# Digital Logic Practical Lab Report

## Multiplexer and Decoder/Demultiplexer

Student:

Abhiroop Mukherjee

Teacher:

Prof. Surajeet Ghosh

Prof. Sekhar Mandal

Course:

Digital Logic Laboratory (CS351)

# EXPERIMENT 1 (a)

## Objective

Construct a 2-to-1 Multiplexer using basic logic gates and verify the truth table.

## Theory

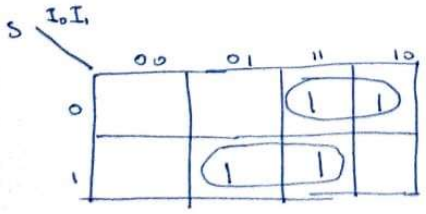
→ A 2-to-1 Multiplexer consists of two inputs I0 and I1, one select line S and one output Y. Depending of S, the output Y is connected to either of the inputs.

→ If S = 0, output will be switched to I0, if S = 1, output will be switched to I1

→ It's Truth Table is as follows:

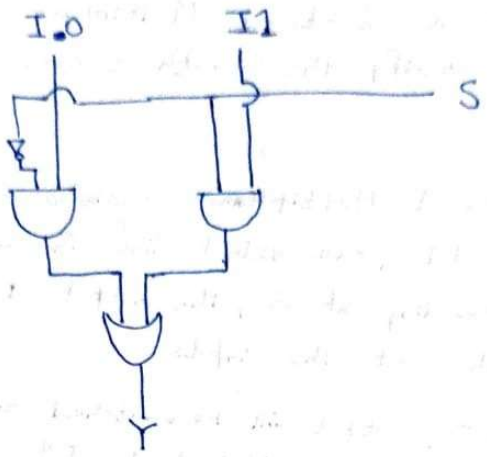
| S | I0 | I1 | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

→ from the truth table.




$$Y = I_0 \bar{S} + I_1 S$$

→ Hence the Circuit Diagram




Result

As we see from the simulation, we require 1 NOT, 2 AND and 1 OR gates to make 2-to-1 multiplexer. Moreover the simulation result match with truth table

## Inputs / Outputs:

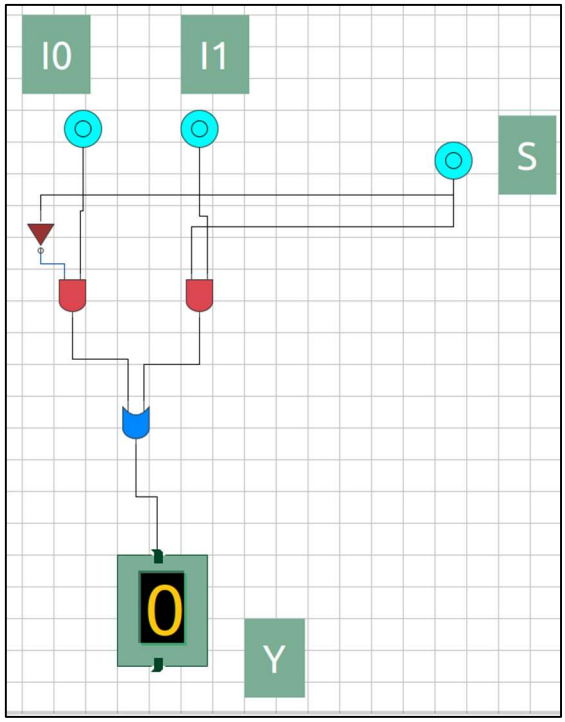



Figure 1: S = 0, I0 = 0, I1 = 0; Y=0

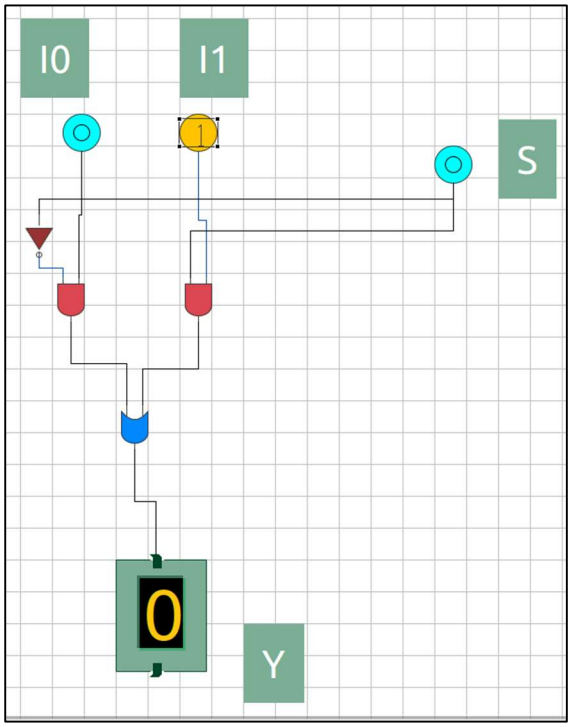



Figure 2: S = 0, I0 = 0, I1 = 1; Y=0

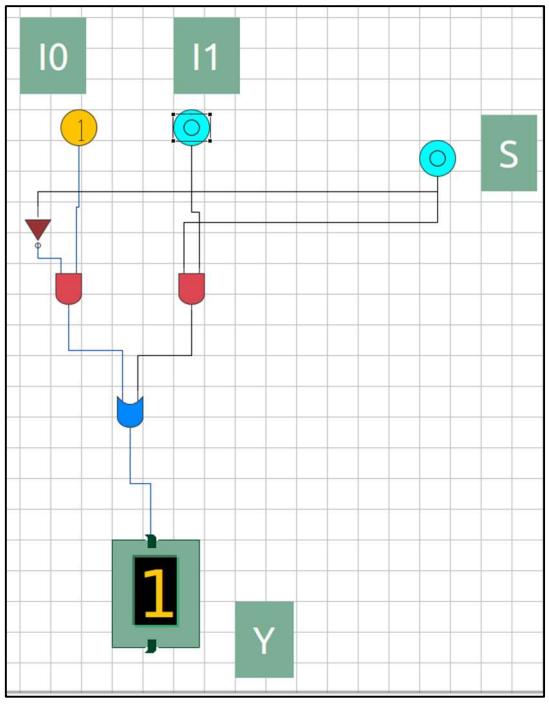



Figure 3: S = 0, I0 = 1, I1 = 0; Y=1

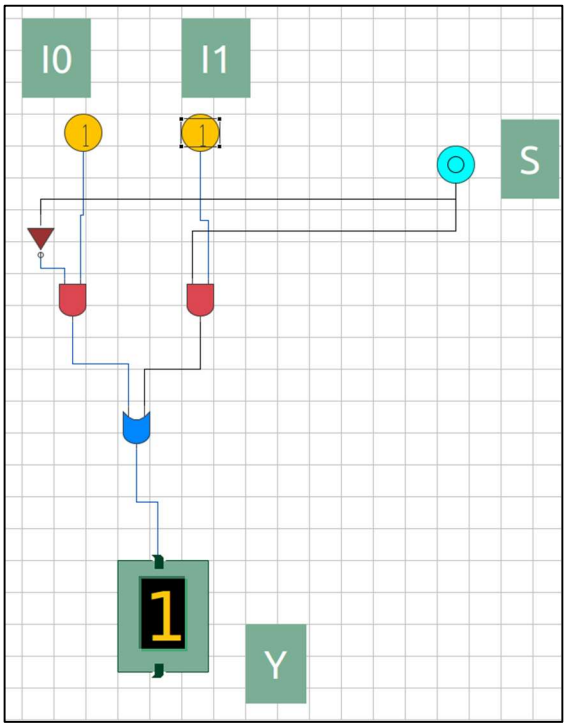



Figure 4: S = 0, I0 = 1, I1 = 1; Y=1

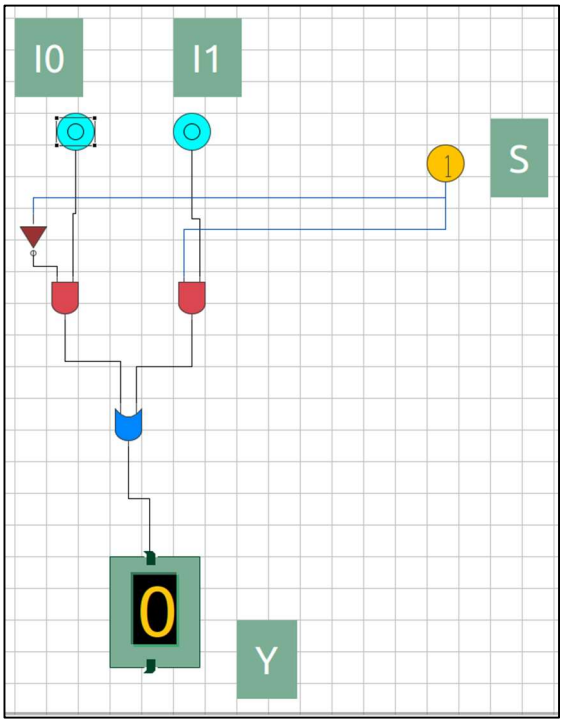



Figure 5: S = 1, I0 = 0, I1 = 0; Y=0

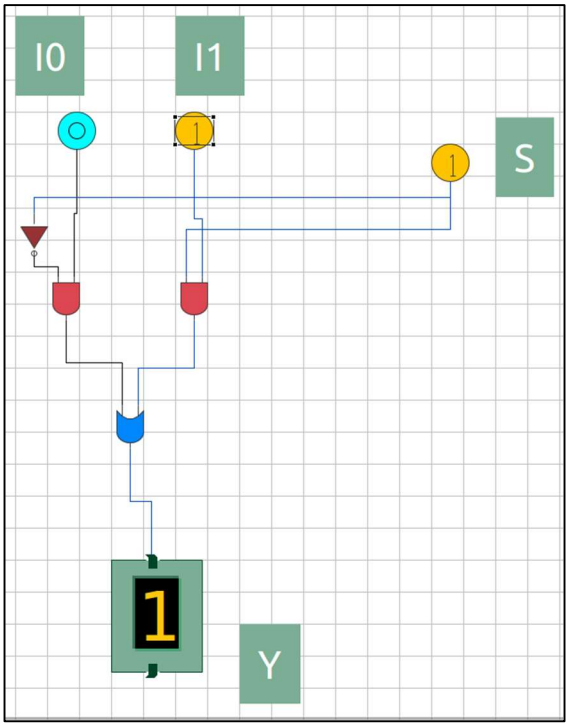



Figure 6: S = 1, I0 = 0, I1 = 1; Y=1

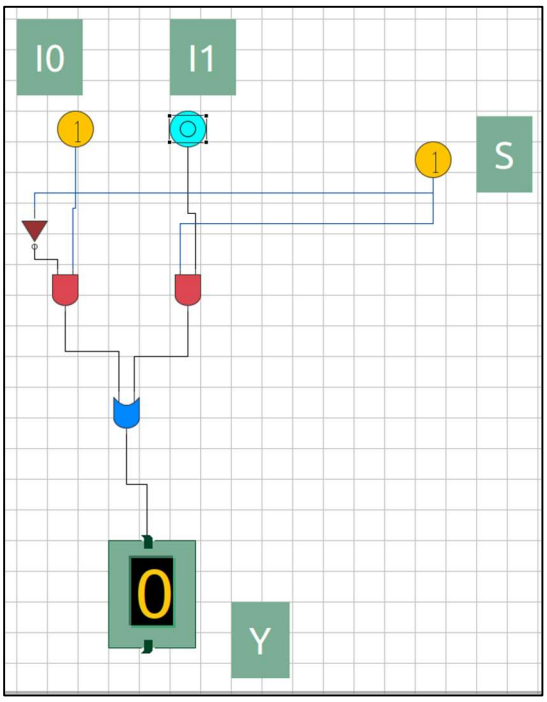



Figure 7: S = 1, I0 = 1, I1 = 0; Y=0

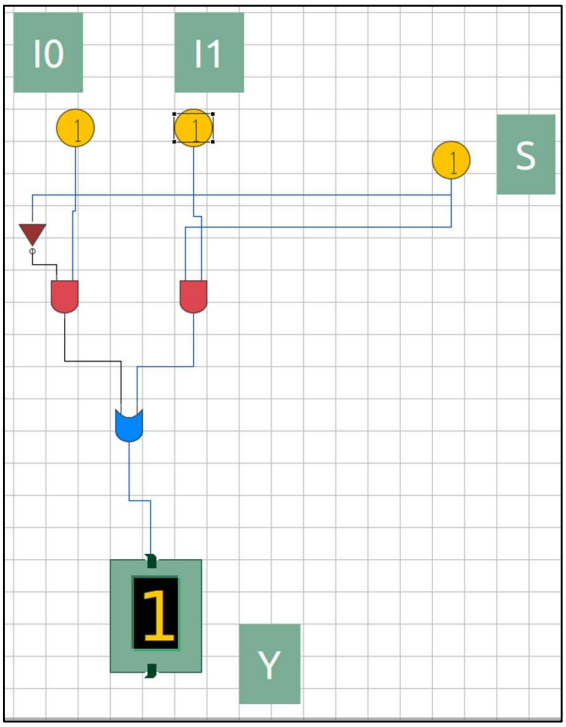



Figure 8: S = 1, I0 = 1, I1 = 1; Y=1

## EXPERIMENT 1 (b)

### Objective

Realize any 3 variable Boolean expression using 4 - to - 1 multiplexer chip.

