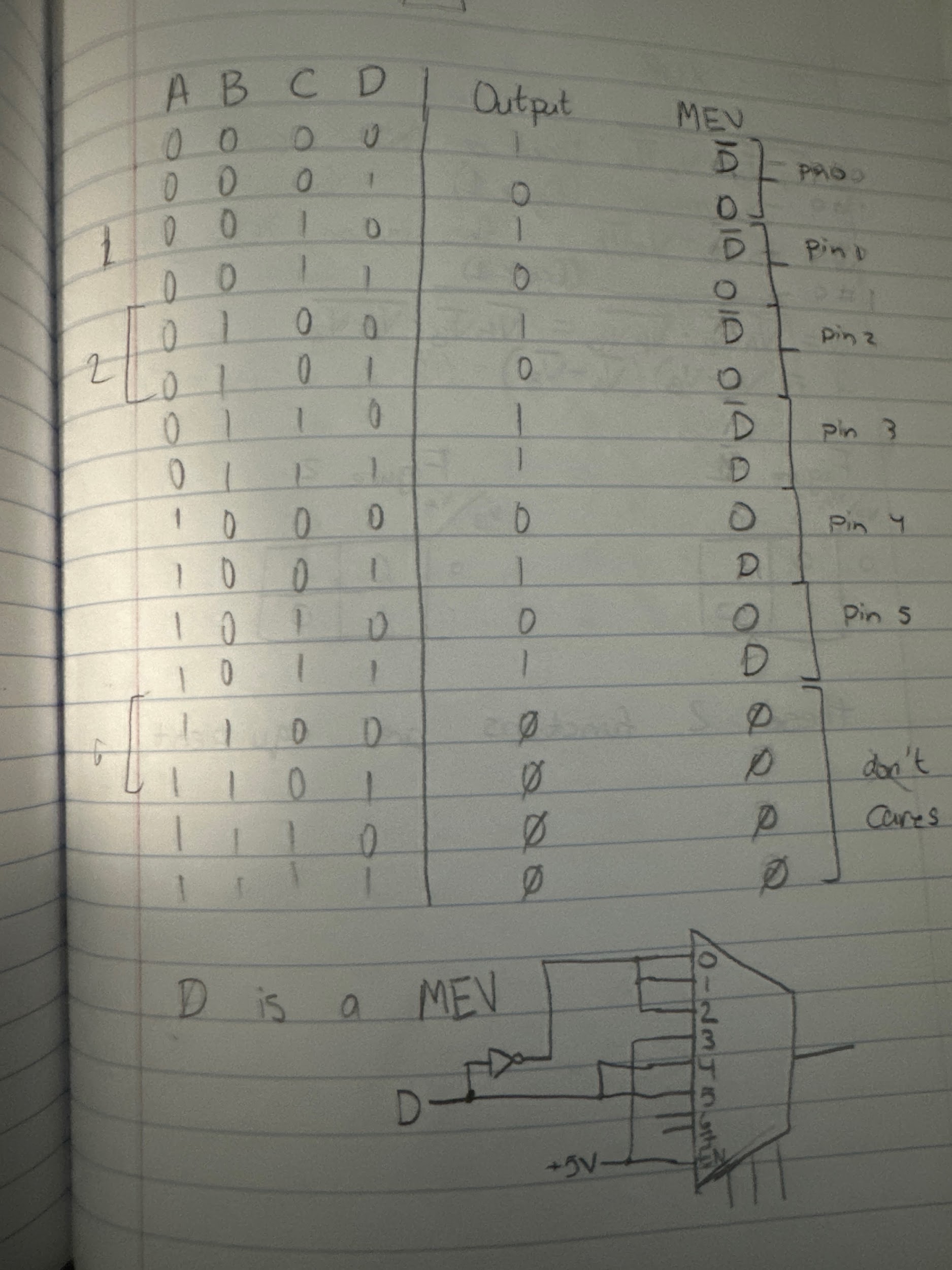
In the making of this circuit we had to use the XOR gate we designed in the first part, but since we need more than one XOR it also includes a regular 7486 TTL XOR gate.



