

# **ECE 265**

## **Lab Report Portfolio**

University Of Illinois Chicago

Sulaiman Khan

# ECE 265 LAB 1

University Of Illinois Chicago

Sulaiman Khan

## Purpose:

The purpose of this lab is to convert a Boolean expression into a real circuit that performs the same operation. This is so that we are familiarized with implementing logic into real applications, and so that we gain experience using online simulations like CircuitVerse, and gain experience with physical hardware using the breadboard

## Procedure:

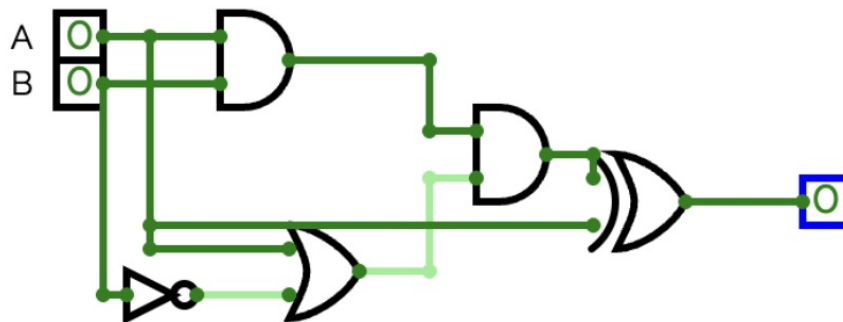
The given expression was  $x = ((ab)(a + b'))a$ ,

First, I built a CircuitVerse simulation of the Boolean expression using two inputs, a and b. Then I created a truth table for both the expression and the simulated circuit to verify that the outputs matched and that the design was correct.

After that, I created a wiring diagram using the provided breadboard template to plan the hardware layout before assembling anything.

Finally, I constructed the physical circuit using the 74LS08 (AND), 74LS32 (OR), 74LS04 (NOT), and 74LS86 (XOR) ICs, along with pushbuttons, resistors, and an LED as the output indicator.

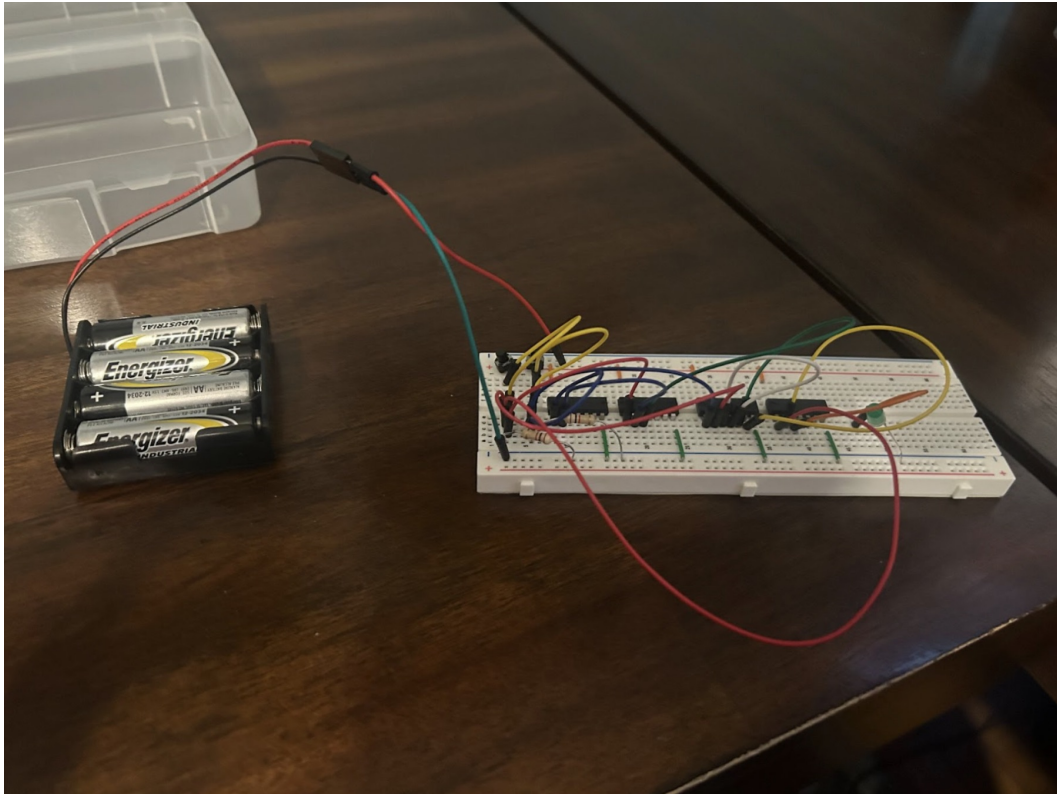
## CircuitVerse Simulation:



### Truth Table:

a	b	x
0	0	0
0	1	0
1	0	1
1	1	0

### Physical Circuit:



### Conclusion:

In this lab, I learned how to convert a Boolean expression into both a simulated circuit and real hardware. I used circuit verse to verify that I was on the right track, then used the wiring diagram to help prepare before building the physical circuit. I gained hands-on experience using breadboards and also learned a great deal about how to utilize logic gates. Overall, I became more comfortable in actually applying the theory that we learned in class

# ECE 265 LAB 2

University Of Illinois Chicago

Sulaiman Khan

## Purpose:

The purpose of this experiment is to take a given logic function and reverse-engineer a circuit that reproduces its behavior. This lab helped us practice translating an unknown operation into our own hardware implementation, while also building hands-on experience with both simulated logic and physical components.

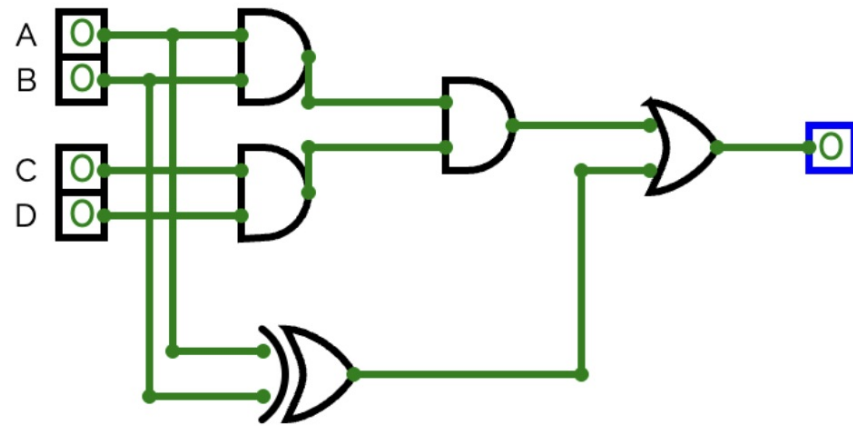
## Procedure:

First, I analyzed the given circuit by testing every possible input combination and recording the output in a truth table. Using that truth table, I wrote the sum-of-minterms expression for the output  $x$  and simplified it into a Boolean expression. I then simulated the simplified expression in CircuitVerse and verified it by generating a second truth table. Once both truth tables matched, I constructed the physical circuit on a breadboard using the appropriate ICs, four pushbuttons for input, and an LED for output. Finally, I confirmed that the hardware outputs matched the CircuitVerse simulation.