### Theory

→ Let us realize the Full Adder Sum

→ It's Truth Table is as follows:

| Inputs | | | outputs |
|---|---|---|---|
| A | B | C | S |
| 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 |

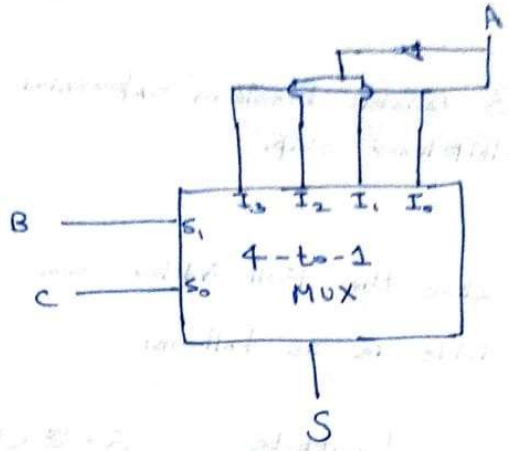
$S = \Sigma(0,1,2,4,7)$

→ as we are expressing n variable Boolean expression using $2^{n-1}$ - to - 1 Multiplexer Chip, we build the following Implementation Table.

→ (taking B and C as select lines)

| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| $\bar{A}$ | 0 | (1) | (2) | 3 |
| A | (4) | 5 | 6 | (7) |
| Ans | A | $\bar{A}$ | $\bar{A}$ | A |

Using Implication Table, we build the following circuit.




Result

The simulator outputs matches with the Truth Table

## Inputs / Outputs:

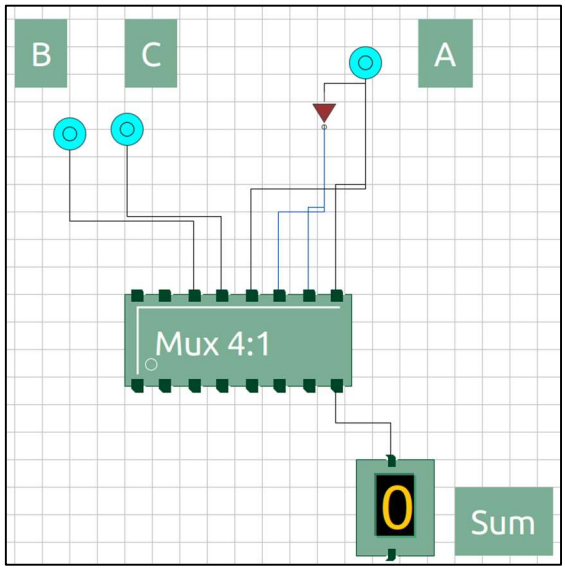



Figure 1: A = 0, B = 0, C = 0; S=0

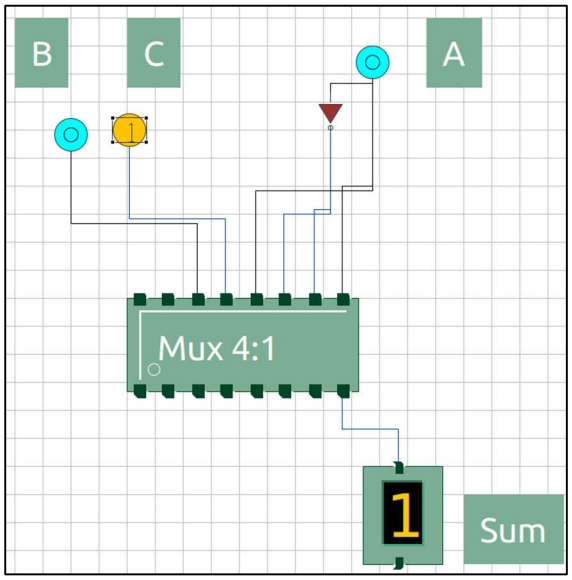



Figure 2: A = 0, B = 0, C = 1; S=1

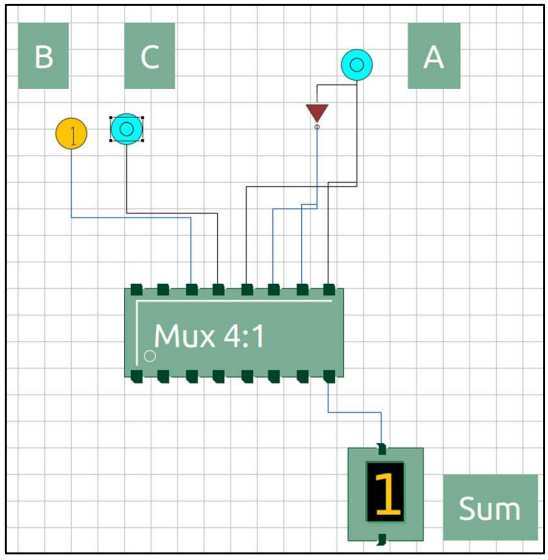



Figure 3: A = 0, B = 1, C = 0; S=1

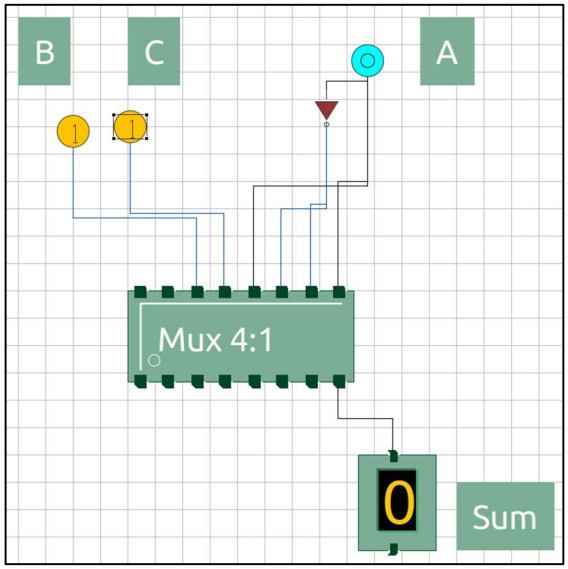



Figure 4: A = 0, B = 1, C = 1; S=0

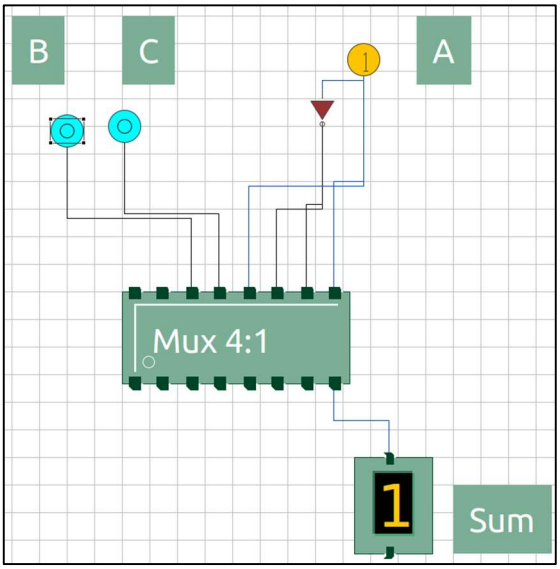



Figure 5: A = 1, B = 0, C = 0; S=1

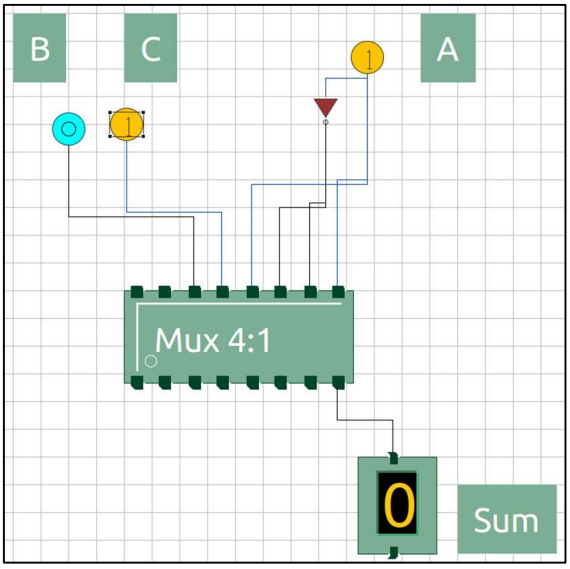



Figure 6: A = 1, B = 0, C = 1; S=0

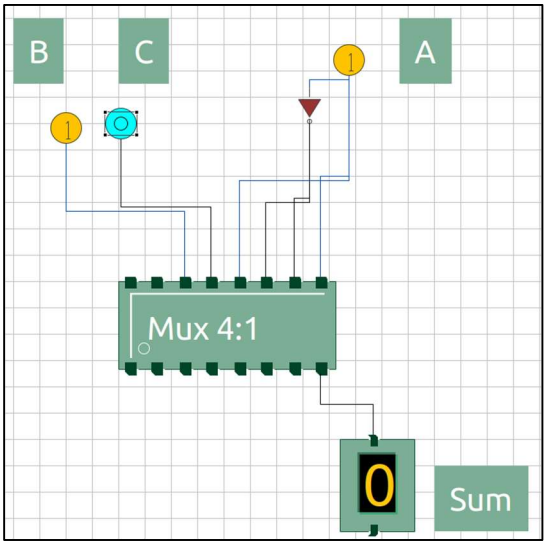



Figure 7: A = 1, B = 1, C = 0; S=0

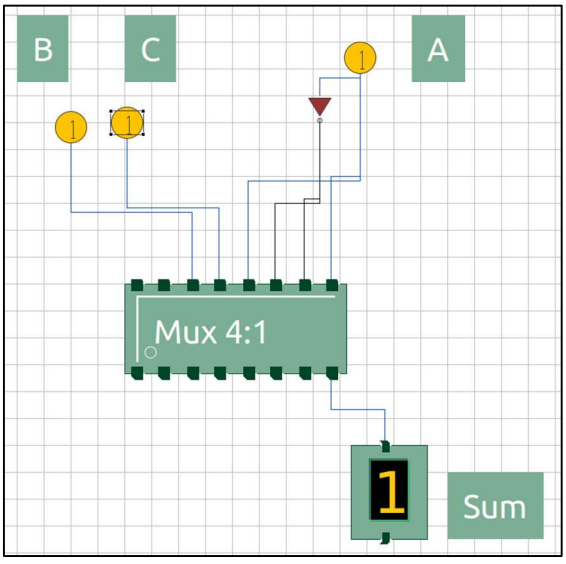



Figure 8: A = 1, B = 1, C = 1; S=1

# EXPERIMENT 1 (c)

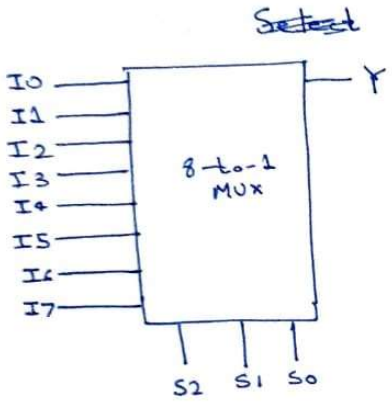
## Objective

Implement an 8-to-1 MUX using smaller multiplexers and verify the truth table.

