21BDS0340

Abhinav Dinesh Srivatsa

Digital Systems Design Lab

Task 3

S. No.	Components	Page No.	Student Check Mark	RA Check Mark
	PART I-Multiplexer based Design	4	✓	
1	Aim	4	✓	
2	Components Required and Tools Required	4	✓	
3	Procedure	4	✓	
4	Pin diagram	4	✓	
5	Truth Table and Boolean expression of Multiplexer, Demultiplexer.	5	✓	
6	Multiplexer Theory	7	✓	
7	Demultiplexer Theory	7	✓	
8	Block diagram/Circuit diagram for implementing 16:1 MUX using 8:1 MUX	7	✓	
9	Block diagram and Circuit diagram for implementing 8:1 MUX using 4:1 MUX	7	✓	
10	Block diagram and Circuit diagram for implementing 4:1 MUX using 2:1 MUX	8	✓	
11	SOP canonical expression of MUX and circuit implemented in MUX	8	✓	
12	POS canonical expression of MUX and circuit implemented in MUX	8	✓	
13	Circuit diagram of 2:1 MUX Circuit using SOP Equation --- AOI circuit	9	✓	
14	Circuit diagram of 2:1 MUX Circuit using POS Equation --- OAI circuit	9	✓	
15	Circuit diagram of 2:1 MUX Circuit using SOP	10	✓	

Part I – Multiplexer Based Design

Aim

Using Reg.no. formulate expressions in SOP and POS for F and F '. Implement the circuits using only

a. Design 16:1 Multiplexer using 8:1 Multiplexer constructed using 4:1 Multiplexer constructed using 2:1 Multiplexer.
b. Give the Internal structure of 2:1 Multiplexer using SOP, POS, NAND, NOR logic design, CMOS, and transmission gates.
c. Implement a Combinational circuit framed by your registration number using only
    a. 16:1 Multiplexer.
    b. 8:1 Multiplexer.
d. Write the Verilog code for Multiplexer in different styles and verify the results using the truth table and show the output waveform.
e. Show the steps and procedure in the tool used.

Components Required

a. AND, OR, NOT, NAND and NOR gates
b. 5V voltage source
c. Led indicator

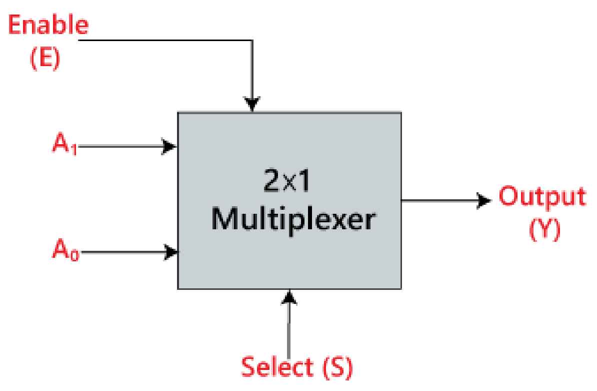
Tools Required

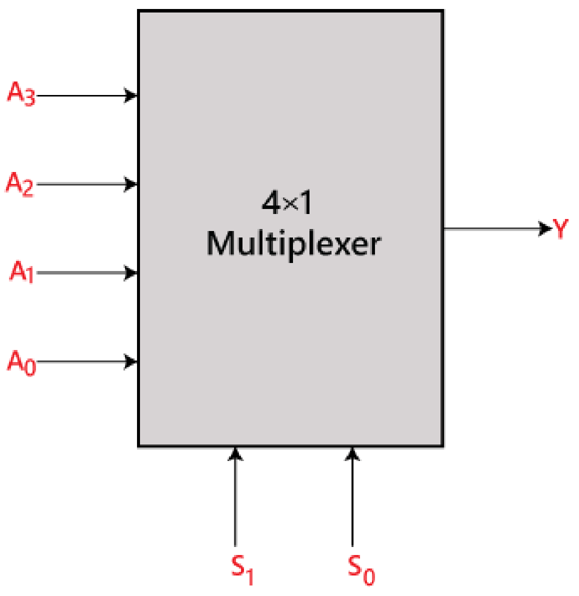
a. Multisim simulator

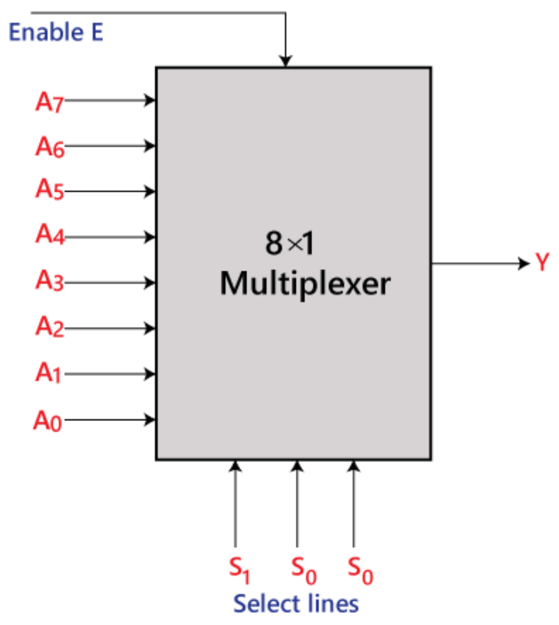
Procedure

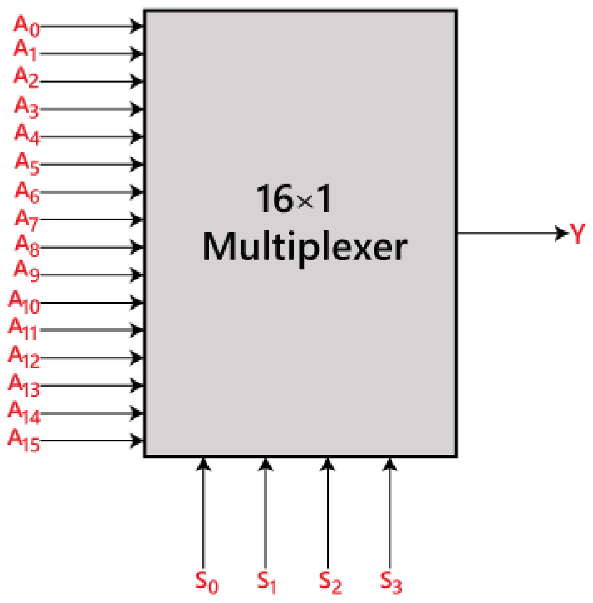
1. Draw circuit diagrams to convert X:1 mux to X/2:1 mux till 2:1
2. Write SOP and POS canonical forms of mux equation
3. Simplify registration number using implementation table and represent with mux