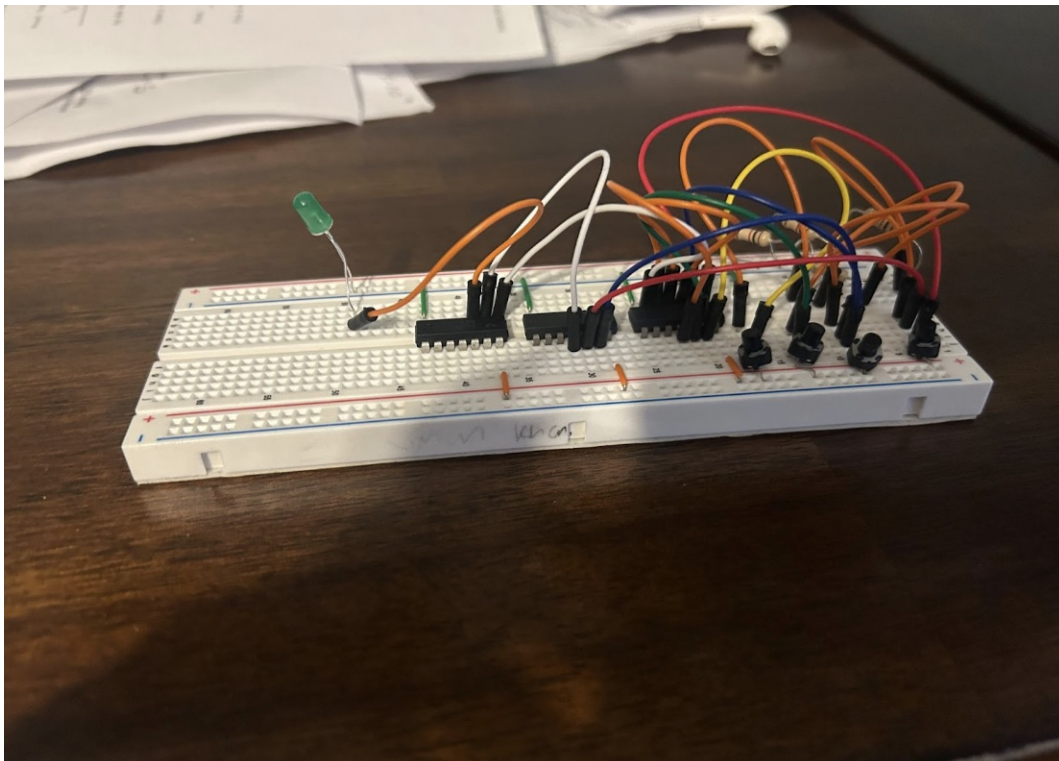
## Truth Table:

A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

### CircuitVerse Simulation:



### Physical Circuit:



**Sum-of-minterm expression for the function  $x(a, b, c, d)$ :**

$$x(a, b, c, d) = A'B + AB' + ABCD$$

**Steps of Boolean algebra used to simplify the expression for  $x$  in order to try to minimize the number of literals:**

We start with  $x = A'BC'D' + A'BC'D + A'BCD' + A'BCD + AB'C'D' + AB'C'D + AB'CD' + AB'CD + ABCD$

We can use the distributive and complimentary properties to simplify this to  $A'BC' + A'BC + AB'C' + AB'C + ABCD$

We can then combine terms like  $A'BC' + A'BC = A'B(C' + C) = A'B$  resulting in  $A'B + AB' + ABCD$

**Conclusion:**

Through this experiment I learned how to analyze unknown logic and determine its function so that I can reverse engineer it in an efficient manner. I also learned a lot about how to turn logical functions into circuit verse simulations as well as gaining hands-on experience using physical hardware.

# ECE 265 LAB 3

University Of Illinois Chicago

Sulaiman Khan

## Purpose:

The purpose of this lab was to design and implement digital logic circuits using a 3x8 decoder to realize the given function. By using the decoder along with AND, OR, NAND, and NOR gates, I was able to determine how expressions can be identified and simplified using different logic gates.

## Procedure:

First, I created a truth table for the outputs x, y, and z, listing all eight possible input combinations and their corresponding values.

Then, I expressed each output both as a sum of minterms and as a product of maxterms.

Next, I built four digital circuits in CircuitVerse using a 3x8 decoder — two versions with active-high outputs and two with active-low outputs. After confirming that all four versions produced the same outputs, I combined the logic to design an optimized final circuit and implemented it on the breadboard.

## Truth Table:

a	b	c	x	y	z
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	0

## Sum of Minterms:

$$x(a,b,c): x = ab'c + abc' + abc$$

$$y(a,b,c): y = a'bc' + a'bc + ab'c' + ab'c + abc' + abc$$

$$z(a,b,c): z = ab'c'$$

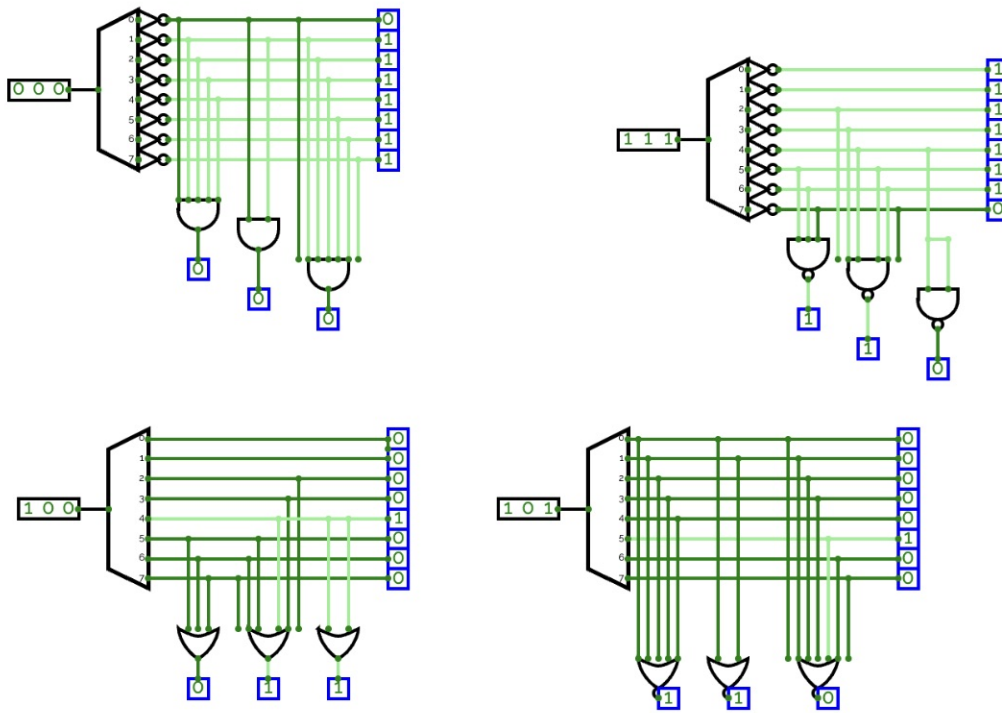
## Product of Maxterms:

$$x(a,b,c): x = (a + b + c)(a + b + c')(a + b' + c)(a + b' + c')(a' + b + c)$$

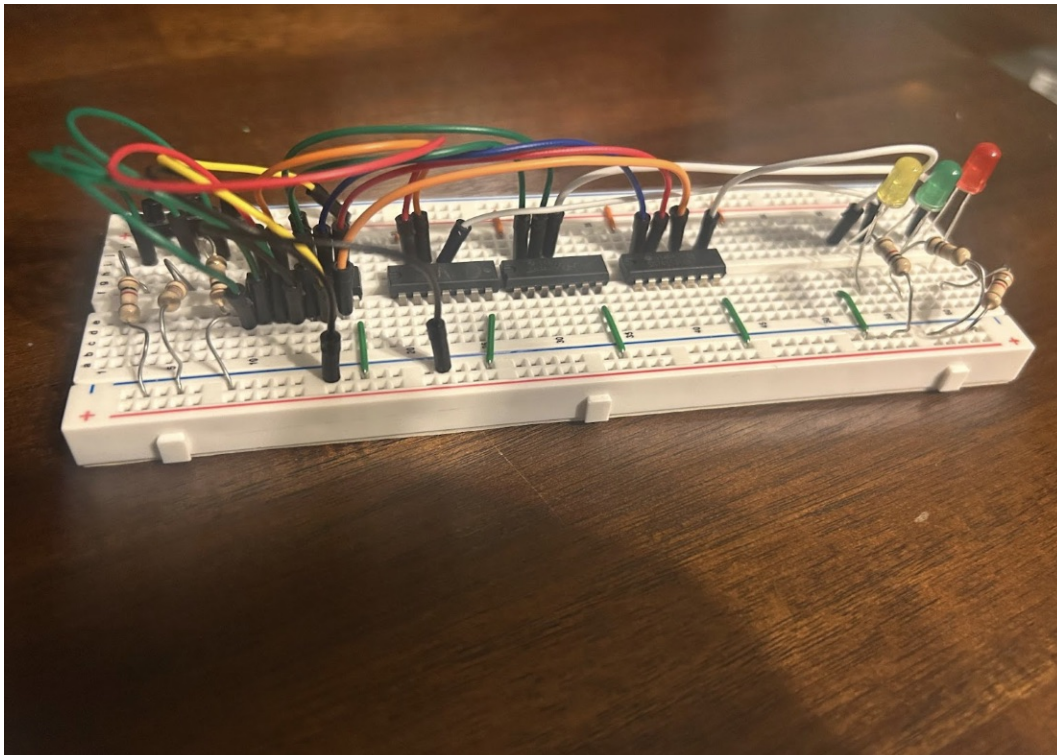
$$y(a,b,c): y = (a + b + c)(a + b + c')$$