## Theory

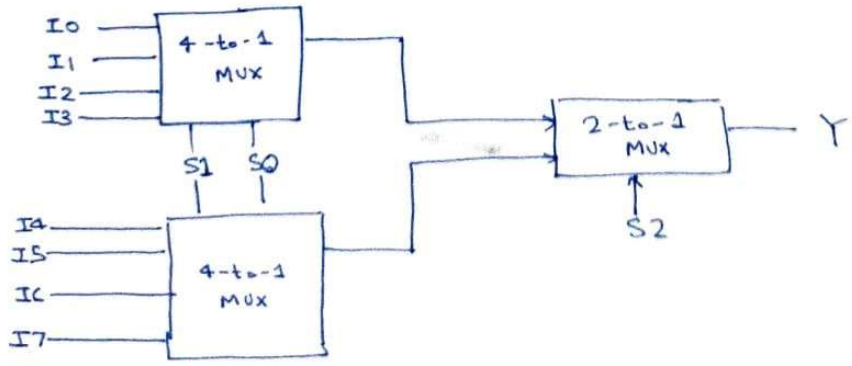
→ Due to the fact that 8-to-1 MUX has 8 input lines and 3 select lines, it's Truth Table has been simplified for easier readability

→ It's simplified Truth Table is as follows:




| Select lines | | | Output |
|---|---|---|---|
| $S_2$ | $S_1$ | $S_0$ | Y |
| 0 | 0 | 0 | I0 |
| 0 | 0 | 1 | I1 |
| 0 | 1 | 0 | I2 |
| 0 | 1 | 1 | I3 |
| 1 | 0 | 0 | I4 |
| 1 | 0 | 1 | I5 |
| 1 | 1 | 0 | I6 |
| 1 | 1 | 1 | I7 |

→ we can make 8-to-1 MUX using one 2-to-1 MUX, and two 4-to-1 MUX by following:




Result: The Simulation gives same output as that of the truth Table

## Inputs / Outputs:

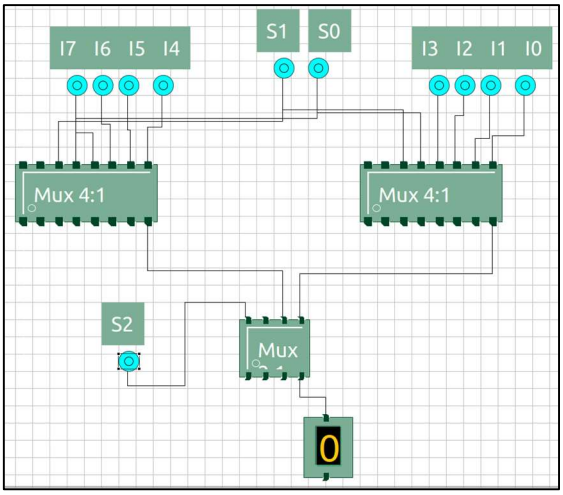



Figure 1: S2 = 0, S1 = 0, S0 = 0, I0 = 0; Y=0

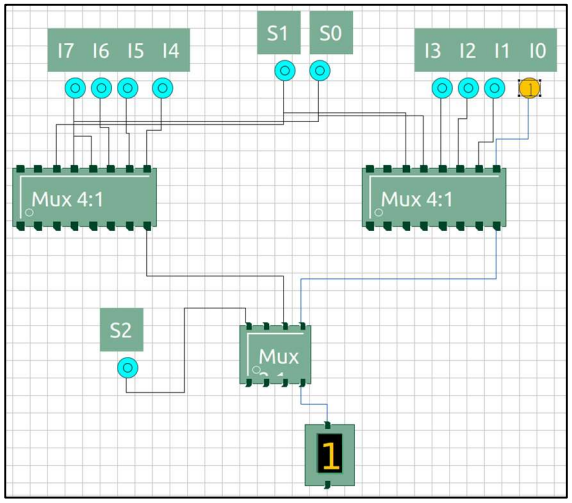



Figure 2: S2 = 0, S1 = 0, S0 = 0, I0 = 1; Y=1

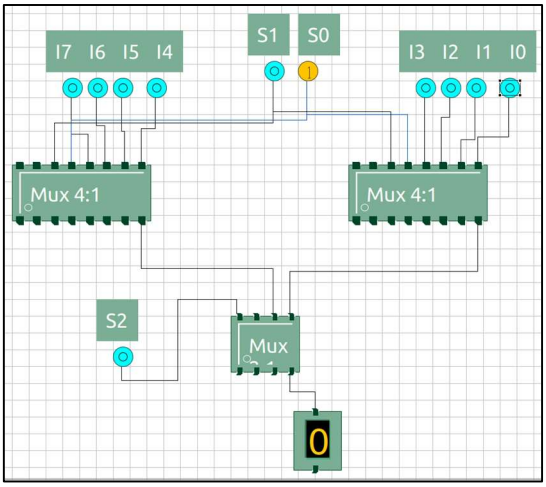



Figure 3: S2 = 0, S1 = 0, S0 = 1, I1 = 0; Y=0

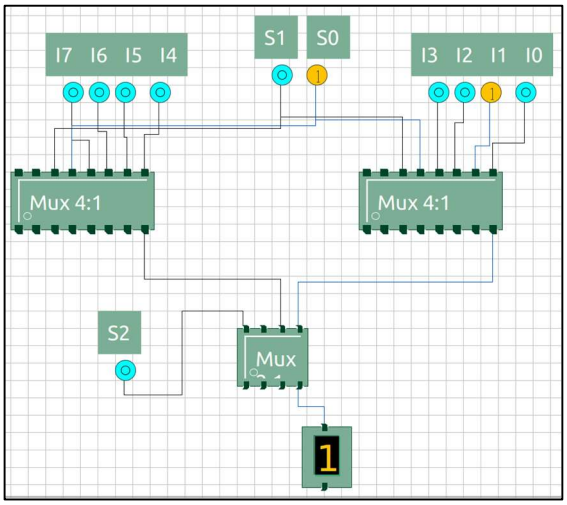



Figure 4: S2 = 0, S1 = 0, S0 = 1, I1 = 1; Y=1

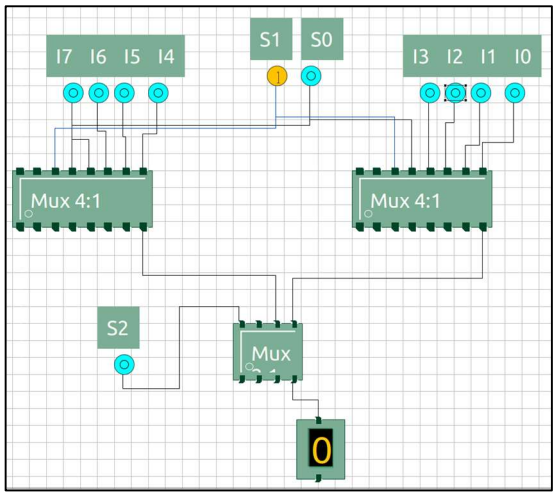



Figure 5: S2 = 0, S1 = 1, S0 = 0, I2 = 0; Y=0

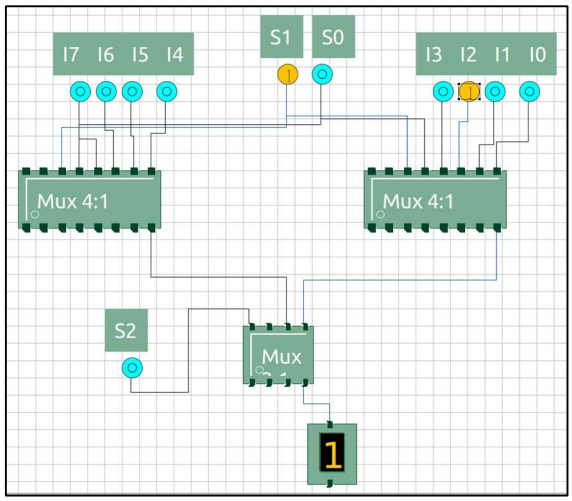



Figure 6: S2 = 0, S1 = 1, S0 = 0, I2 = 1; Y=1

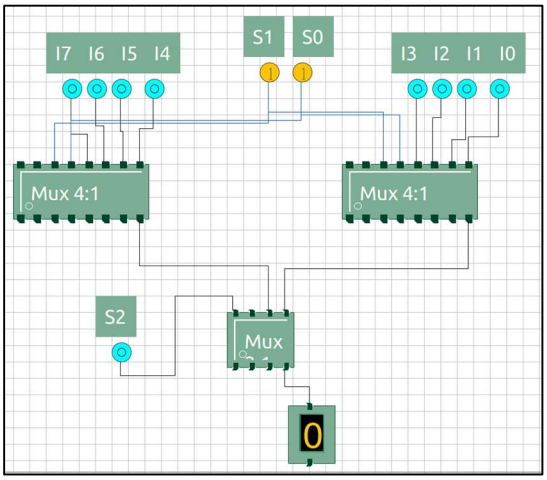



Figure 7: S2 = 0, S1 = 1, S0 = 1, I3 = 0; Y=0

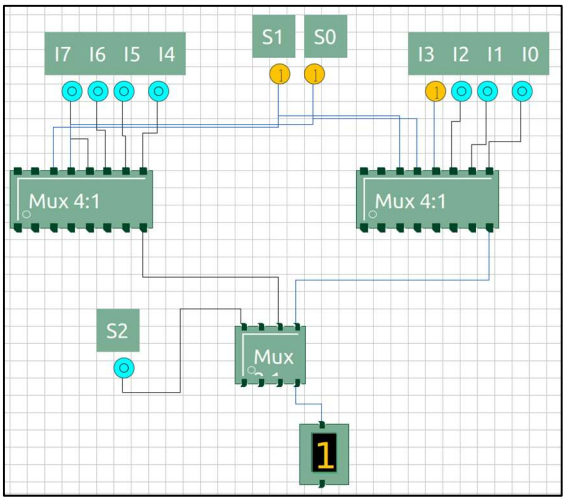



Figure 8: S2 = 0, S1 = 1, S0 = 1, I3 = 1; Y=1

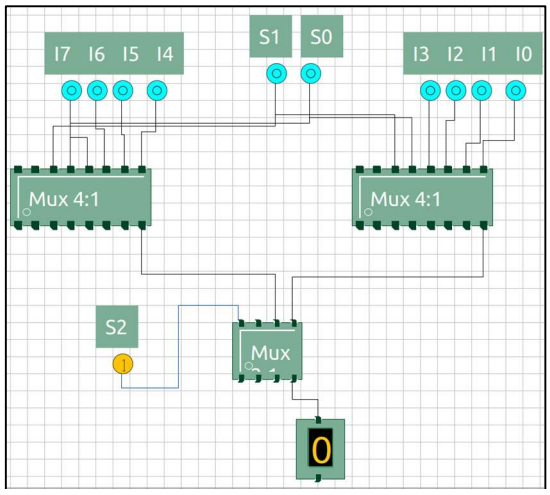



Figure 9: S2 = 1, S1 = 0, S0 = 0, I4 = 0; Y=0

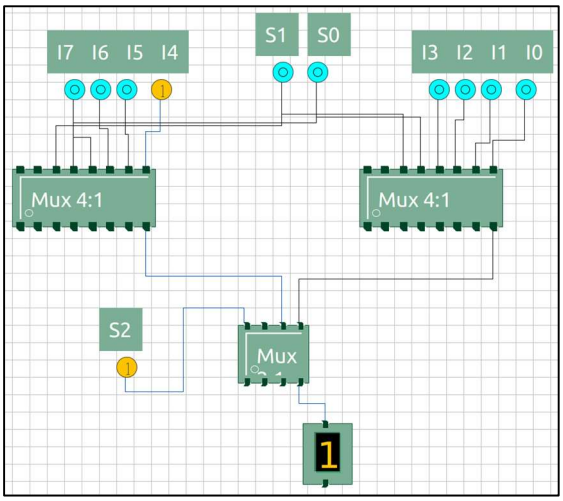



Figure 10: S2 = 1, S1 = 1, S0 = 0, I4 = 1; Y=1

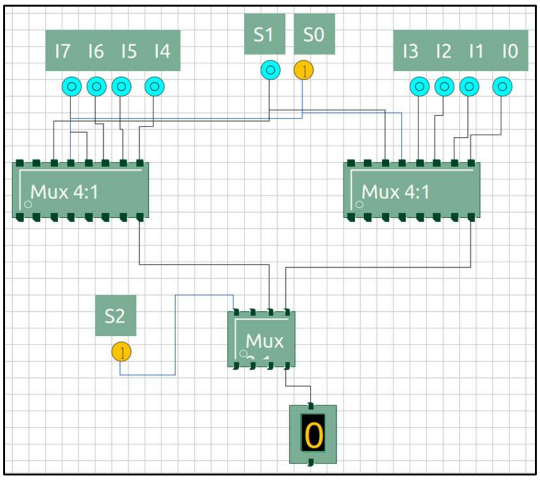



Figure 11: S2 = 1, S1 = 0, S0 = 1, I5 = 0; Y=0

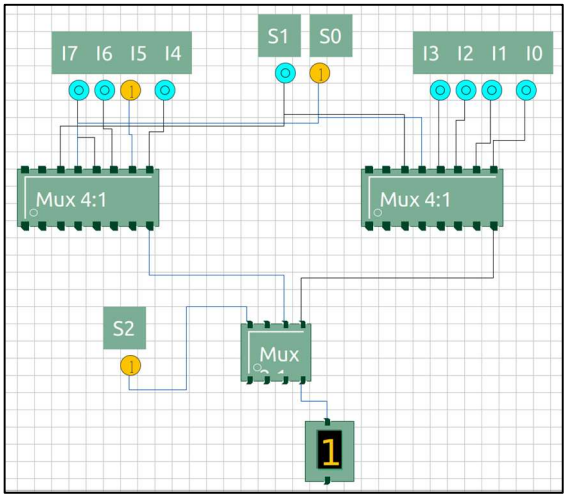



Figure 12: S2 = 1, S1 = 0, S0 = 1, I5 = 1; Y=1

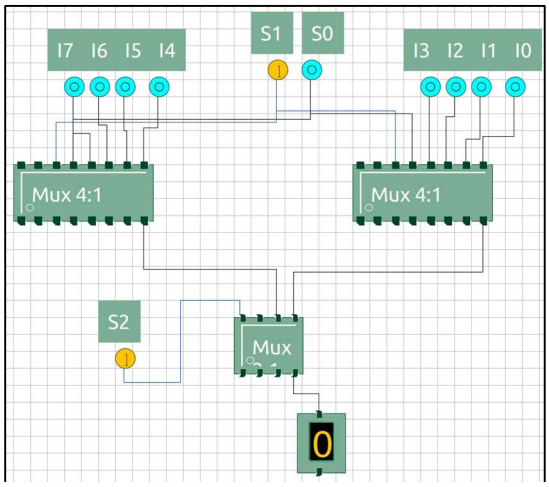



Figure 13: S2 = 1, S1 = 1, S0 = 0, I6 = 0; Y=0

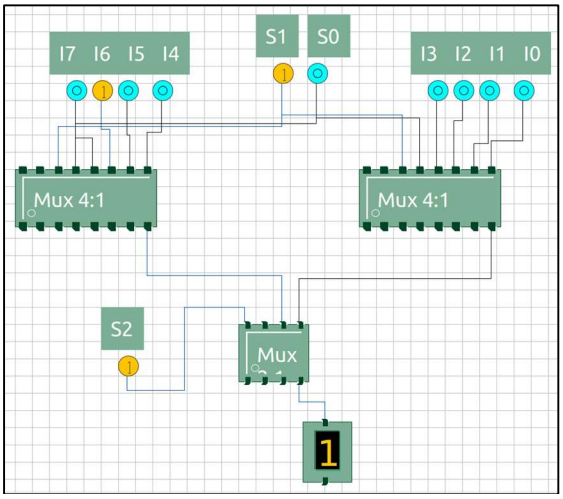



Figure 14: S2 = 1, S1 = 1, S0 = 0, I6 = 1; Y=1

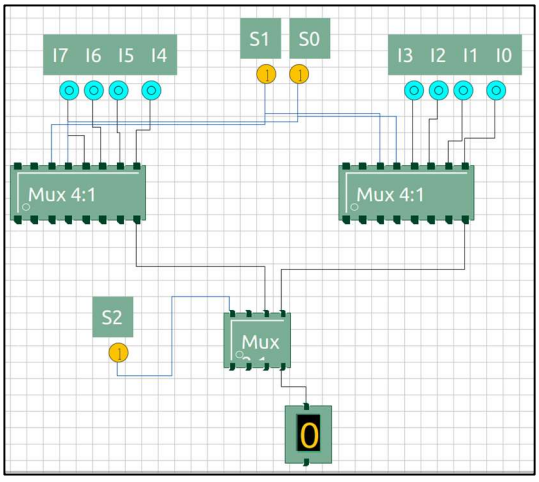



Figure 15: S2 = 1, S1 = 1, S0 = 1, I7 = 0; Y=0

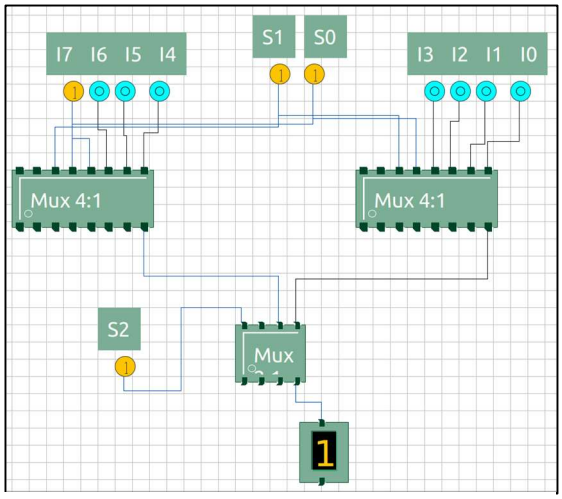



Figure 16: S2 = 1, S1 = 1, S0 = 1, I7 = 1; Y=1

## EXPERIMENT 1 (d)

**Objective**

Realize the Boolean Function $F = A'B + BC + AC'$, using 2-to-1 multiplexers

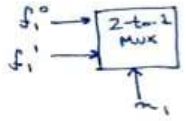
**Theory**

→ As we are only using 2-to-1 multiplexers, we are gonna use Shannon's Decomposition Theorem

→ Shannon's Decomposition Theorem states that: for any boolean function $f(x_1, x_2, \dots x_n)$ with n inputs,

$$f(x_1, x_2, \dots x_n) = \bar{x}_1 f(0, x_2, \dots x_n) + x_1 f(1, x_2, \dots x_n)$$
$$= \bar{x}_1 f_1^0 + x_1 f_1'$$

→ we do it (n-1) times to divide the boolean function to it's appropriate form for representing it via 2-to-1 MUX.




→ For the boolean Function

$$F = A'B + BC + AC'$$

Using Shanon's Theorem, we get:

$$F = \bar{A}(B + BC) + A(0 + BC + \bar{C})$$
$$= \bar{A}(B + BC) + A(BC + \bar{C})$$

now simplyfying $B + BC$

→ $B + BC = \bar{B}(0 + 0) + B(1 + 1)$
$= B$ (or we can use the formula identity)

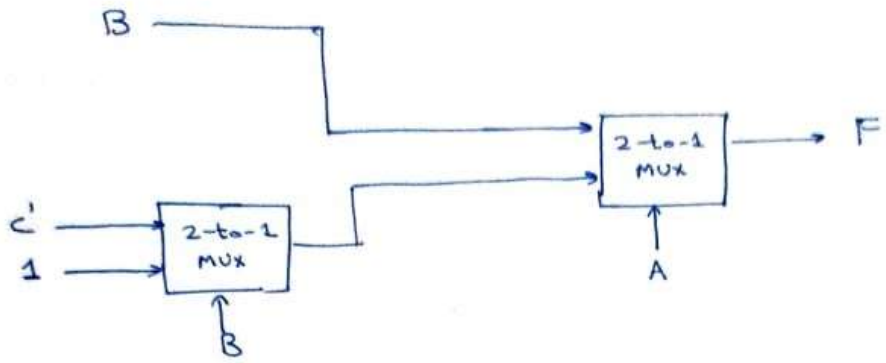
now simplyfing $BC + \bar{C}$

→ $BC + \bar{C} = \bar{B}(0 + \bar{C}) + B(C + \bar{C})$
$= \bar{B}(\bar{C}) + B$

∴ we get

$$F = \bar{A}[B] + A[\bar{B}(C) + B]$$

→ Therefore, we get the following circuit




→ Making Truth Table of F to cross-check with the circuit [ F= A'B + BC+AC']

| Inputs | | | Outputs |
|---|---|---|---|
| A | B | C | F |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Result

The simulation Result matches with the Truth Table.