Pin Diagram









<u>Truth Table and Boolean expression of Multiplexer, Demultiplexer</u>

Multiplexer:

Binary Equivalent	S_0	S_1	S_2	S_3	Y
0	0	0	0	0	A_0
1	0	0	0	1	A_1
2	0	0	1	0	A_2
3	0	0	1	1	A_3
4	0	1	0	0	A_4
5	0	1	0	1	A_5
6	0	1	1	0	A_6
7	0	1	1	1	A_7
8	1	0	0	0	A_8
9	1	0	0	1	A_9
10	1	0	1	0	A_{10}
11	1	0	1	1	A_{11}
12	1	1	0	0	A_{12}
13	1	1	0	1	A_{13}
14	1	1	1	0	A_{14}
15	1	1	1	1	A_{15}

$Y = A_0.S_0'S_1'S_2'S_3' + A_1.S_0'S_1'S_2'S_3 + A_2.S_0'S_1'S_2S_3' + A_3.S_0'S_1'S_2S_3 + A_4.S_0'S_1S_2'S_3' + A_5.S_0'S_1S_2'S_3 + A_6.S_0'S_1S_2S_3' + A_7.S_0'S_1S_2S_3 + A_8.S_0S_1'S_2'S_3' + A_9.S_0S_1'S_2'S_3 + A_{10}.S_0S_1'S_2S_3' + A_{11}.S_0S_1'S_2S_3 + A_{12}.S_0S_1S_2'S_3' + A_{13}.S_0S_1S_2'S_3 + A_{14}.S_0S_1S_2S_3' + A_{15}.S_0S_1S_2S_3$

S_0	S_1	S_2	S_3	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
0	0	0	0	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	A	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	A	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	A	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	A	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	A	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	A	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	A	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	A	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A

$A_0 = S_0'S_1'S_2'S_3'$

$A_1 = S_0'S_1'S_2'S_3$

$A_2 = S_0'S_1'S_2S_3'$

$A_3 = S_0'S_1'S_2S_3$

$A_4 = S_0'S_1S_2'S_3'$

$A_5 = S_0'S_1S_2'S_3$

$A_6 = S_0'S_1S_2S_3'$

$A_7 = S_0'S_1S_2S_3$

$A_8 = S_0S_1'S_2'S_3'$

$A_9 = S_0S_1'S_2'S_3$

$A_{10} = S_0S_1'S_2S_3'$

$A_{11} = S_0S_1'S_2S_3$

$A_{12} = S_0S_1S_2'S_3'$

$A_{13} = S_0S_1S_2'S_3$

$A_{14} = S_0S_1S_2S_3'$

$A_{15} = S_0S_1S_2S_3$

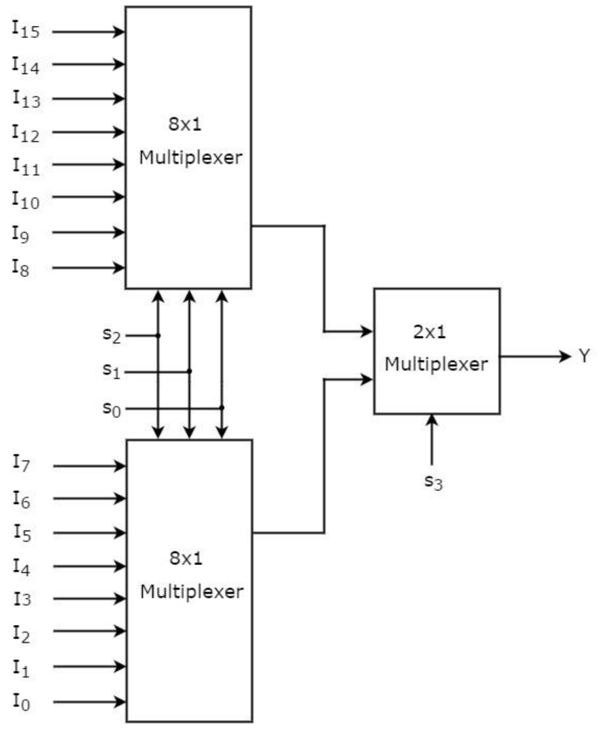
Multiplexer Theory

The *multiplexer*, shortened to “MUX” or “MPX”, is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. Multiplexers operate like very fast acting multiple position rotary switches connecting or controlling multiple input lines called “channels” one at a time to the output.

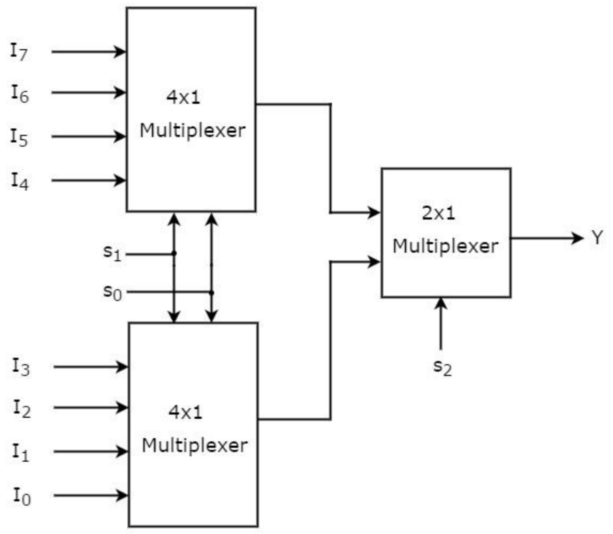
Demultiplexer Theory

The *demultiplexer* takes one single input data line and then switches it to any one of several individual output lines one at a time. The demultiplexer converts a serial data signal at the input to a parallel data at its output lines as shown below.