$$z(a,b,c): z = (a + b + c)(a + b + c')(a + b' + c)(a + b' + c')(a' + b + c')(a' + b' + c)(a' + b' + c')$$

## CircuitVerse Diagrams:



## Physical Circuit:



**Conclusion:**

Through this experiment, I learned how to translate a functional description into a truth table and then derive Boolean expressions using minterms and maxterms. I also learned how to minimize those expressions to create a more efficient circuit. Working with a  $3 \times 8$  decoder in both CircuitVerse and on the breadboard helped me understand how decoders operate and strengthened the connection between theory and practical hardware implementation.

# ECE 265 LAB 4

University Of Illinois Chicago

Sulaiman Khan

## Purpose:

The purpose of this experiment is to strengthen our understanding of binary arithmetic by applying techniques we learned in class and applying the theory to real-life situations. We apply a shift, addition, and 2's complement to create a circuit that computes  $Z = 2Y - 1$  from a 3-bit input, demonstrating how mathematical operations can be performed in hardware.

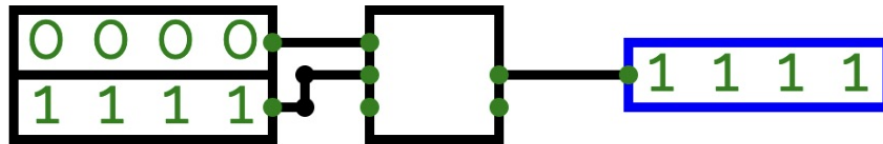
## Procedure:

First, I made a truth table for  $Z = 2Y - 1$  using 3-bit input values, then I designed the circuit in circuit verse using a 4-bit adder. I used a left shift to double Y and then added the 2's complement of 1 to simulate subtraction. Then, I built the same circuit on a breadboard using a 74LS283, using buttons for inputs and LEDs for outputs. Finally, I tested all inputs and confirmed that they matched my original truth table.

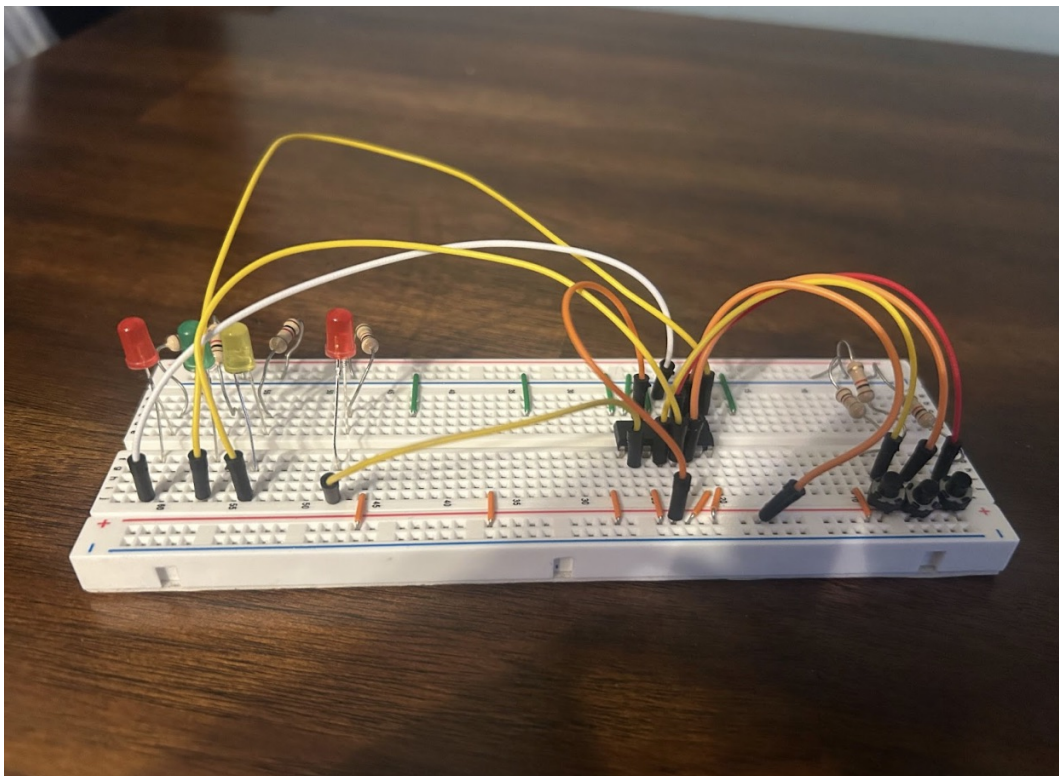
## Truth Table:

a	b	c	d	e	f	g
0	0	0	x	x	x	x
0	0	1	0	0	0	1
0	1	0	0	0	1	1
0	1	1	0	1	0	1
1	0	0	0	1	1	1
1	0	1	1	0	0	1
1	1	0	1	0	1	1
1	1	1	1	1	0	1

Circuit Verse:



Circuit:



**Conclusion:**

Through this experiment, I learned how mathematical operations such as multiplication and subtraction can be done using hardware. I designed a circuit that computes  $Z = 2Y = 1$ , giving me a good understanding of how binary shifting is used for multiplication and how the 2's complement is used for subtraction. I also strengthened my skills in creating physical circuits using truth tables and simulations, which helped me understand the relation between the theory learned in class and actual implementation

# ECE 265 LAB 5

University Of Illinois Chicago

Sulaiman Khan

## Purpose:

The goal of this lab is to use K maps to find digital logic functions that calculate the 9's complement of a BCD codeword. We will simplify the expressions using dont care conditions, then build and test them in a simulator with 2 input logic gates

## Procedure:

We begin by constructing a complete truth table for the 9's-complementer, assigning all invalid BCD inputs the output 0000. We then use Karnaugh maps for each output bit and simplify the logic to obtain the minimal sum-of-products expressions.

After that, we create a second truth table in which all invalid states are assigned 1111, and we again use K-maps to reduce each output to its simplest SOP form.

Next, we prepare a third truth table where every invalid state is written as XXXX to indicate don't care conditions. We replace each X with either 0 or 1 in whatever way produces the greatest simplification in the K-maps.

Finally, using the minimized expressions from these steps, we construct the completed circuit in CircuitVerse, implementing it exclusively with 2-input logic gates and inverters.