## Inputs / Outputs:

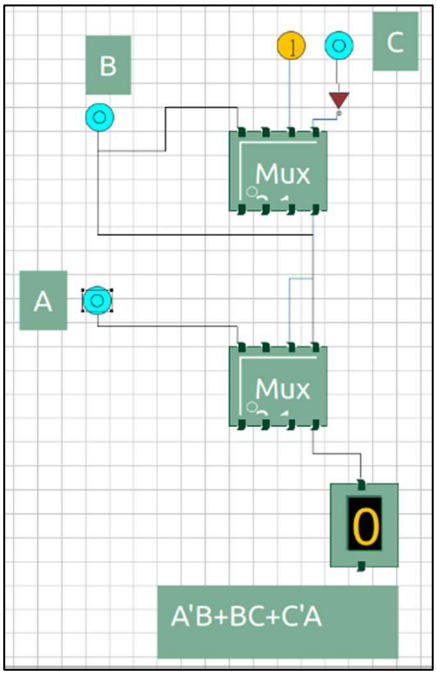



Figure 1: A = 0, B = 0, C = 0; Y=0

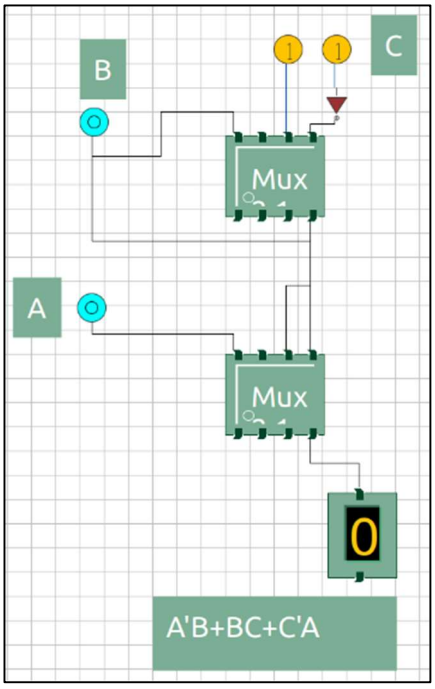



Figure 2: A = 0, B = 0, C = 1; Y=0

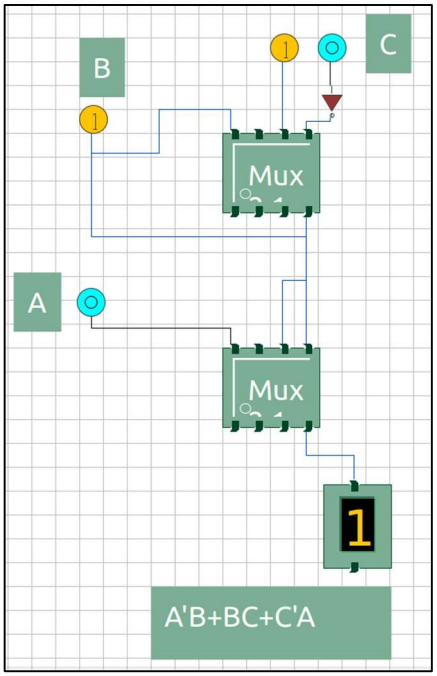



Figure 3: A = 0, B = 1, C = 0; Y=1

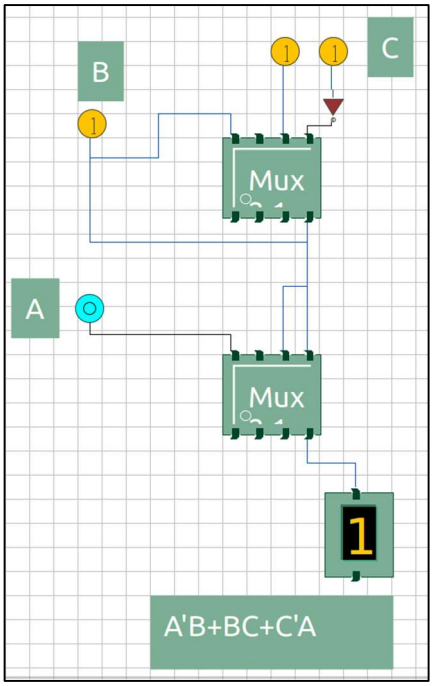



Figure 4: A = 0, B = 1, C = 1; Y=1

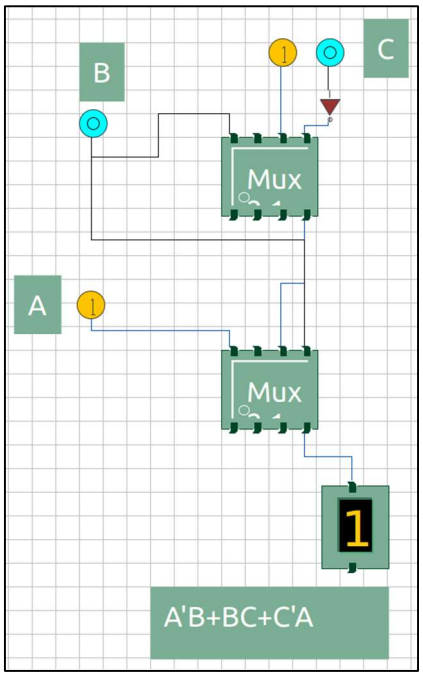



Figure 5: A = 1, B = 0, C = 0; Y=1

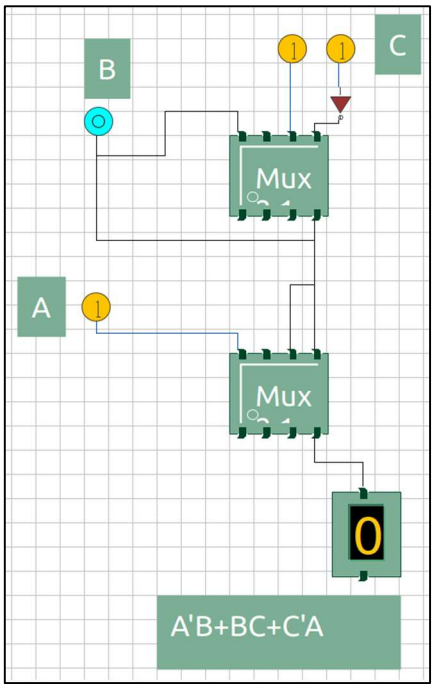



Figure 6: A = 1, B = 0, C = 1; Y=0

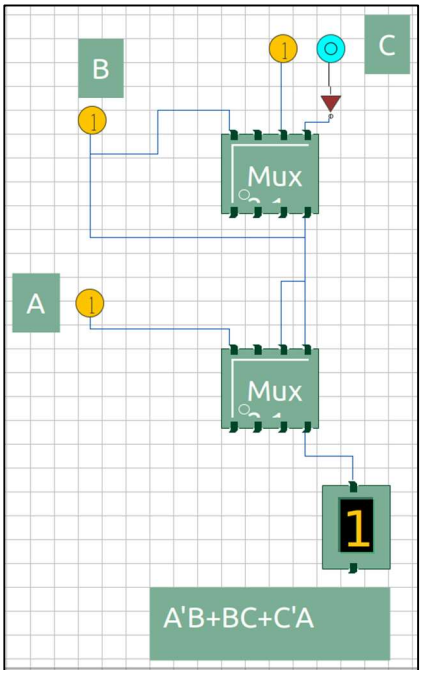



Figure 7: A = 1, B = 1, C = 0; Y=1

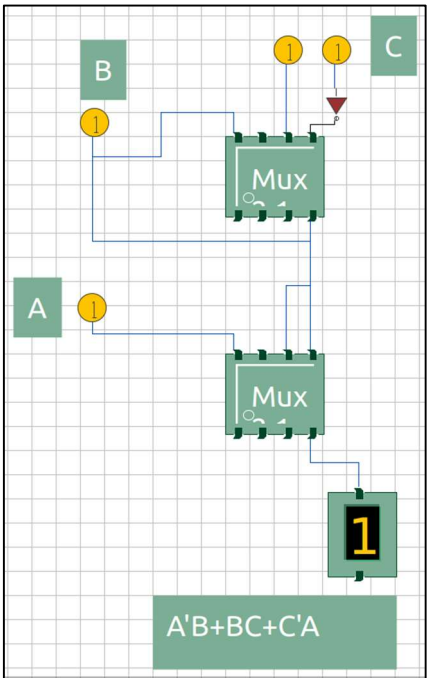



Figure 8: A = 1, B = 1, C = 1; Y=1

## EXPERIMENT 2 (a)

### Objective

Construct a 2-4 decoder using NAND gates and verify it's Truth Table. Also use dual 2-4 decoders to make a 3-8 Decoder

### Theory

→ A decoder is a combinational circuit that converts binary information from n input lines to a maximum of $2^n$ unique output lines.

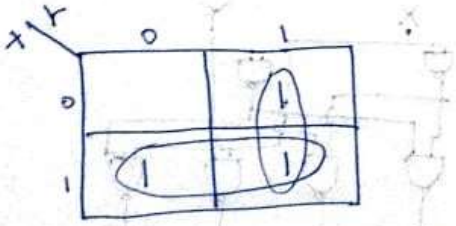
→ Truth Table of a 2-4 decoder is as follows:

| Input | | Output | | | |
|---|---|---|---|---|---|
| X | Y | D0 | D1 | D2 | D3 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

→ we can also ~~enable~~ insert "Enable" here, so that, when E=1; Output will be same, but when E=0, output will be 1 1 1 1 for any values of X and Y

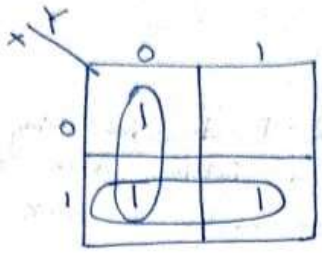
→ from the Truth Table, making K-Maps:

I) D0



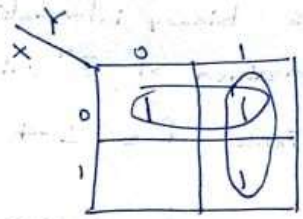

$D0 = X + Y = (X'Y')'$ (for NAND gate)

II) D1



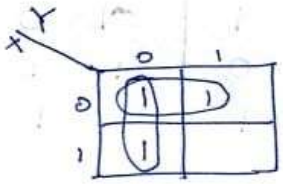

$D1 = X + Y' = (X'Y)'$

III) D2



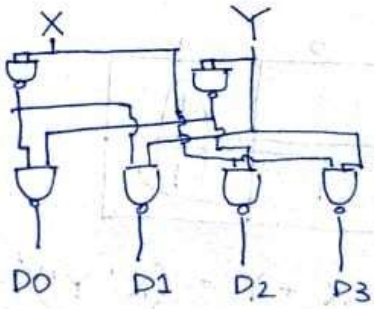

$D2 = X' + Y = (XY')'$

IV) D3



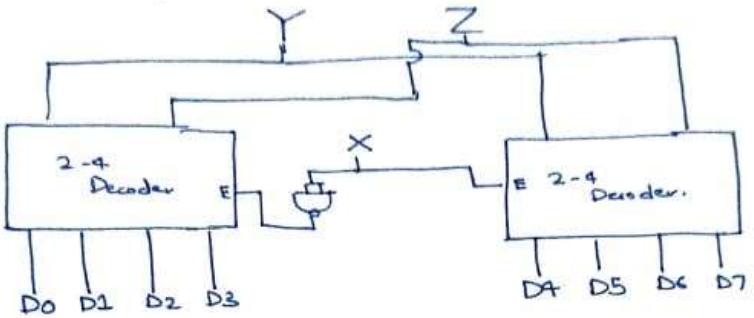

$D3 = X' + Y' = (XY)'$

→ Using the expressions, we can make the following circuit



→ for making 3-8 decoder, using 2-4 decoder, we can use the fact that we can "turn off" a decoder using E = 0.

→ Circuit Diagram of 3-8 Decoder using 2-4 decoder




→ Truth Table for checking circuit:

| Inputs | | | Outputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| X | Y | Z | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Result:

Output from the Simulation matches with the Truth Table.