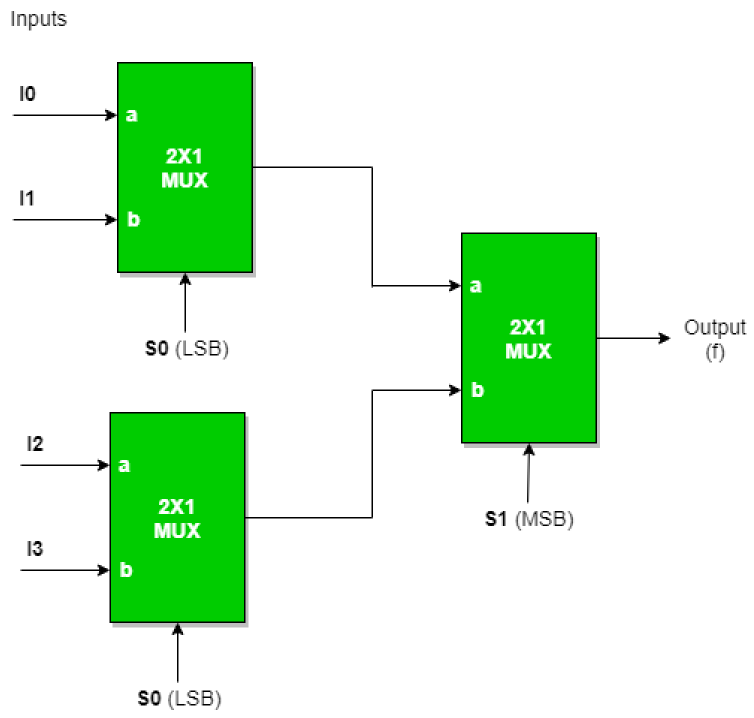
Block diagram/Circuit diagram for implementing 16:1 MUX using 8:1 MUX



Block diagram and Circuit diagram for implementing 8:1 MUX using 4:1 MUX



Block diagram and Circuit diagram for implementing 4:1 MUX using 2:1 MUX



SOP canonical expression of MUX and circuit implemented in MUX

$F = S_0'S_1'S_2'S_3' + S_0'S_1'S_2'S_3 + S_0'S_1'S_2S_3' + S_0'S_1'S_2S_3 + S_0'S_1S_2'S_3' + A_{11}.S_0S_1'S_2S_3 + A_{13}.S_0S_1S_2'S_3$

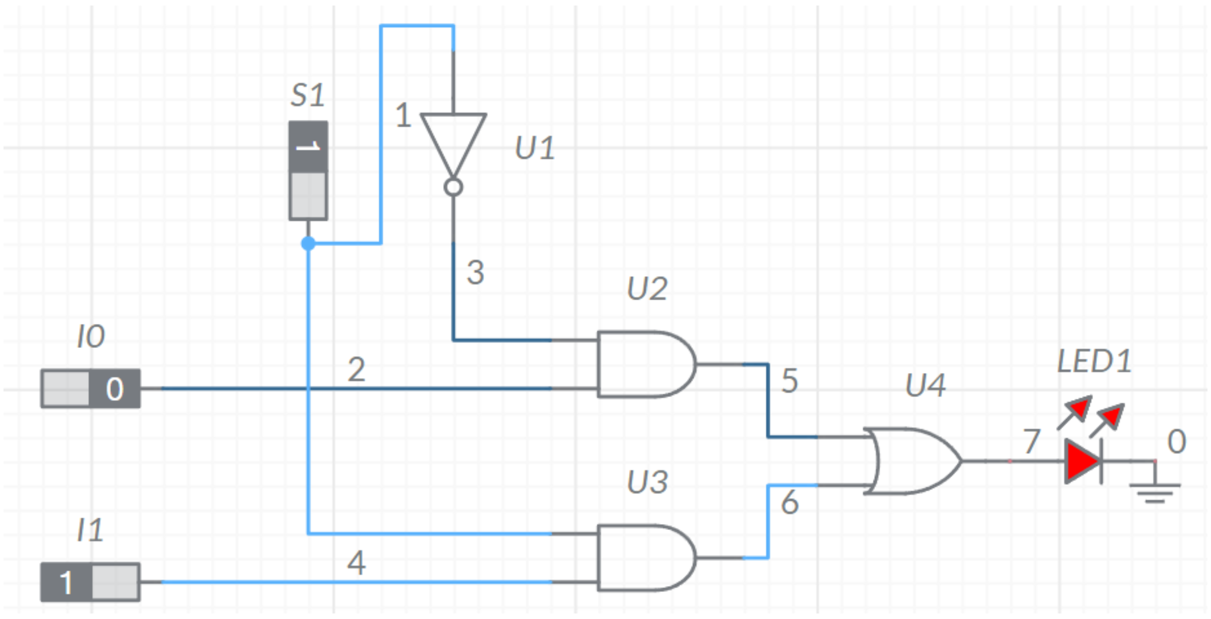
$F' = S_0'S_1S_2'S_3 + S_0'S_1S_2S_3' + S_0'S_1S_2S_3 + S_0S_1'S_2'S_3' + S_0S_1'S_2'S_3 + S_0S_1'S_2S_3' + S_0S_1S_2'S_3' + S_0S_1S_2S_3' + S_0S_1S_2S_3$

POS canonical expression of MUX and circuit implemented in MUX

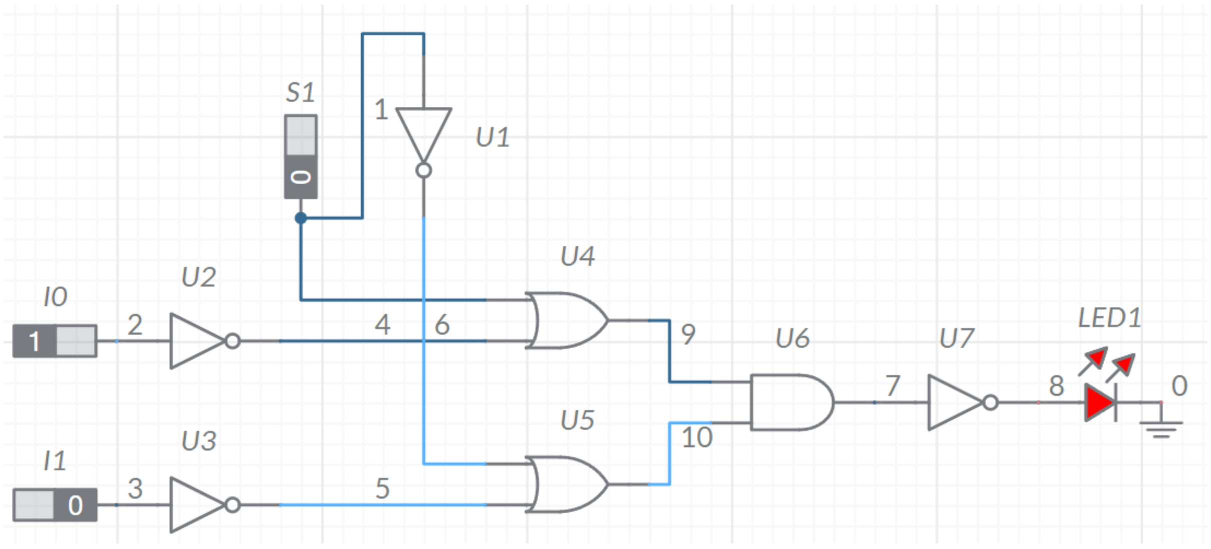
$F = (S_0+S_1+S_2+S_3)\ (S_0+S_1+S_2+S_3')\ (S_0+S_1+S_2'+S_3)\ (S_0+S_1+S_2'+S_3')\ (S_0+S_1'+S_2+S_3)\ (S_0'+S_1+S_2'+S_3')\ (S_0'+S_1'+S_2+S_3')$