**Part 4:**

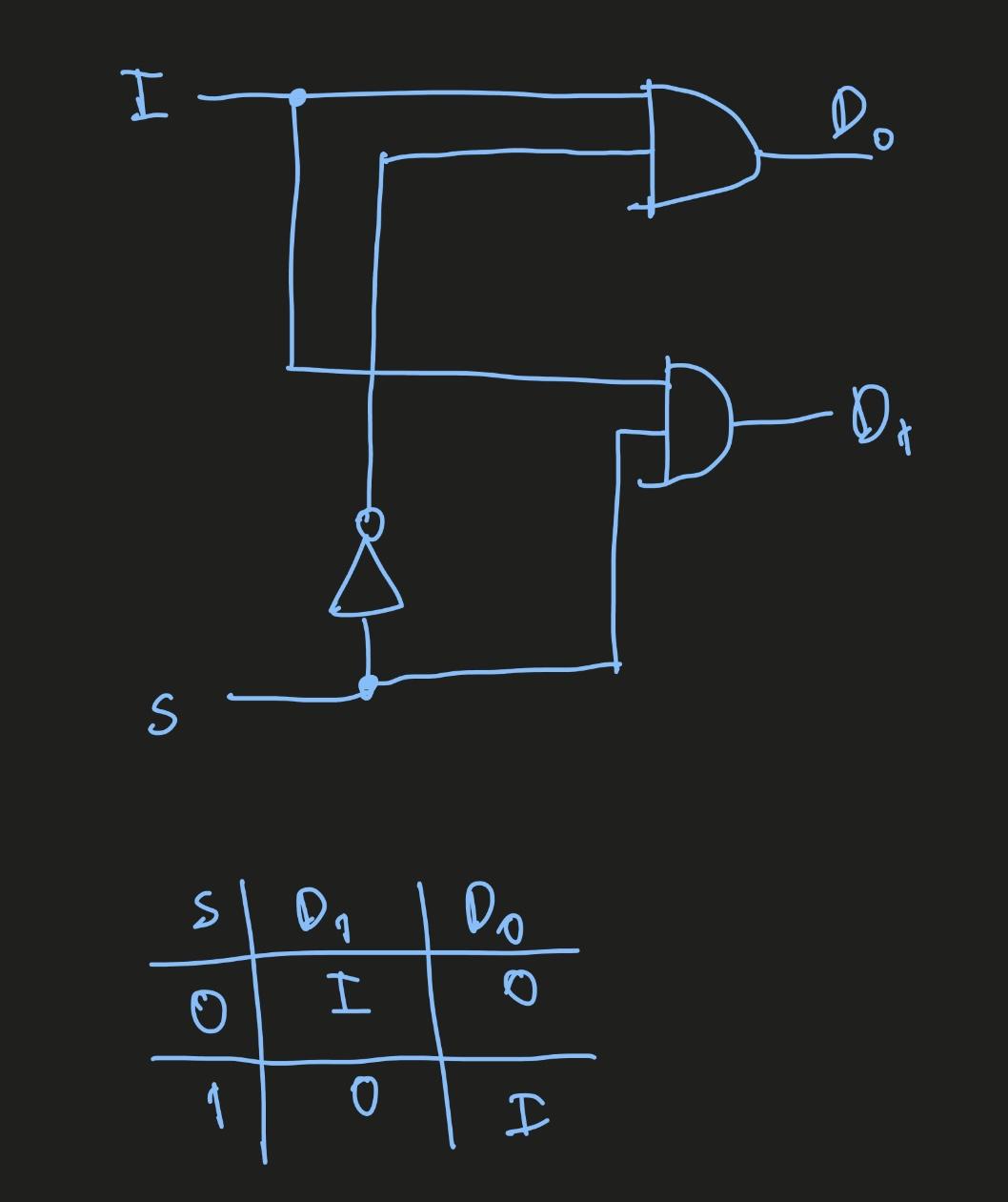
A multiplexer, shortened to a mux, is a IC which takes many inputs and depending on selector lines will send the input into the output. This multiplexer is a 4x1, which means it has 4 input lines and one output. Its address lines are 2^n = 4, so it has two address lines as well. I0, I1, I2, and I3 are the four inputs, and S1 and S2 are the selectors.