Following Gates were used

I) 2-4 Decoder: 6 NAND gates

II) 3-8 Decoder: 2 2-4 Decoder Chips and 1 NAND

## Inputs / Outputs:

1. 2-4 Decoder

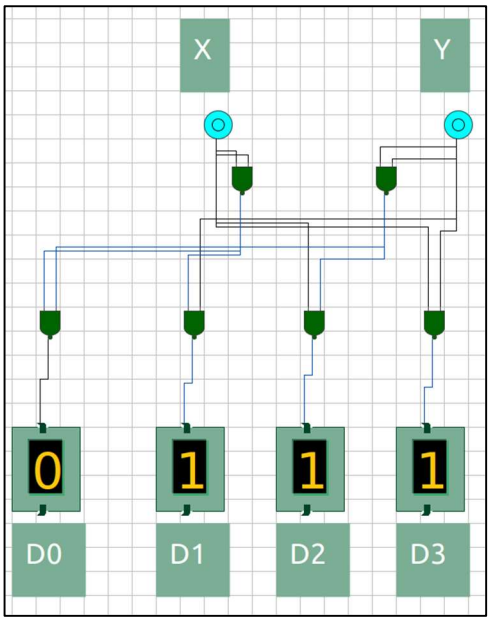



Figure 1: X = 0, Y = 0; D0 = 0, D1 = 1, D2 = 1, D3 = 1

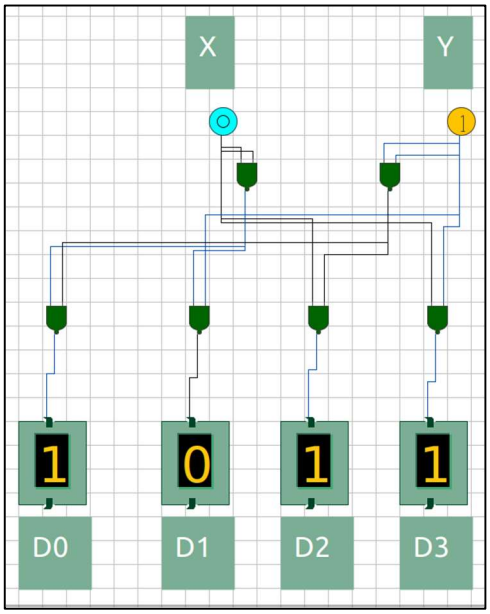



Figure 2: X = 0, Y = 1; D0 = 1, D1 = 0, D2 = 1, D3 = 1

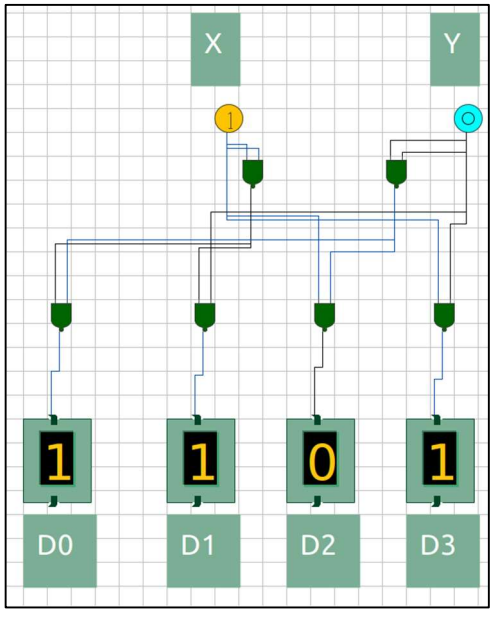



Figure 3: X = 1, Y = 0; D0 = 1, D1 = 1, D2 = 0, D3 = 1

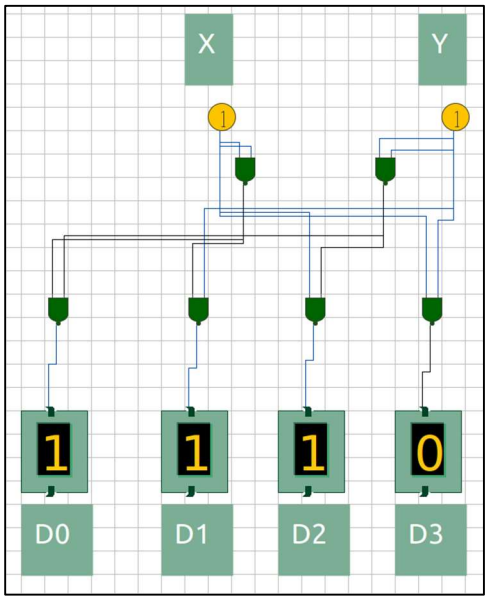



Figure 4: : X = 1, Y = 1; D0 = 1, D1 = 1, D2 = 1, D3 = 0

2. 3-8 Decoder using Dual 2-4 Decoder

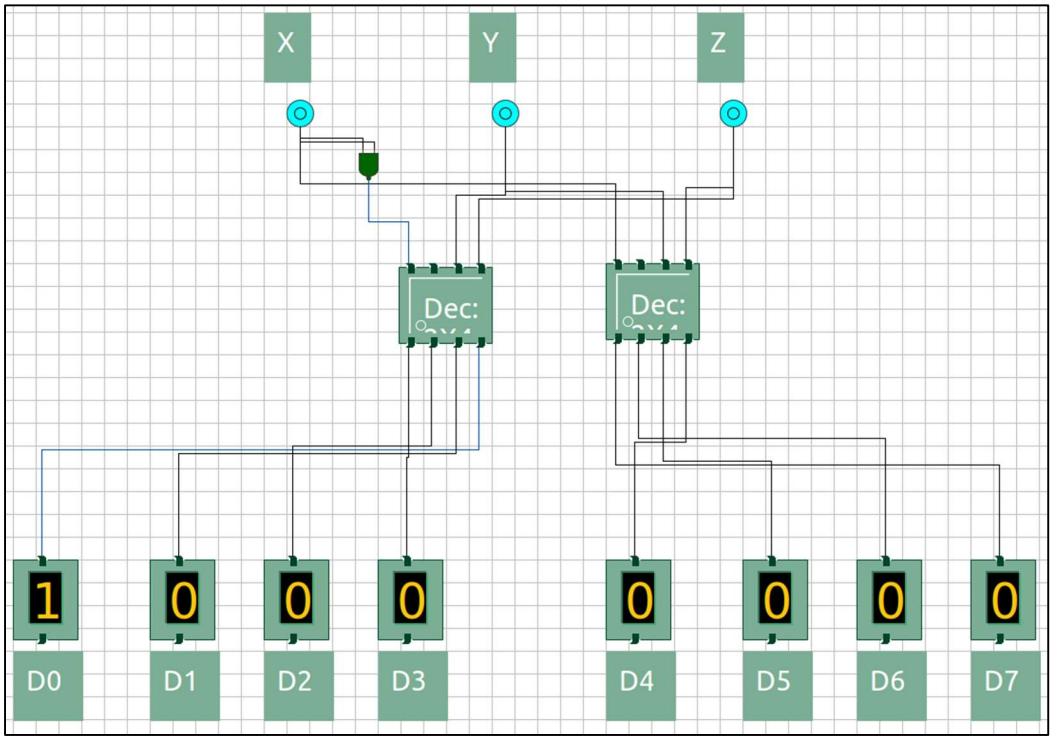



Figure 1: X = 0, Y = 0, Z = 0

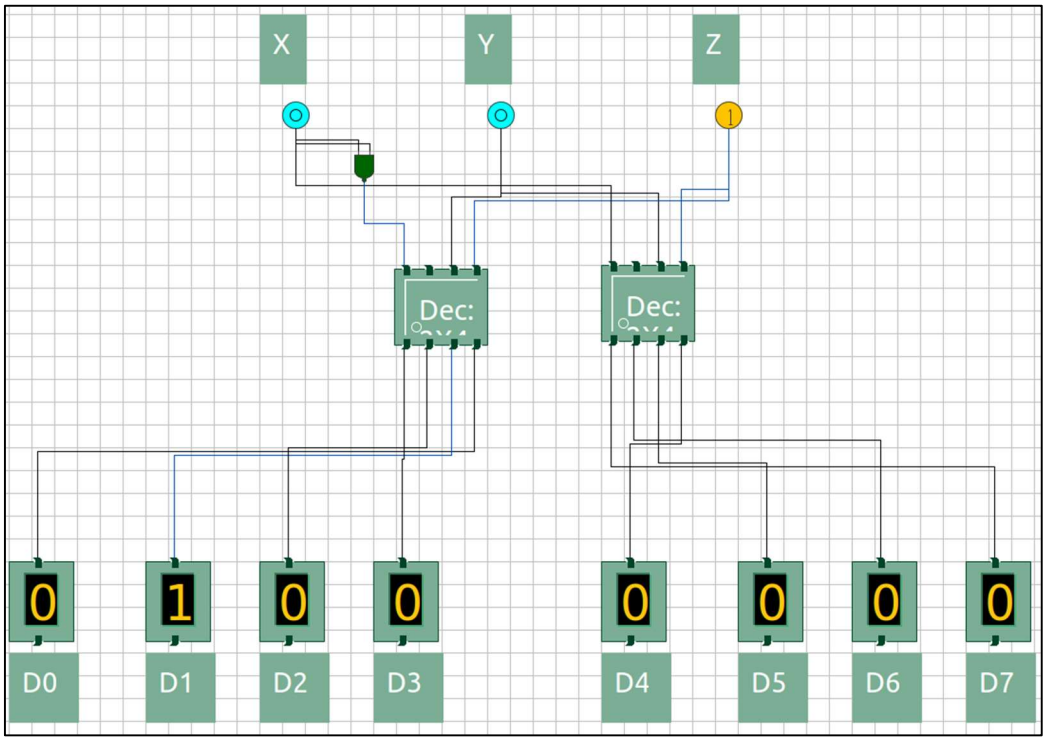



Figure 2: X = 0, Y = 0, Z = 1

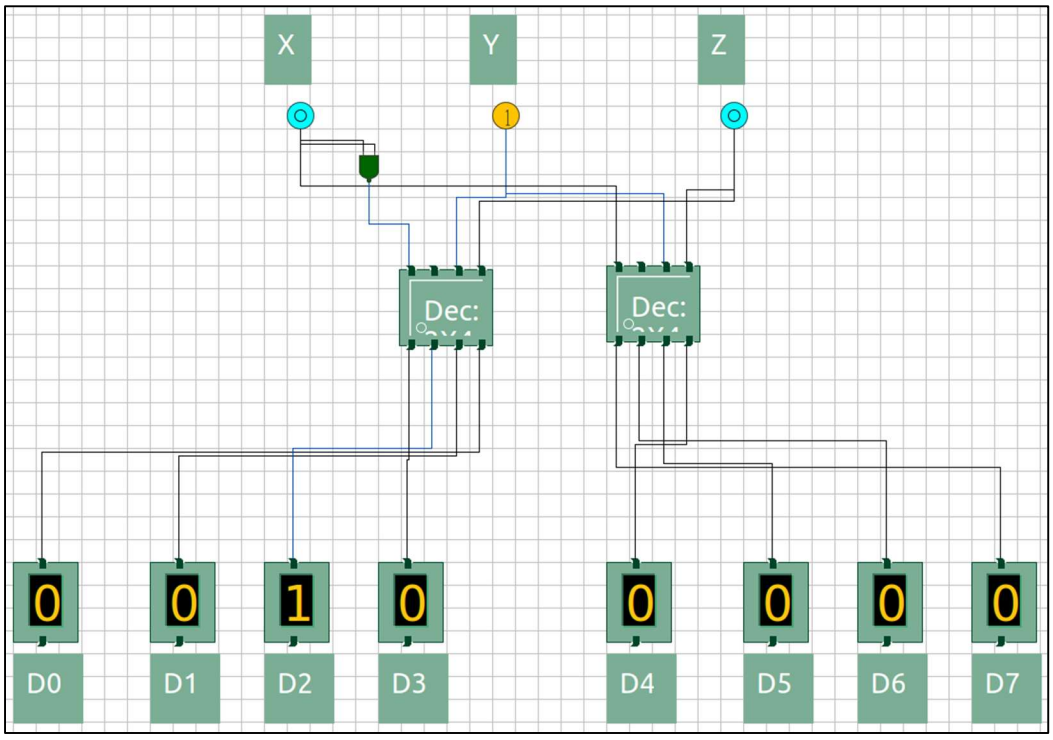



Figure 3: X = 0, Y = 1, Z = 0

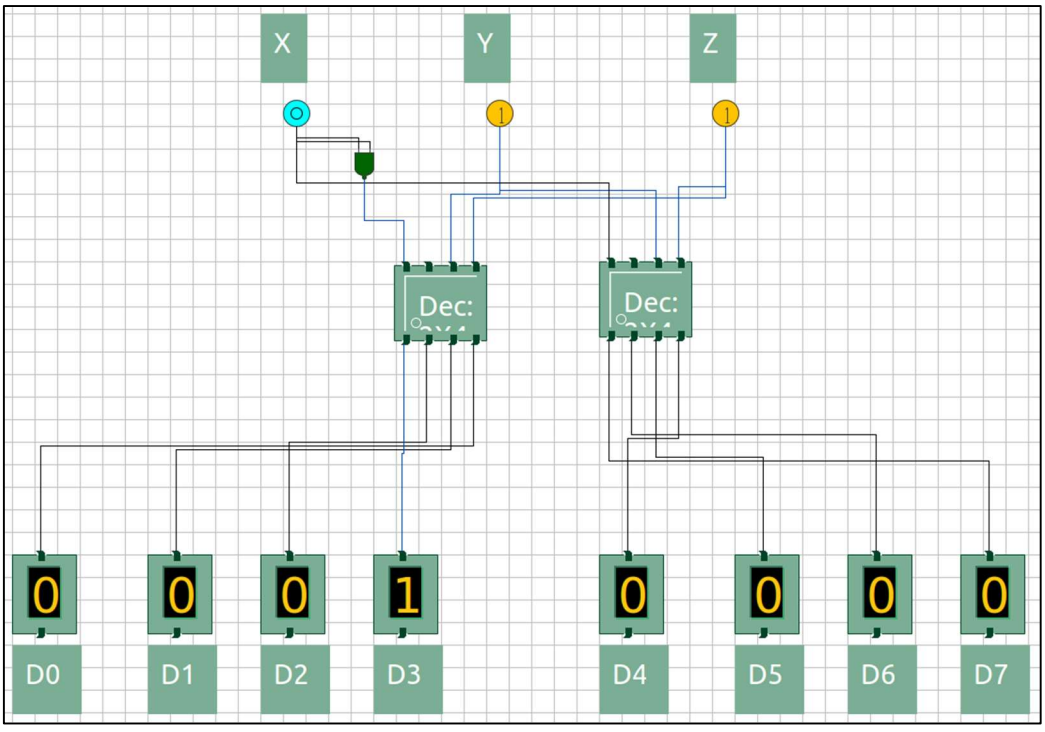



Figure 4: X = 0, Y = 1, Z = 1

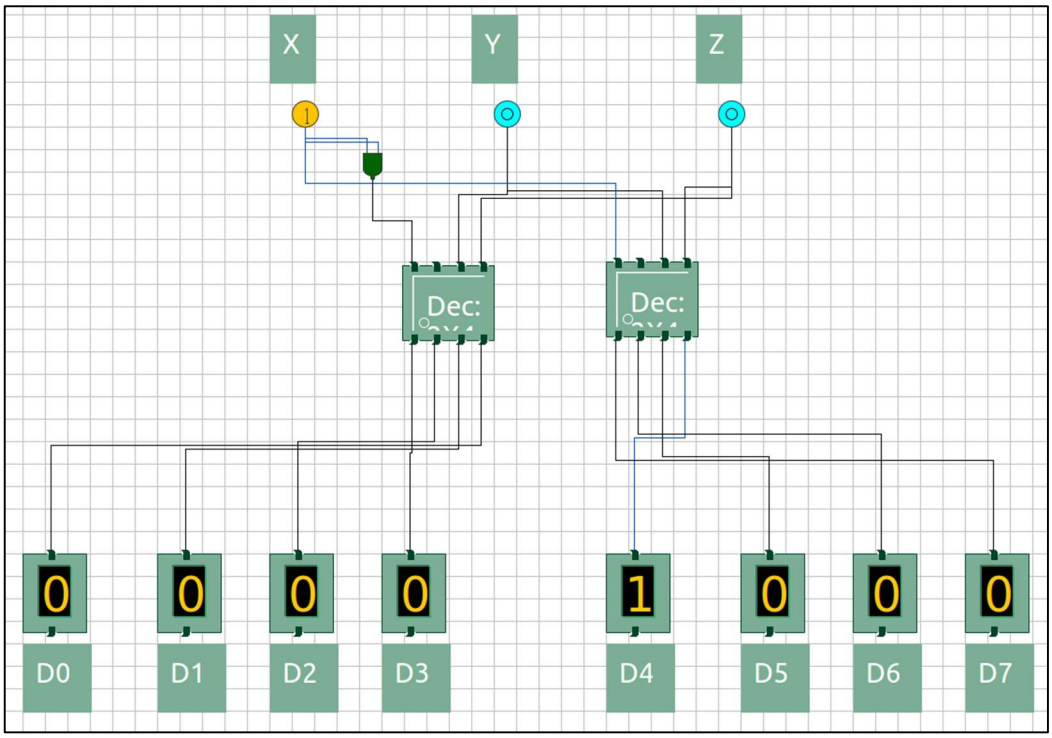