$F' = (S_0+S_1'+S_2+S_3')\ (S_0+S_1'+S_2'+S_3)\ (S_0+S_1'+S_2'+S_3')\ (S_0'+S_1+S_2+S_3)\ (S_0'+S_1+S_2+S_3')\ (S_0'+S_1+S_2'+S_3)\ (S_0'+S_1'+S_2+S_3)\ (S_0'+S_1'+S_2'+S_3)\ (S_0'+S_1'+S_2'+S_3')$

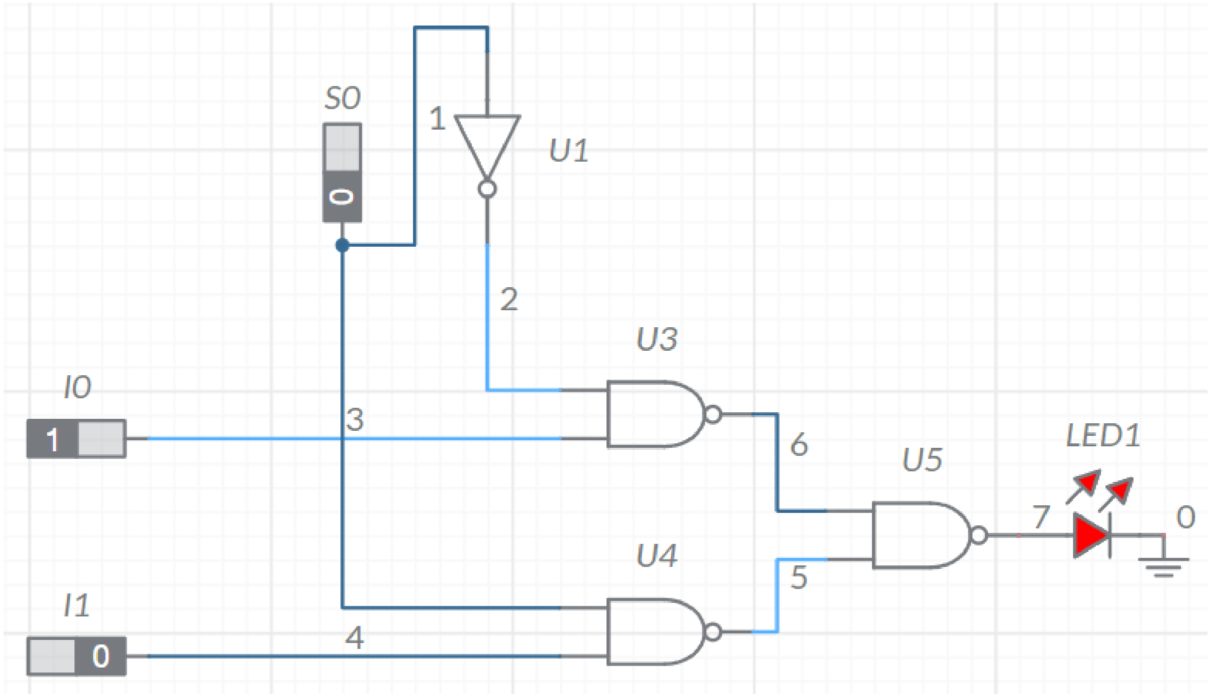
Circuit diagram of 2:1 MUX Circuit using SOP Equation --- AOI circuit



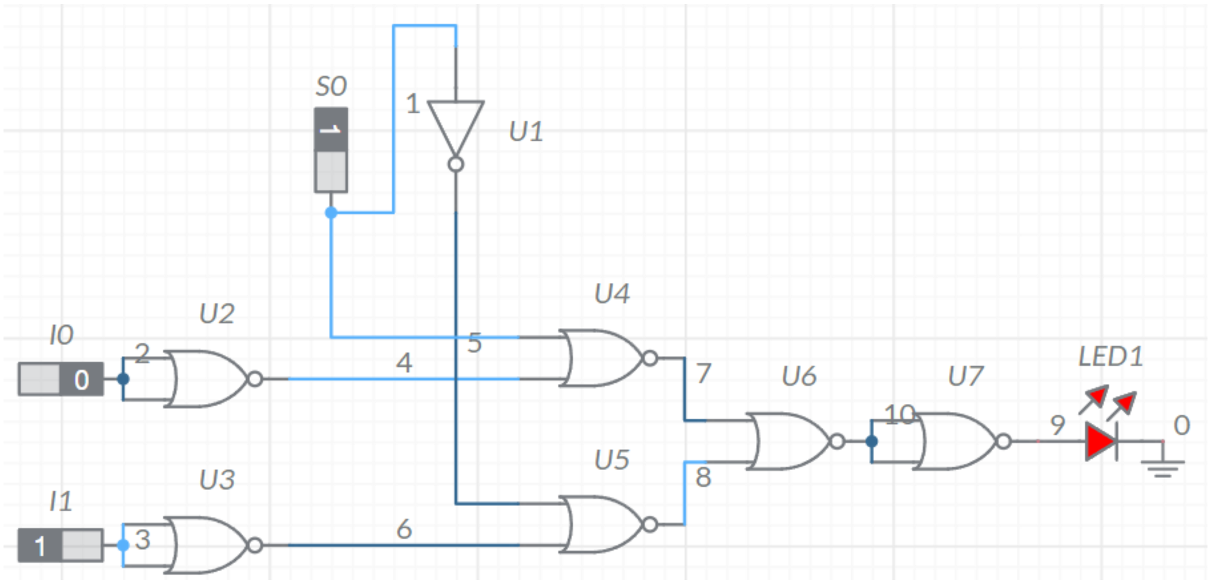
Circuit diagram of 2:1 MUX Circuit using POS Equation --- OAI circuit