## Truth Table (All invalid = 0000):

$a_3$	$a_2$	$a_1$	$a_0$	$b_3$	$b_2$	$b_1$	$b_0$
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0

**K-maps all invalid = 0000:**

B0:  $a'd' + b'c'd'$

B1:  $a'c$

B2:  $a'bc' + a'b'c$

B3:  $a'b'c'$

**Truth Table (All invalid = 1111):**

$a_3$	$a_2$	$a_1$	$a_0$	$b_3$	$b_2$	$b_1$	$b_0$
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	1
1	1	0	0	1	1	1	1
1	1	0	1	1	1	1	1
1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1

**K-maps all invalid = 1111:**

B0:  $ab + cd' + c'd'$

B1:  $ab + c$

B2:  $ac + bc' + b'c$

B3:  $ac + ab + a'b'c'$

Truth Table (All invalid = XXXX):

$a_3$	$a_2$	$a_1$	$a_0$	$b_3$	$b_2$	$b_1$	$b_0$
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

K-maps for all invalid = XXXX, optimized as well for the simplest SOP:

B0:  $d'$

B1:  $c$

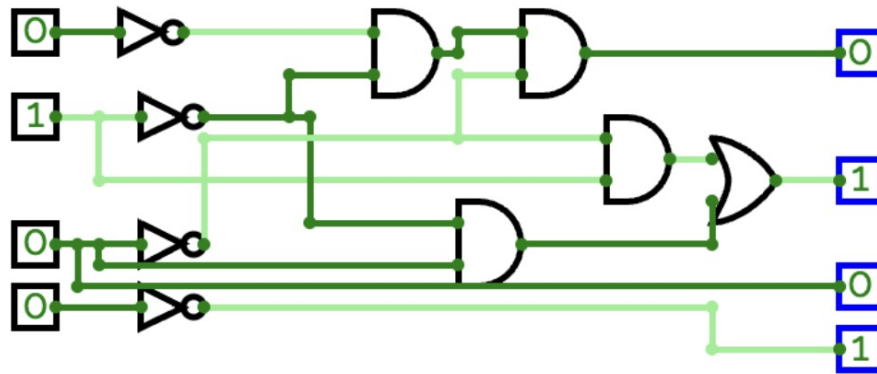
B2:  $bc' + cb'$

B3:  $a'b'c'$

Optimized Truth Table:

$a_3$	$a_2$	$a_1$	$a_0$	$b_3$	$b_2$	$b_1$	$b_0$
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	0	1	1	1
1	0	1	1	0	1	1	0
1	1	0	0	0	1	0	1
1	1	0	1	0	0	1	0
1	1	1	0	0	0	1	1
1	1	1	1	1	1	0	0

Circuitverse:



### Conclusion:

In this lab, I learned how to take a BCD 9's complement and turn it into a circuit using logic gates. First, I started with truth tables and used them to create K-maps. I optimized the dont cares to produce the simplest SOP and then implemented it all into CircuitVerse.

# ECE 265 LAB 6

University Of Illinois Chicago

Sulaiman Khan

## Purpose:

The purpose of this experiment is to design and implement a traffic signal controller using a 4-bit binary counter and clock circuit, and using AND and NAND gates to create the correct output. We will learn how logical components can be implemented to create real world technology.

## Procedure:

We begin by taking the truth table and using Karnaugh maps to derive the simplest logical expressions for each output. After obtaining the minimized forms, we implement the expressions in CircuitVerse and verify that each output matches the expected behavior. Once the simulated circuit is confirmed to be correct, we proceed to build the design in hardware.

## Truth Table:

N	R1	Y1	G1	R2	Y2	G2
0	0	0	1	1	0	0
1	0	0	1	1	0	0
2	0	0	1	1	0	0
3	0	0	1	1	0	0
4	0	0	1	1	0	0
5	0	0	1	1	0	0
6	0	1	0	1	0	0
7	0	1	0	1	0	0
8	1	0	0	0	0	1
9	1	0	0	0	0	1
10	1	0	0	0	0	1
11	1	0	0	0	0	1
12	1	0	0	0	0	1
13	1	0	0	0	0	1
14	1	0	0	0	1	0
15	1	0	0	0	1	0

### Simplest SOP:

$$R1 = A$$

$$Y1 = A'BC$$

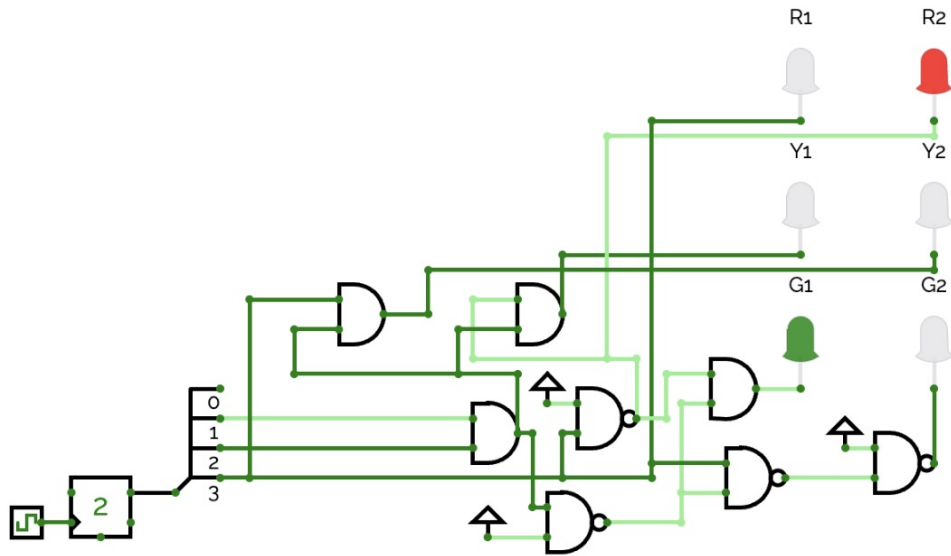
$$G1 = A'C' + A'B'$$

$$R2 = A'$$

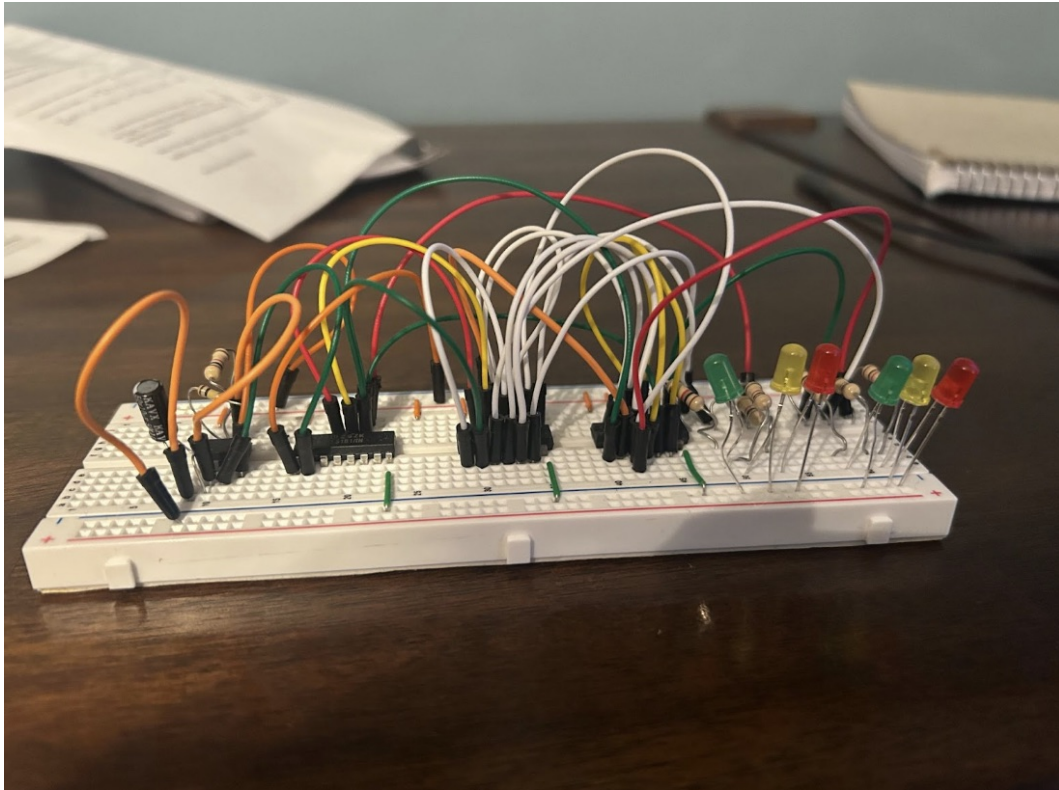
$$Y2 = ABC$$

$$G2 = AB' + AC'$$

### CircuitVerse:



## Breadboard



### Conclusion:

Through this experiment, we developed a traffic signal controller using a binary counter, a clock, and basic logic gates. Working through the design and implementation strengthened our understanding of how the concepts introduced in class translate into real hardware behavior. The process showed that even simple digital components, when combined thoughtfully, can form practical systems similar to those used in real-world engineering applications experienced by millions of people.