Figure 5: X = 1, Y = 0, Z = 0

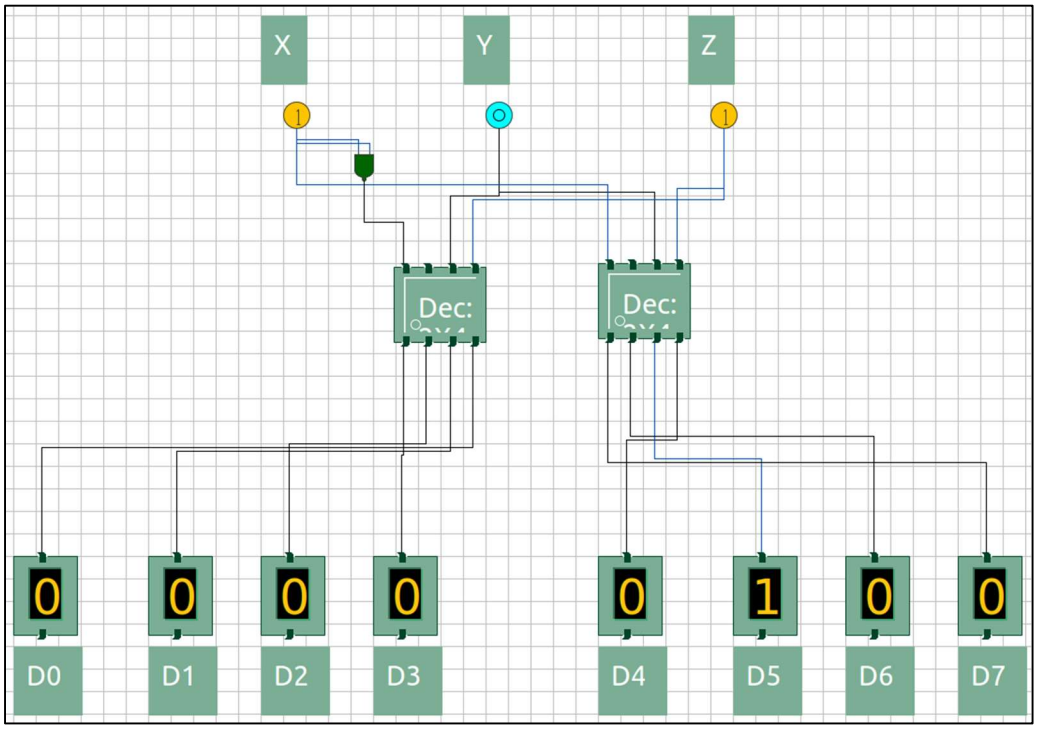



Figure 6: X = 1, Y = 0, Z = 1

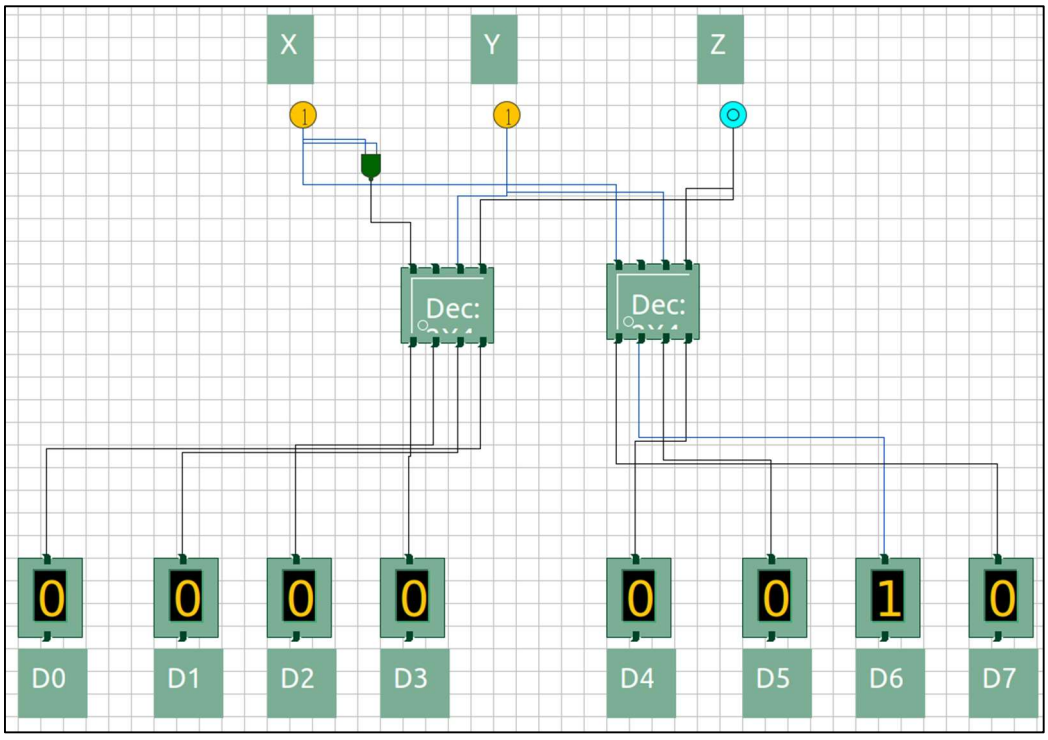



Figure 7: X = 1, Y = 1, Z = 0

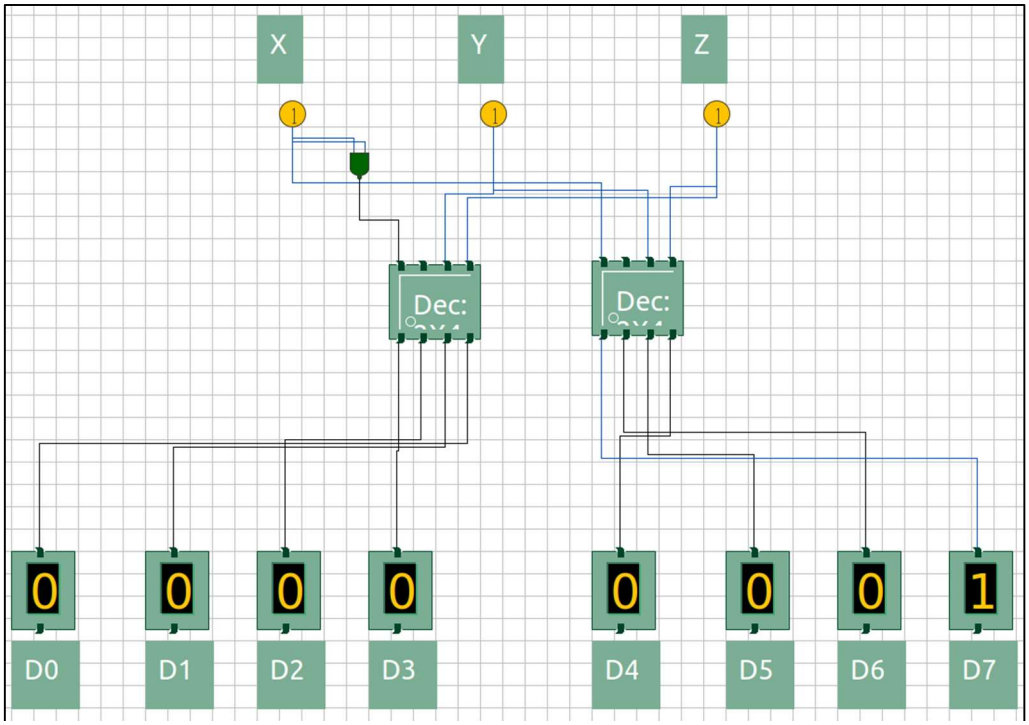



Figure 8: X = 1, Y = 1, Z = 1

# EXPERIMENT 2 (b)

## Objective

Realize any three variable Boolean Expression using a 3-8 Decoder with minimum extra logic.

## Theory

→ Let us realize the Full Adder Sum

→ It's Truth Table is as follows:

| Inputs | | | Output |
|---|---|---|---|
| X | Y | Z | S |
| 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 |

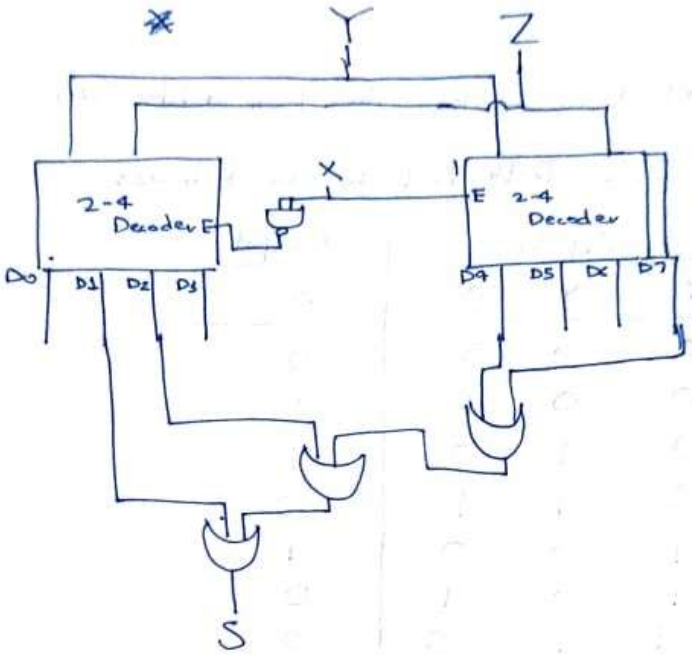
→ from the truth Table, we can say that

$$S = \Sigma(1, 2, 4, 7)$$

→ So to realize it, with a 3-8 decoder, we just ~~a~~ boolean OR, D1, D2, D4, D7, - for this we can use a 4 input OR gate, or 3 three 2 input OR gates.

→ we will be using the same 3-8 decoder made in experiment 2(a)