Circuit diagram of 2:1 MUX Circuit using SOP Equation --- NAND circuit



Circuit diagram of 2:1 MUX Circuit using POS Equation --- NOR circuit



Implementation steps for Task I and II using 4:1 Multiplexer

	C’D’	C’D	CD’	CD
	I_0	I_1	I_2	I_3
A’B’	1	1	1	1
A’B	1	0	0	0
AB’	0	0	0	1
AB	0	1	0	0
F:	A’	(A^B)’	A’B’	B’
F’:	A	A^B	A+B	B

F implementation:

I_0 = A’

I_1 = (A ^ B)’

I_2 = A’B’

I_3 = B’

F’ implementation:

I_0 = A

I_1 = A ^ B

I_2 = A + B

I_3 = B

Implementation steps for Task I and II using 8:1 Multiplexer

	B’C’D’	B’C’D	B’CD’	B’CD	BC’D’	BC’D	BCD’	BCD
	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A’	1	1	1	1	1	0	0	0
A	0	0	0	1	0	1	0	0
F	A’	A’	A’	1	A’	A	0	0
F’	A	A	A	0	A	A’	1	1

F implementation:

I_0 = I_1 = I_2 = I_4 = A’

I_3 = 1

I_5 = A

I_6 = I_7 = 0

F’ implementation:

$I_0 = I_1 = I_2 = I_4 = A$

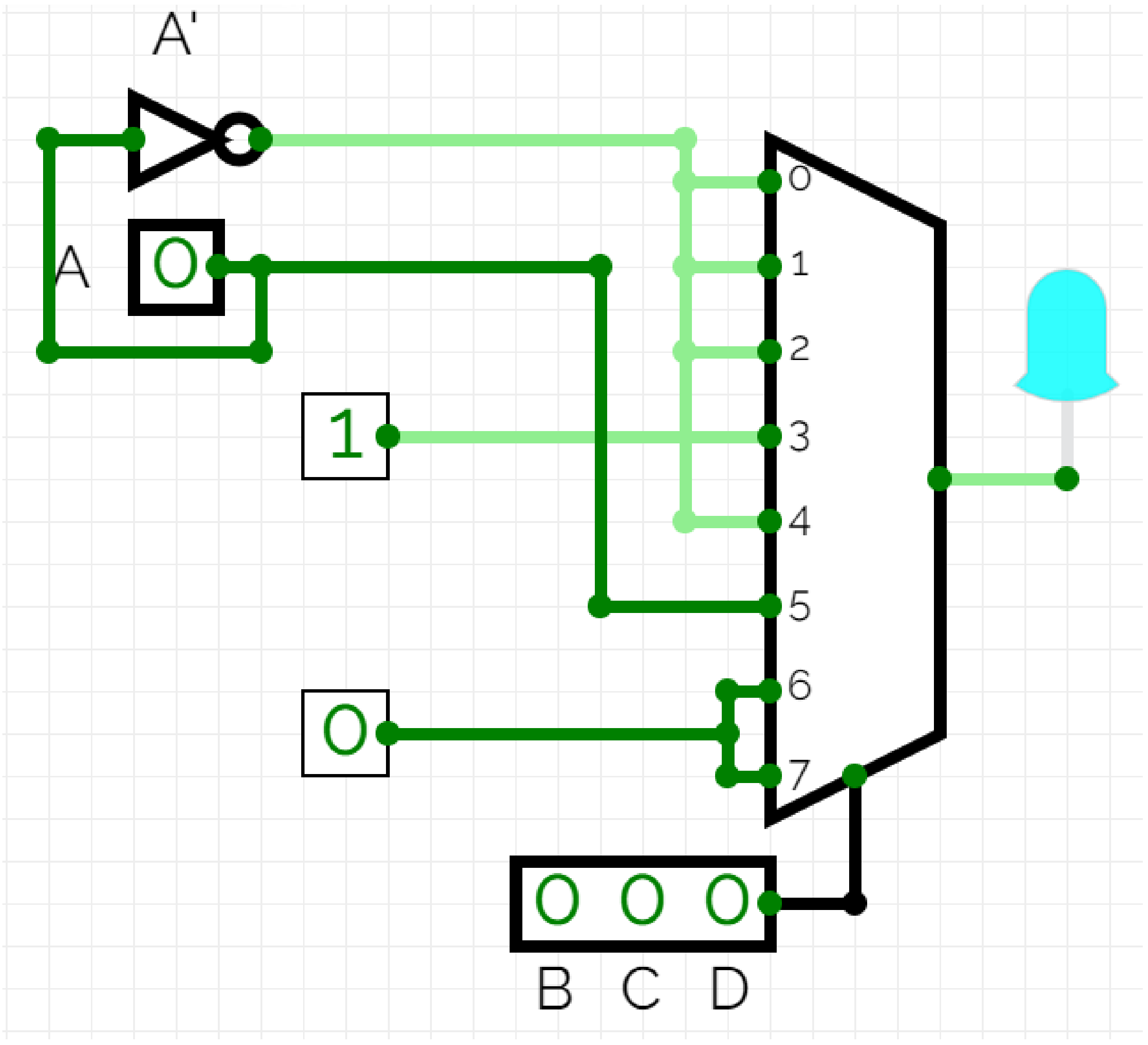
$I_3 = 0$

$I_5 = A'$

$I_6 = I_7 = 1$

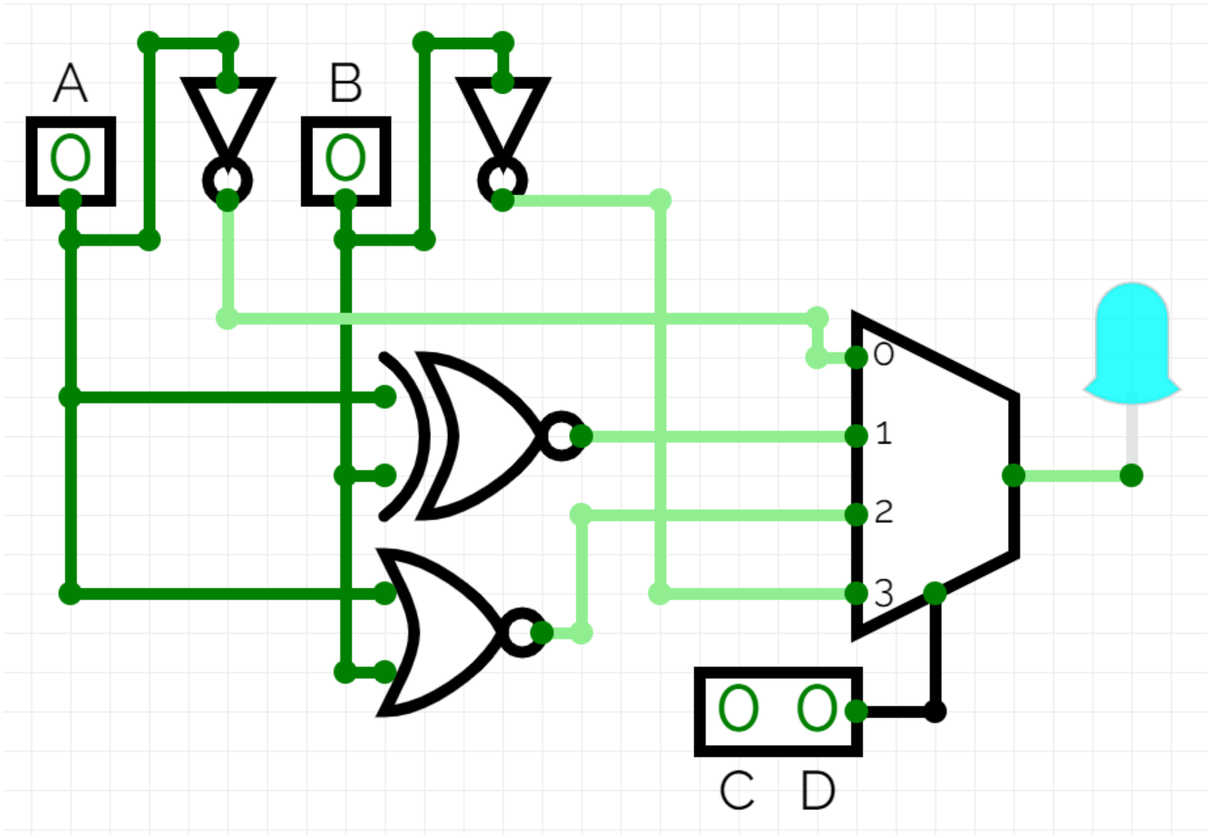
Multisim live / Circuitverse.org Simulation link for Multiplexer based circuit implemented using 8:1 Multiplexer

https://circuitverse.org/simulator/21bds0340-8-1-regno-mux



Multisim live / Circuitverse.org Simulation link for Multiplexer based circuit implemented using 4:1 Multiplexer

https://circuitverse.org/simulator/21bds0340-4-1-regno-mux



Verilog code: SL

```
module mux2to1(x, s, f);
    input [0:1]x;
    input s;
    output f;
    wire nots, prod1, prod2;
    not(nots, s);
    and(prod1, x[0], nots);
    and(prod2, x[1], s);
    or(f, prod1, prod2);
endmodule

module mux4to1(x, s, f);
    input [0:3]x;
    input [0:1]s;
    output f;
    wire [0:1]farr;
    mux2to1 tto1(x[0:1], s[1], farr[0]);
    mux2to1 tto2(x[2:3], s[1], farr[1]);
    mux2to1 tto3(farr, s[0], f);
endmodule

module mux8to1(x, s, f);
    input [0:7]x;
    input [0:2]s;
    output f;
    wire [0:1]farr;
    mux4to1 fto1(x[0:3], s[1:2], farr[0]);
    mux4to1 fto2(x[4:7], s[1:2], farr[1]);
    mux2to1 tto1(farr, s[0], f);
endmodule

module mux16to1(x, s, f);
    input [0:15]x;
    input [0:3]s;
    output f;
    wire [0:1]farr;
    mux8to1 eto1(x[0:7], s[1:3], farr[0]);
    mux8to1 eto2(x[8:15], s[1:3], farr[1]);
    mux2to1 tto1(farr, s[0], f);
endmodule
```

Verilog code: BL

```
module mux2to1(x, s, f);
    input [0:1]x;
    input s;
    output f;
    wire nots, prod1, prod2;
    always @(*);
    begin
        nots = ~s;
        prod1 = x[0] & nots;
        prod2 = x[1] & s;
        f = prod1 | prod2;
    end
endmodule


module mux4to1(x, s, f);
    input [0:3]x;
    input [0:1]s;
    output f;
    wire [0:1]farr;
    mux2to1 tto1(x[0:1], s[1], farr[0]);
    mux2to1 tto2(x[2:3], s[1], farr[1]);
    mux2to1 tto3(farr, s[0], f);
```

```
endmodule

module mux8to1(x, s, f);
    input [0:7]x;
    input [0:2]s;
    output f;
    wire [0:1]farr;
    mux4to1 fto1(x[0:3], s[1:2], farr[0]);
    mux4to1 fto2(x[4:7], s[1:2], farr[1]);
    mux2to1 tto1(farr, s[0], f);
endmodule

module mux16to1(x, s, f);
    input [0:15]x;
    input [0:3]s;
    output f;
    wire [0:1]farr;
    mux8to1 eto1(x[0:7], s[1:3], farr[0]);
    mux8to1 eto2(x[8:15], s[1:3], farr[1]);
    mux2to1 tto1(farr, s[0], f);
endmodule
```

Verilog code: DFL

```
module mux2to1(x, s, f);
    input [0:1]x;
    input s;
    output f;
    wire nots, prod1, prod2;
    assign nots = ~s;
    assign prod1 = x[0] & nots;
    assign prod2 = x[1] & s;
    assign f = prod1 | prod2;
endmodule

module mux4to1(x, s, f);
    input [0:3]x;
    input [0:1]s;
    output f;
    wire [0:1]farr;
    mux2to1 tto1(x[0:1], s[1], farr[0]);
    mux2to1 tto2(x[2:3], s[1], farr[1]);
    mux2to1 tto3(farr, s[0], f);
endmodule

module mux8to1(x, s, f);
    input [0:7]x;
    input [0:2]s;
    output f;
    wire [0:1]farr;
    mux4to1 fto1(x[0:3], s[1:2], farr[0]);
    mux4to1 fto2(x[4:7], s[1:2], farr[1]);
    mux2to1 tto1(farr, s[0], f);
endmodule

module mux16to1(x, s, f);
    input [0:15]x;
    input [0:3]s;
    output f;
    wire [0:1]farr;
    mux8to1 eto1(x[0:7], s[1:3], farr[0]);
    mux8to1 eto2(x[8:15], s[1:3], farr[1]);
    mux2to1 tto1(farr, s[0], f);
endmodule
```