# ECE 265 LAB 7

University Of Illinois Chicago

Sulaiman Khan

## Purpose:

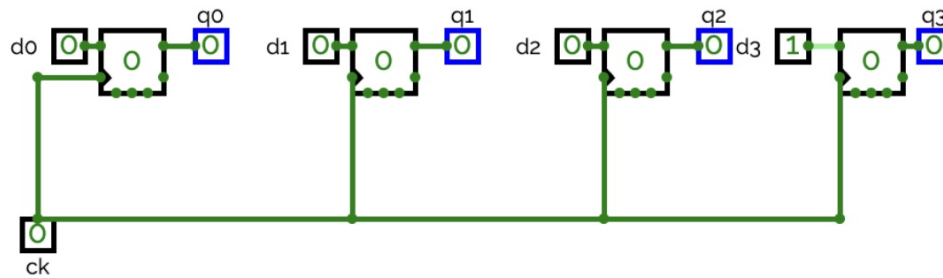
The purpose of this lab is to design and implement a 4-bit increment counter using D flip-flops and a 4-bit adder. The exporometer demonstrates how a finite state machine can be utilized to create a counter that increments or decrements by a specified value. By varying the input N, we can make it function as a counter that increments, decrements, or skips N numbers.

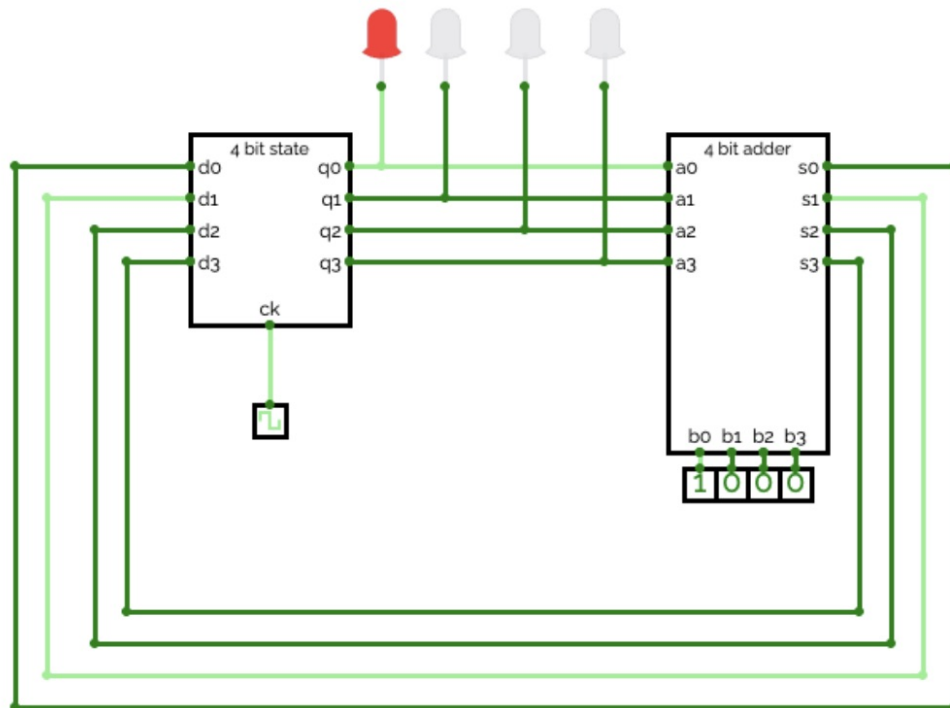
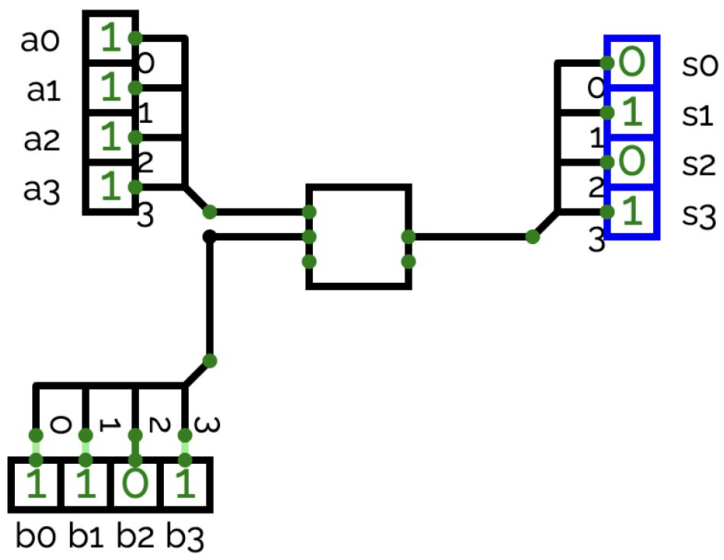
## Procedure:

I built the circuit in CircuitVerse using four D flip-flops and a 4-bit adder. The outputs of the flip-flops went into one side of the adder, and {b3b2b1b0} set the value of N. The adder's output looped back into the D inputs to update the state each clock cycle.

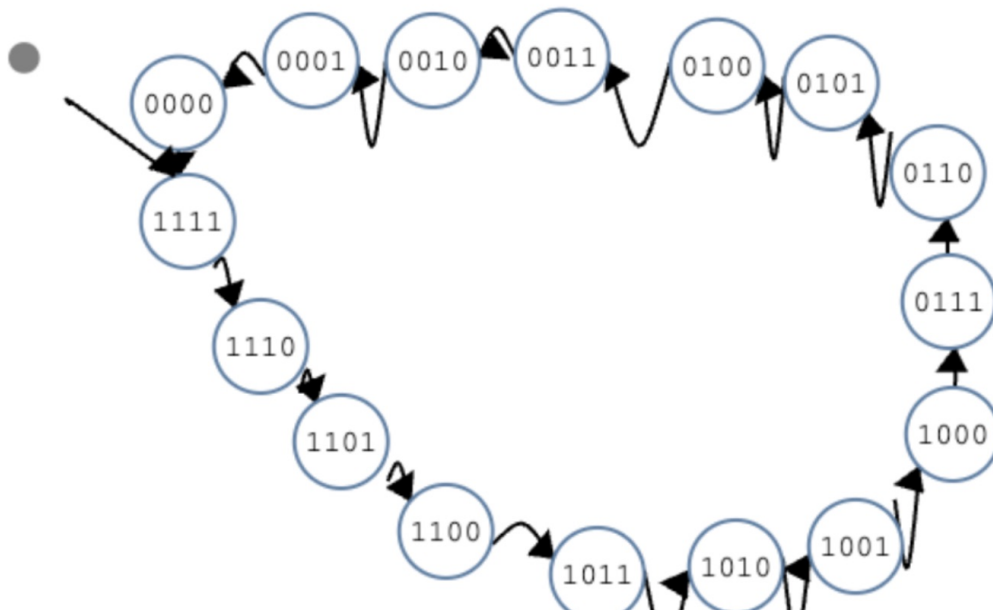
I first set  $N = 0001$  to create an up counter, then 1111 for a down counter, and finally 0011 for an increment by 3 counter. After testing all three in simulation, I built the regular up counter on a breadboard using a 555 timer for the clock and LEDs for output.