In the case of this part, we were given a specific problem of outputting a one during months where there are 31 days. The way we solved it was by setting up a truth table and solving the boolean expression for 31 days. This did require using a MEV, which we used as the LSB, labelled D. Once the truth table was made and condensed it could be made into a multiplexer.



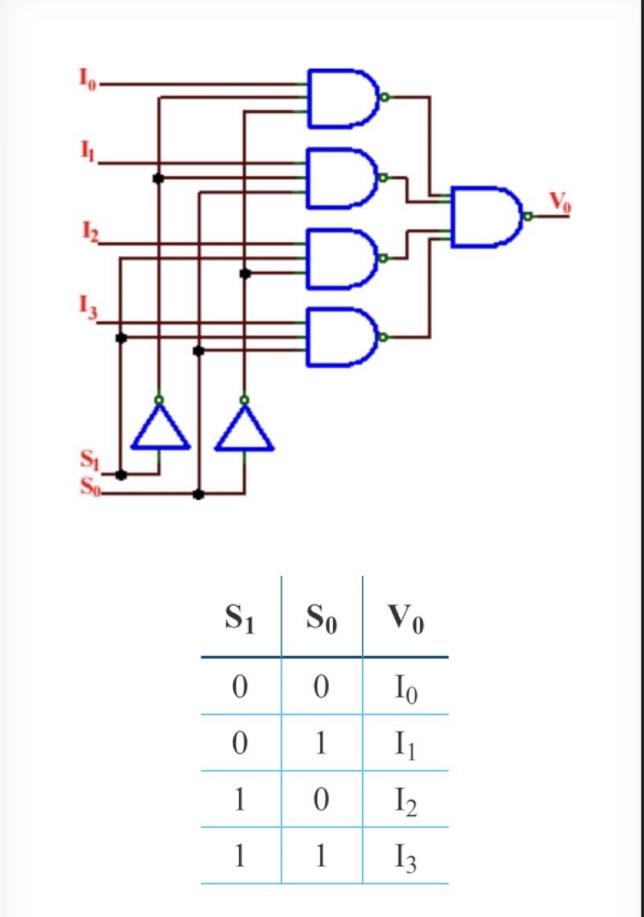
**Part 5:**

A



For this part, we created a 1-to-2 DeMUX, which took an input and a signal input and gave us an output D0 and D1. The truth table for this DeMUX is shown in the image above. The output that the DeMUX gave us was the input from an output D0 or D1, depending on the signal input.