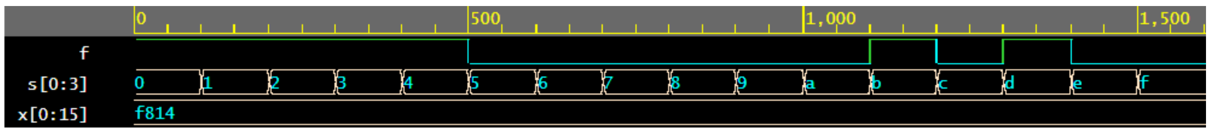
Verilog code: Conditional Operator

```
module mux2to1(x, s, f);
    input [0:1]x;
    input s;
    output f;
    assign f = s ? x[0] : x[1];
endmodule

module mux4to1(x, s, f);
    input [0:3]x;
    input [0:1]s;
    output f;
    wire [0:1]farr;
    mux2to1 tto1(x[0:1], s[1], farr[0]);
    mux2to1 tto2(x[2:3], s[1], farr[1]);
    mux2to1 tto3(farr, s[0], f);
endmodule

module mux8to1(x, s, f);
    input [0:7]x;
    input [0:2]s;
    output f;
    wire [0:1]farr;
    mux4to1 fto1(x[0:3], s[1:2], farr[0]);
    mux4to1 fto2(x[4:7], s[1:2], farr[1]);
    mux2to1 tto1(farr, s[0], f);
endmodule

module mux16to1(x, s, f);
    input [0:15]x;
    input [0:3]s;
    output f;
    wire [0:1]farr;
    mux8to1 eto1(x[0:7], s[1:3], farr[0]);
    mux8to1 eto2(x[8:15], s[1:3], farr[1]);
    mux2to1 tto1(farr, s[0], f);
endmodule
```

Test Bench

```
module testbench;
  reg [0:15]x;
  reg [0:3]s;
  wire f;
  mux16to1 sto1(x, s, f);
  initial begin
    x[0] = 1; x[1] = 1; x[2] = 1; x[3] = 1; x[4] = 1; x[5] = 0; x[6] = 0; x[7]
= 0; x[8] = 0; x[9] = 0; x[10] = 0; x[11] = 1; x[12] = 0; x[13] = 1; x[14] =
0; x[15] = 0;
    s[0] = 0; s[1] = 0; s[2] = 0; s[3] = 0;
    #100
    s[0] = 0; s[1] = 0; s[2] = 0; s[3] = 1;
    #100
    s[0] = 0; s[1] = 0; s[2] = 1; s[3] = 0;
    #100
    s[0] = 0; s[1] = 0; s[2] = 1; s[3] = 1;
    #100
    s[0] = 0; s[1] = 1; s[2] = 0; s[3] = 0;
    #100
    s[0] = 0; s[1] = 1; s[2] = 0; s[3] = 1;
    #100
    s[0] = 0; s[1] = 1; s[2] = 1; s[3] = 0;
    #100
    s[0] = 0; s[1] = 1; s[2] = 1; s[3] = 1;
    #100
    s[0] = 1; s[1] = 0; s[2] = 0; s[3] = 0;
    #100
    s[0] = 1; s[1] = 0; s[2] = 0; s[3] = 1;
    #100
    s[0] = 1; s[1] = 0; s[2] = 1; s[3] = 0;
    #100
    s[0] = 1; s[1] = 0; s[2] = 1; s[3] = 1;
    #100
    s[0] = 1; s[1] = 1; s[2] = 0; s[3] = 0;
    #100
    s[0] = 1; s[1] = 1; s[2] = 0; s[3] = 1;
    #100
    s[0] = 1; s[1] = 1; s[2] = 1; s[3] = 0;
    #100
    s[0] = 1; s[1] = 1; s[2] = 1; s[3] = 1;
    #100
    s[0] = 0;
  end
endmodule
```

Snip of Output waveform with respective code



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Online Verilog code Simulation links

https://www.edaplayground.com/x/Mwqd

Result and Inference

The output of the function matches the digits in hexadecimal of my registration number.

PART II - Demultiplexer / Decoder based Design

Aim

Using Reg.no. formulate expressions in SOP and POS for F and F '. Implement the circuits using only

a. Design 4 to 16 Decoder using 3 to 8 Decoder constructed using 2-4 Decoders.
b. Give the internal circuit of 2 to 4 Decoder using SOP, POS, NAND, NOR logic design.
c. Implement a Combinational logic circuit obtained from your registration number using Decoder.
d. Write the Verilog code for 4:16,3:8 and 2:4 Decoders and Verify the results using the truth table and show the output waveform.
e. Implement the Verilog code in ModelSim and verify the results using the truth table and show the output waveform.
f. Show the steps and procedure for implementation in the tool used.