## CircuitVerse:





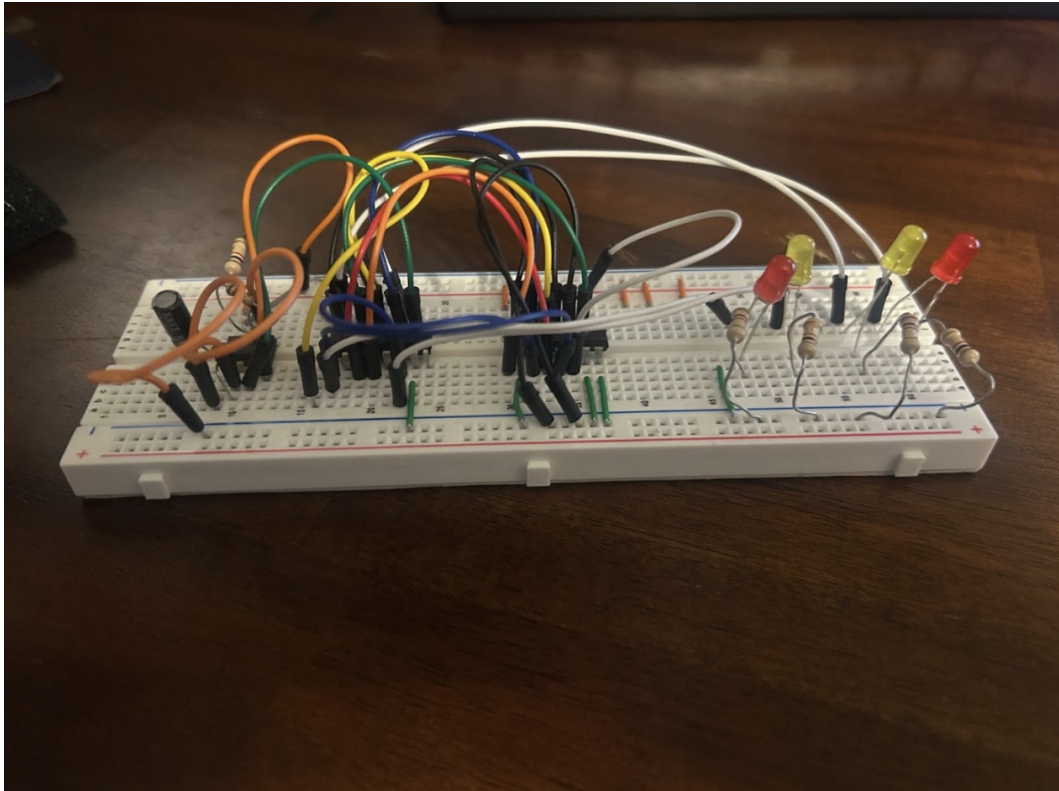
State transition diagram (b3b2b1b0 = 1111):



State transition table from (c) (b3b2b1b0 = 0011):

Present State	Next State
0000	0011
0001	0100
0010	0101
0011	0110
0100	0111
0101	1000
0110	1001
0111	1010
1000	1011
1001	1100
1010	1101
1011	1110
1100	1111
1101	0000
1110	0001
1111	0010

### Physical Circuit:



### Conclusion:

This lab demonstrated how a 4-bit counter can be constructed using D flip-flops and a 4-bit adder to control the increment or decrement of the counter with each clock tick. By changing  $\{b_3b_2b_1b_0\}$ , the same circuit could act as an up counter, down counter, or increment by 3 counter. Through both simulation and hardware testing, I learned how sequential and combinational logic work together to create state transitions.

# ECE 265 LAB 8

University Of Illinois Chicago

Sulaiman Khan

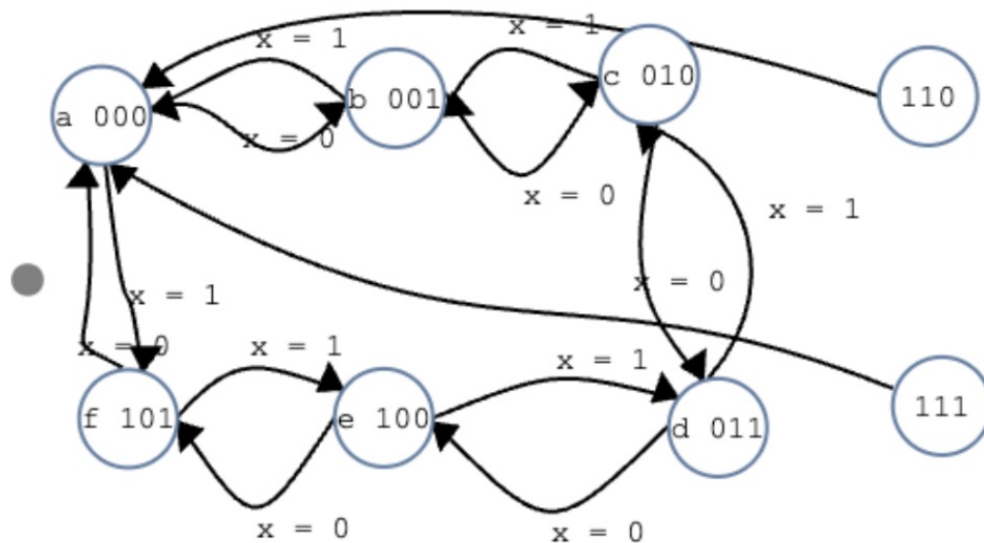
## Purpose:

The purpose of this lab is to design a finite state machine that controls a 7-segment display and makes it show a clockwise or counterclockwise movement based on an input.

## Procedure:

We began by assigning binary codes to the six states and determining how the FSM should transition in both directions based on the value of  $x$ . Using this information, we drew the state diagram and converted it into a complete state-transition table. From that table, we derived the sum-of-minterms expressions for the next-state bits and implemented the steering logic in CircuitVerse. Afterward, we created the output table for the display segments, obtained the sum-of-minterms expressions for each output, and built the output logic. Finally, we connected all components in the presentation circuit and tested the system to confirm that the display rotated correctly.

## State Transition Diagram:



### State Transition Table:

x	Q1	Q2	Q3	D1	D2	D3
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	1	1
0	0	1	1	1	0	0
0	1	0	0	1	0	1
0	1	0	1	0	0	0
1	0	0	0	1	0	1
1	0	0	1	0	0	0
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	0	1	1
1	1	0	1	1	0	0

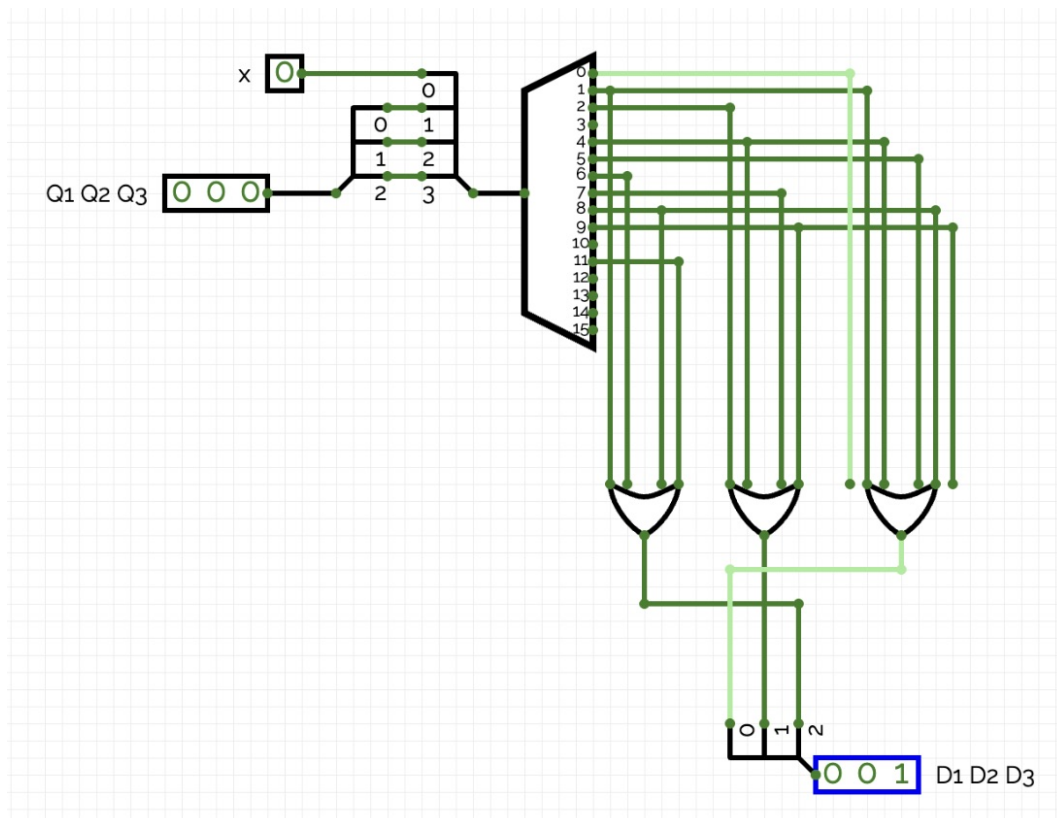
### Steering logic SOP expressions:

$$D1\{x, Q1, Q2, Q3\} = \text{sum}(3, 4, 8, 13)$$

$$D2\{x, Q1, Q2, Q3\} = \text{sum}(1, 2, 11, 12)$$

$$D3\{x, Q1, Q2, Q3\} = \text{sum}(0, 2, 4, 8, 10, 12)$$

### Steering Logic Circuit:



Output logic truth table:

Q1	Q2	Q3	a	b	c	d	e	f
0	0	0	1	0	0	0	0	0
0	0	1	0	1	0	0	0	0
0	1	0	0	0	1	0	0	0
0	1	1	0	0	0	1	0	0
1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0

Output logic SOP expressions:

$$a = Q1'Q2'Q3'$$

$$b = Q1'Q2'Q3$$

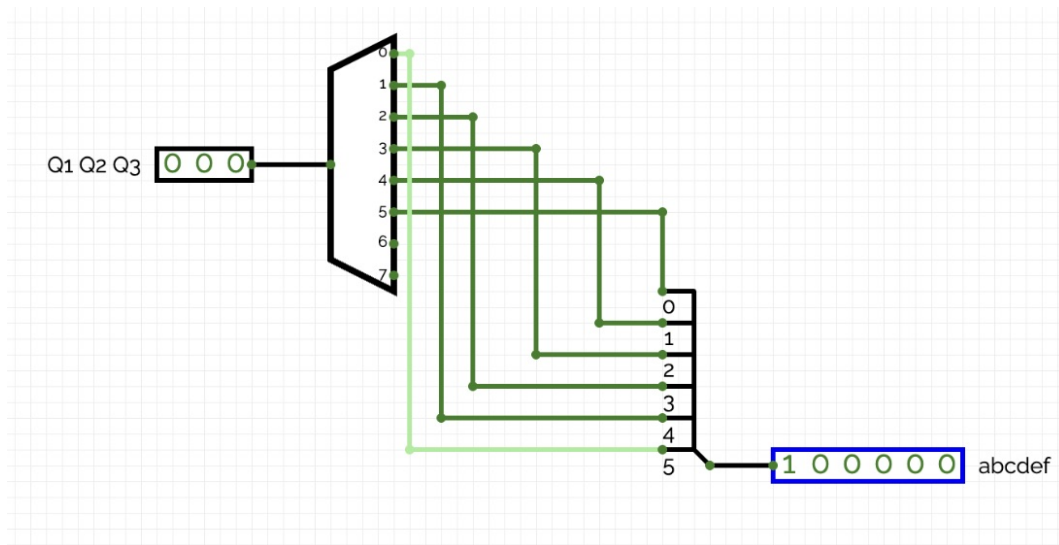
$$c = Q1'Q2Q3'$$

$$d = Q1'Q2Q3$$

$$e = Q1Q2'Q3'$$

$$f = Q1Q2'Q3$$

Output Logic Circuit:



Conclusion:

This Lab helped me gain a stronger understanding of how state diagrams can be used to create useful circuits. I was able to follow a step-by-step process to create a fairly complex circuit just by beginning with a state diagram. This helps me create a better connection between the theory I learn in class and real-life applications.

# ECE 265 LAB 9

University Of Illinois Chicago

Sulaiman Khan

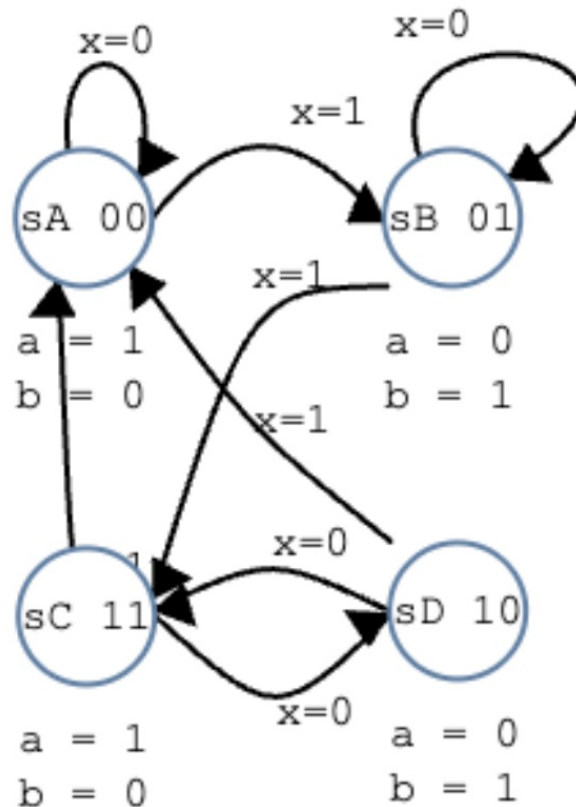
## Purpose:

The purpose of this lab was to bring all that we learned together and create a Mealy FSM. This experiment will help us learn how to use state transition diagrams, transition tables, minimal terms, K-maps, and other techniques to design a real circuit.

## Procedure:

First, we need to design a state transition diagram that shows how the circuit transitions. Then, we need to assign state variables and state transitions along with their corresponding outputs and inputs. Next, we will create a state transition table that presents the logic of the state transition diagram in a comprehensible manner. Next, we will find the simplified expressions for D1 and D2 in terms of the inputs Q1 Q1 and x. Finally, we will create the circuit in circuit verse and then implement it onto our physical breadboard.

## State Transition Diagram:



## State transition table

Q1	Q2	x	D1	D2	Y	N
0	0	1	0	1	0	0
0	0	0	0	0	1	0
0	1	1	1	1	0	0
0	1	0	0	1	0	1
1	0	1	0	0	0	0
1	0	0	1	1	1	0
1	1	1	0	0	0	0
1	1	0	1	0	0	1