→ So the Circuit Diagram will be as follows




Result

→ The Simulation Results matches with the Truth Table

→ two 2-4 Decoder, one NAND gate and 3 OR gates were used in this circuit

## Inputs / Outputs:

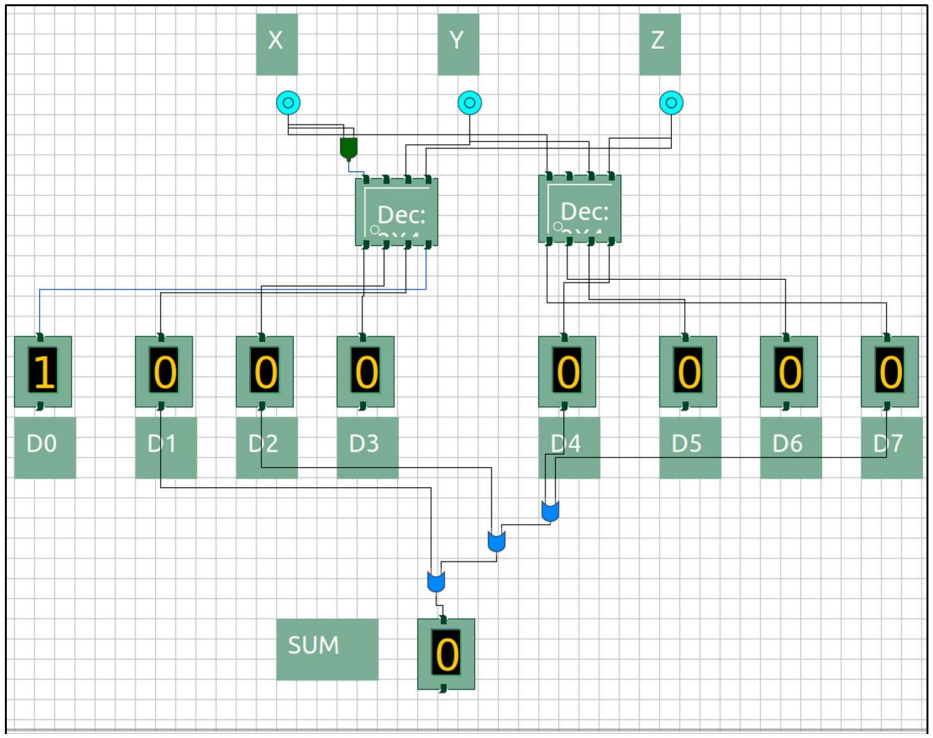



Figure 2: X = 0, Y = 0, Z = 0; S = 0

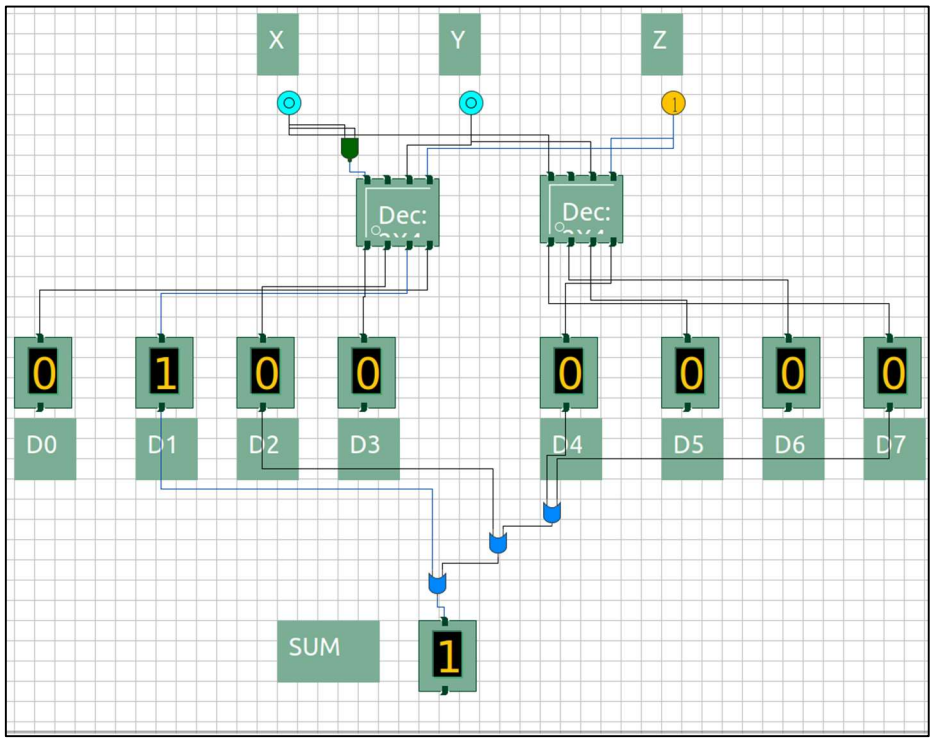



Figure 2: X = 0, Y = 0, Z = 1; S = 1

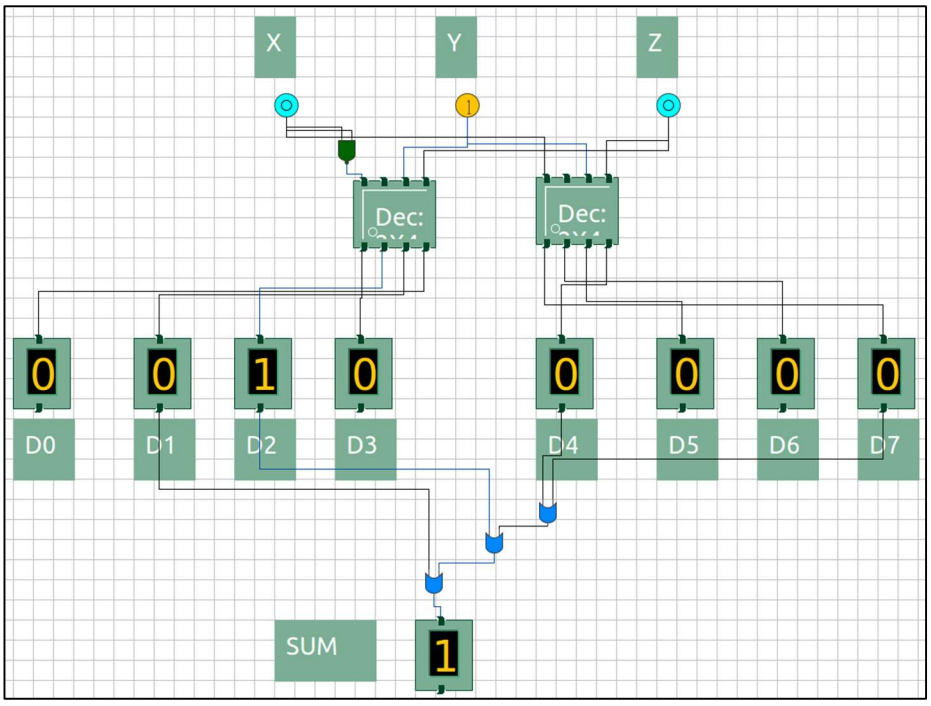



Figure 3: X = 0, Y = 1, Z = 0; S = 1

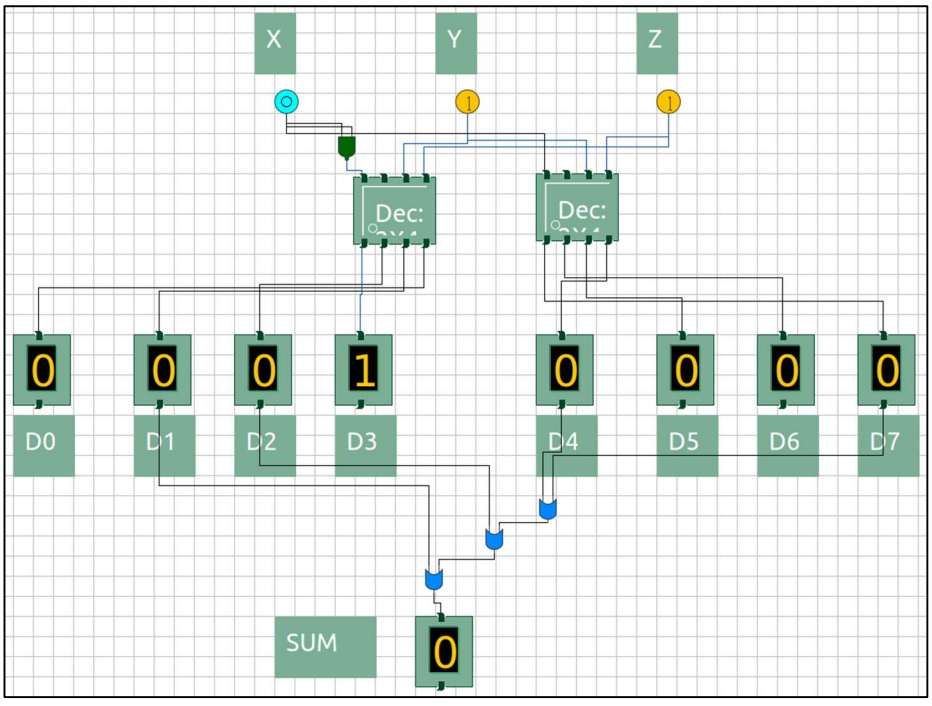



Figure 4: X = 0, Y = 1, Z = 1; S = 0

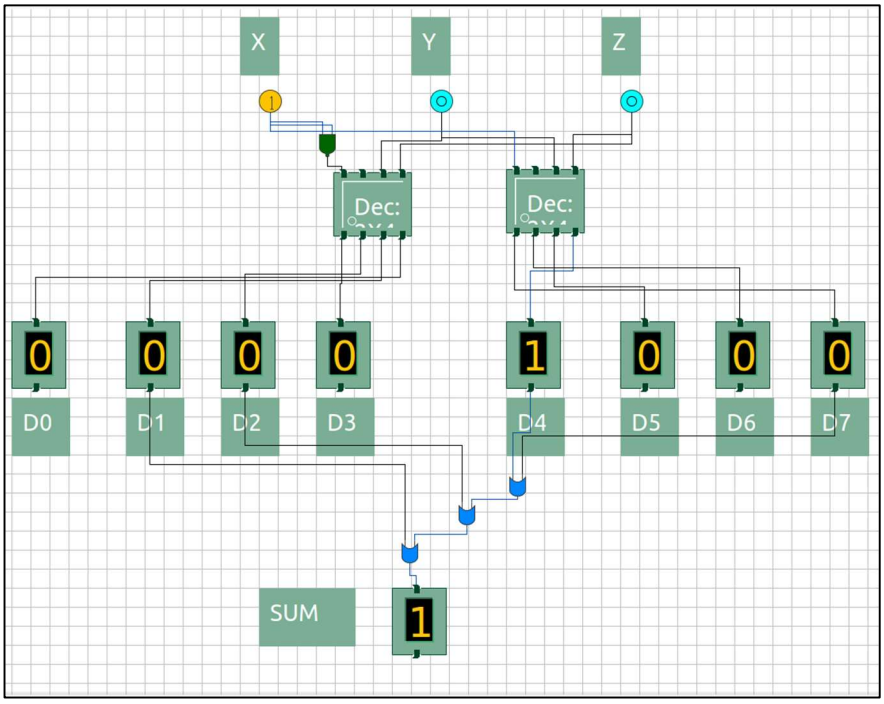



Figure 5: X = 1, Y = 0, Z = 0; S = 1

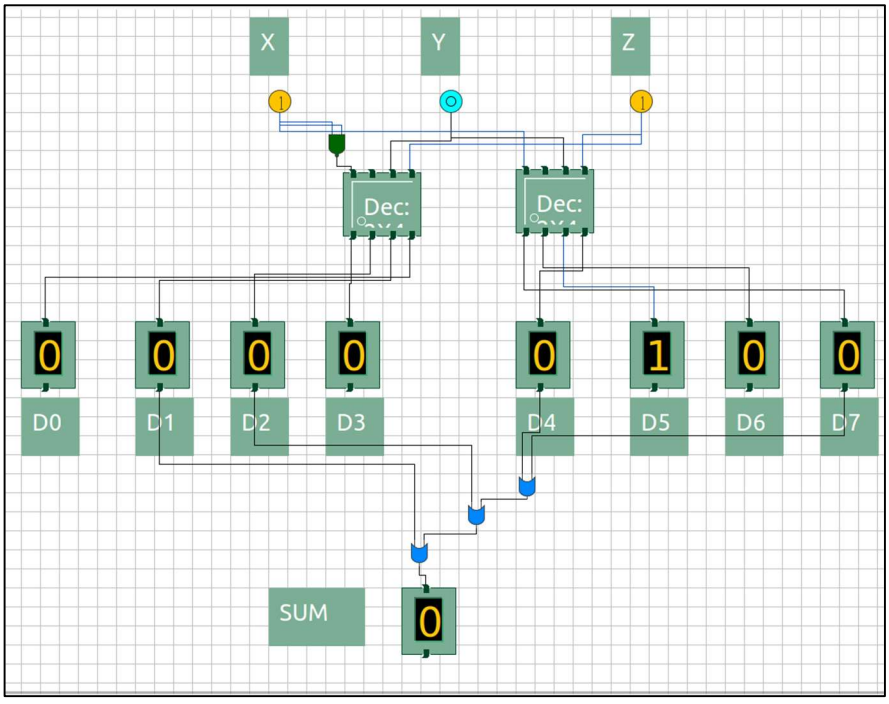



Figure 6: X = 1, Y = 0, Z = 1; S = 0

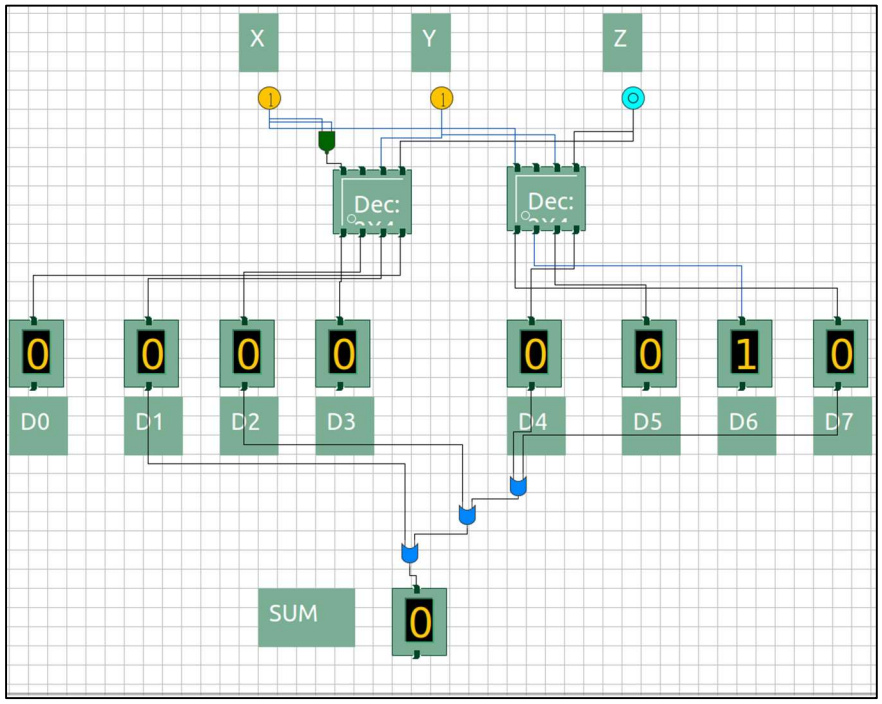



Figure 7: X = 1, Y = 1, Z = 0; S = 0

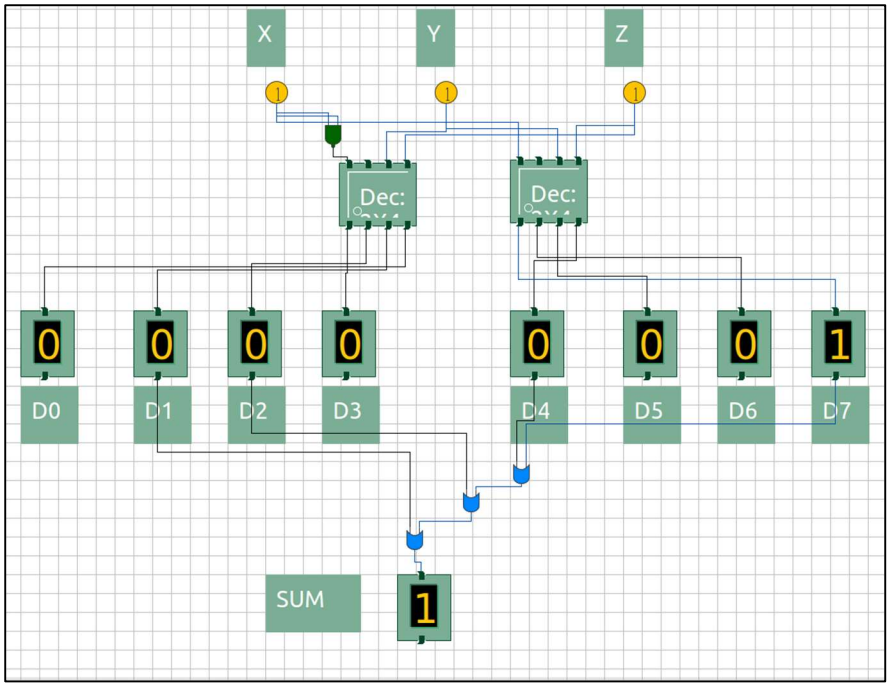



Figure 8: X = 1, Y = 1, Z = 1; S = 1