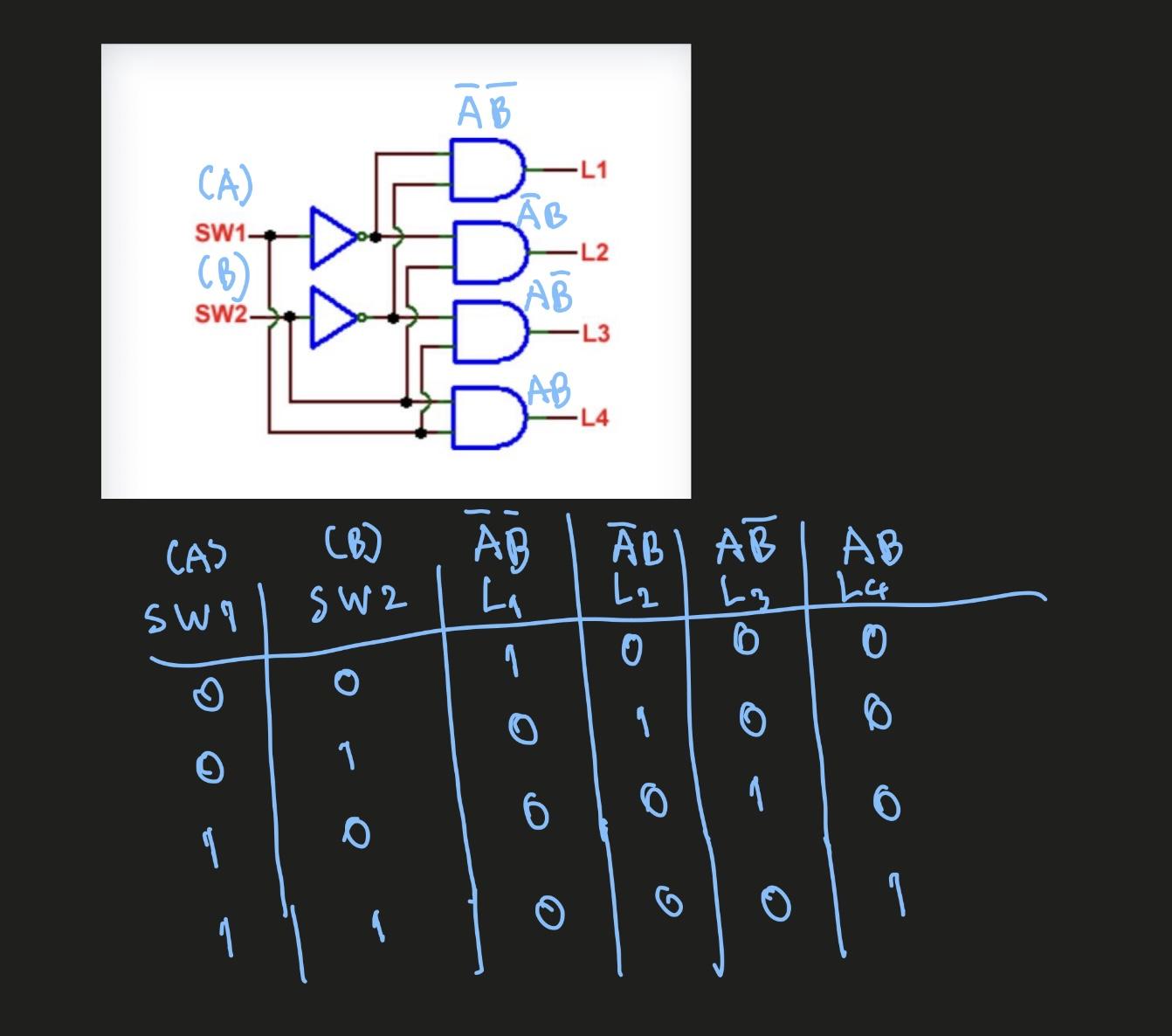
B



This is a 4-to-1 DeMUX, which takes in 4 inputs I0-I3 and 2 signal inputs to produce a single output. From top to bottom, converging into the V0 AND gate were (I0S1’S0)(I1S1’S0’)(I2S1S0)(I3S1S0’) = V0. The truth table showing the output of V0 given the S1 and S0 signal inputs are given above, and were used to select which input to get as an output.

**Part 6:**

A



In this part, we created a decoder, a device that takes in n number of inputs with 2^n possible numbers of outputs. While a MUX has 1 output with multiple inputs, a decoder is the opposite; it takes in n inputs and uses the inputs to select which 2^n possible number of outputs to output from, almost like selecting a channel given an input. The truth table for the decoder is given above.

**Part 7:**

A



In this part, we create an encoder, which performs tasks in the opposite manner of a decoder, where you put in multiple signals to one or many inputs and receive them in a code of a different format. The truth table for the encoder is given above.

## Conclusion

In this lab, we designed and made various combinational logic circuits using CMOS and TTL logic gates. We built an exclusive OR (XOR) gate, a half-adder, and a full-adder, showing how basic logic gates can be combined to create other logic gates, Mux, and so on. We worked with multiplexers (MUX), designing a circuit that outputs 1 for months with 31 days. We also used a demultiplexer (DEMUX), a decoder, and an encoder, gaining a better understanding of data and signal encoding techniques.

Through this lab, we used our theoretical knowledge of digital logic and used it with practical methods using Boolean algebra, and circuit simplification. By making and testing these circuits, we gained hands-on experience in designing efficient combinational logic systems, which will be essential for more advanced digital design applications.