## Expressions

$$D1: Q1'Q2x + Q1x'$$

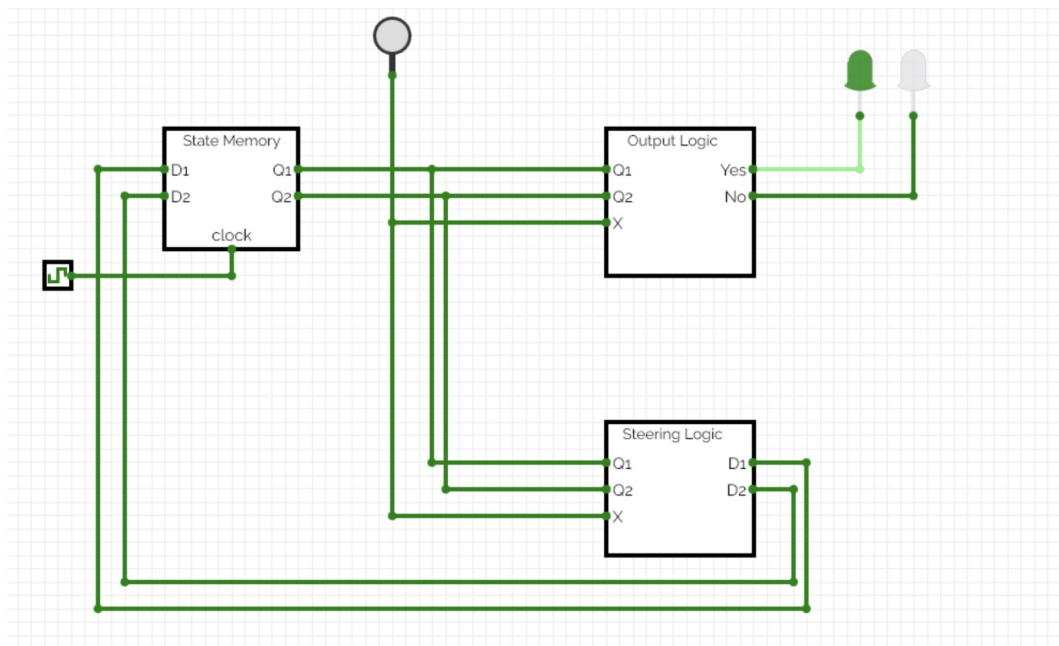
$$D2: Q1'Q2'x + Q1'Q2 + Q1Q2'x'$$

$$Y = Q2'x'$$

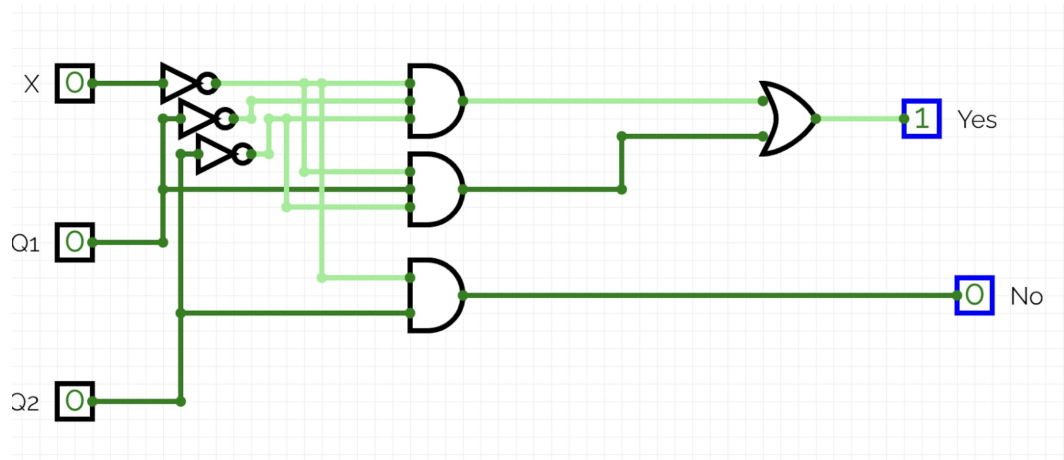
$$N = Q2x'$$

## Circuit Verse

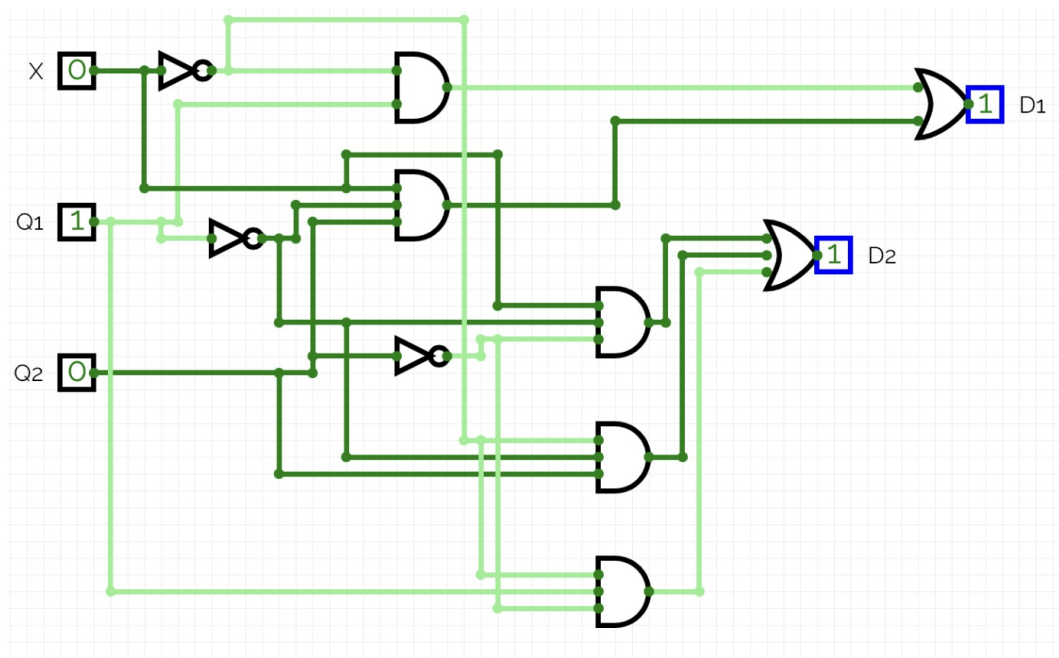
Main



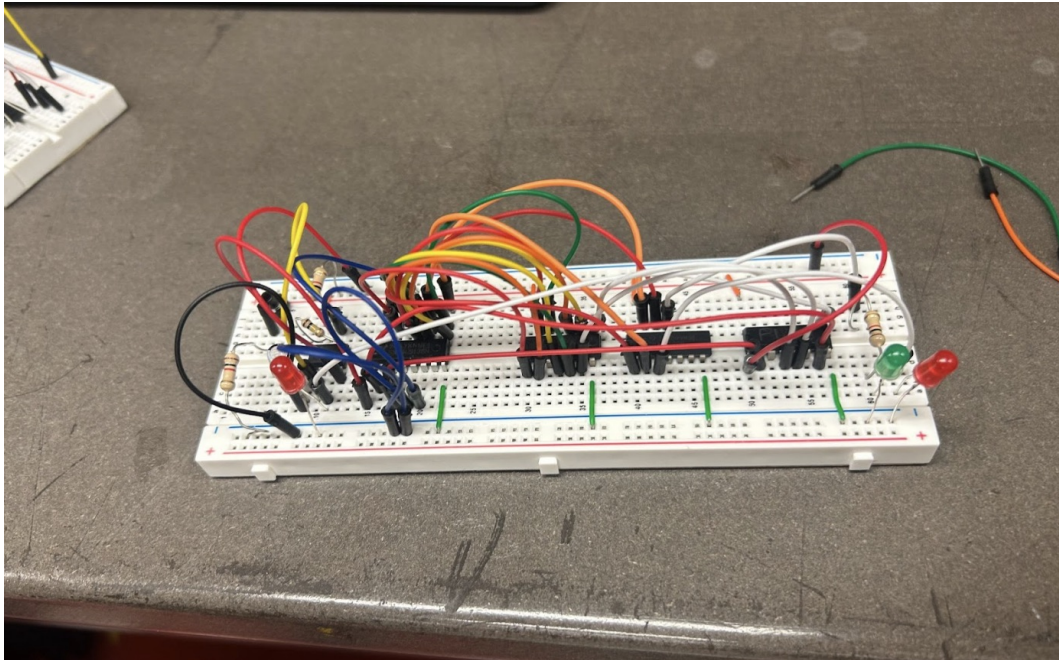
Output:



Steering:



## Breadboard:



## Conclusion

In this lab, I learned how to follow the engineering design process by first identifying the necessary techniques and then combining them to develop a solution. It helped me understand how to use state transition diagrams, state transition tables, simplified expressions, and different types of logical ICs to create an FSM.