Components Required

a. AND, OR, NOT, NAND and NOR gates
b. 5V voltage source
c. Led indicator

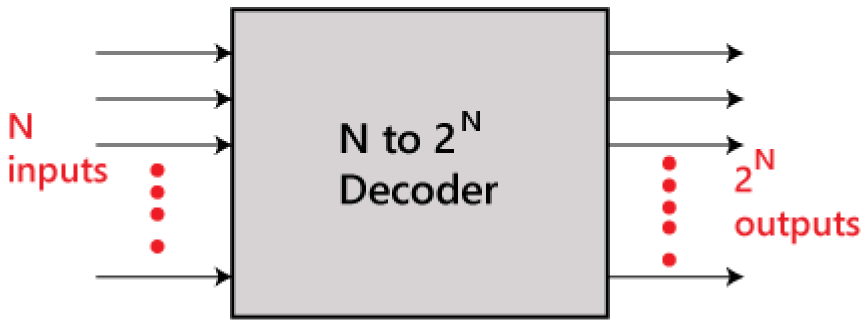
Tools Required

a. Multisim simulator

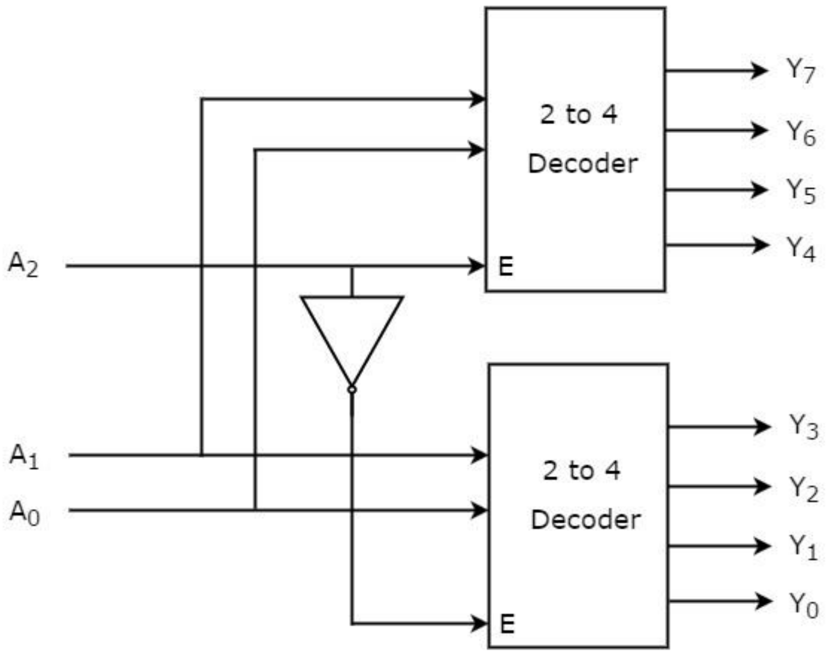
Procedure

1. Decode registration number with a 4:16 decoder.

Pin Diagram



Block diagram for implementing 3:8 Decoder using 2:4 Decoder



Decoder theory

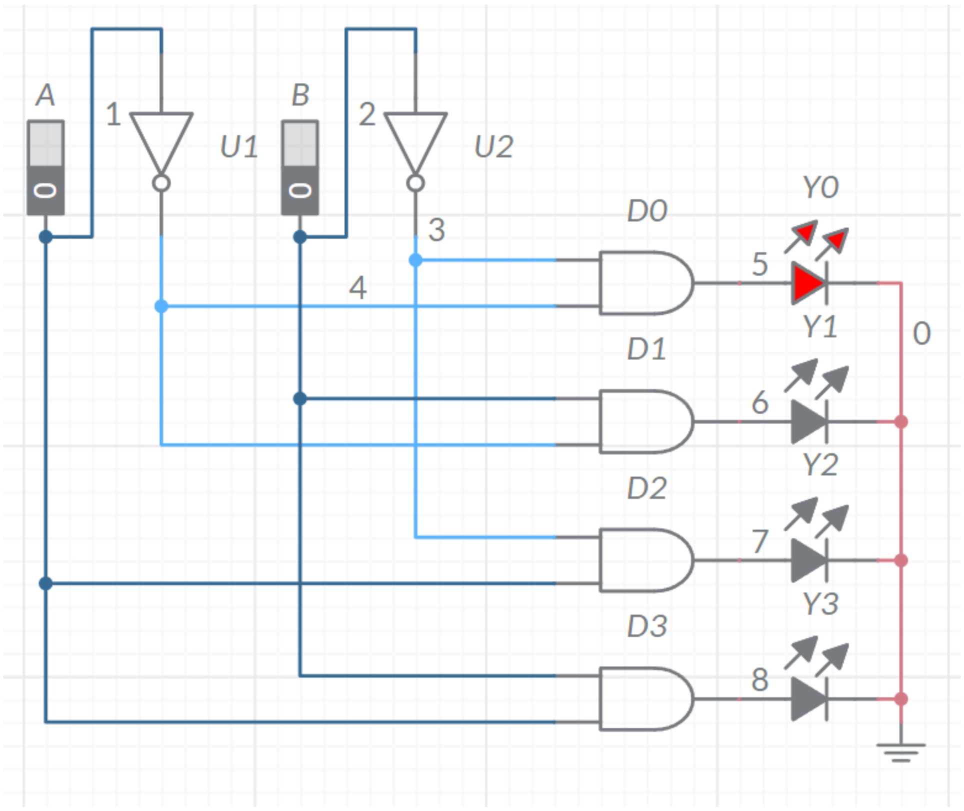
A decoder can take the form of a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different e.g. n-to-2^n, binary-coded decimal decoders.

Encoder and Priority Encoder Theory

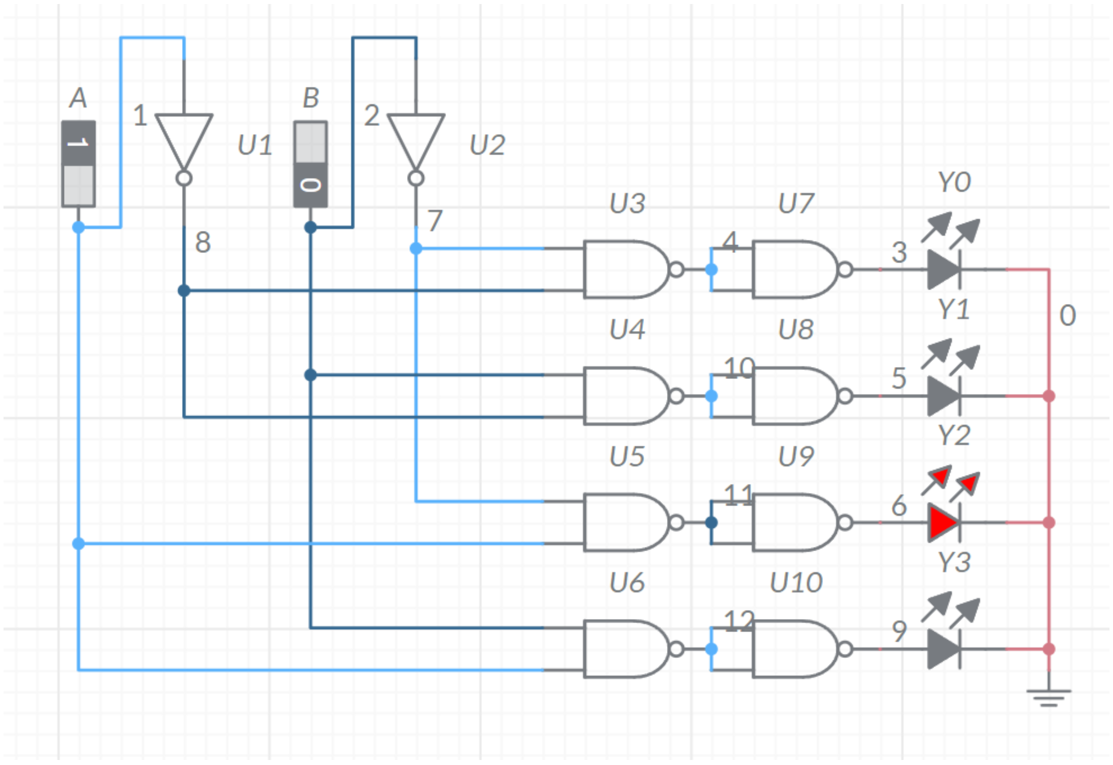
An Encoder is a combinational circuit that performs the reverse operation of decoder. It has maximum of 2^n input lines and ‘n’ output lines, hence it encodes the information from 2^n inputs into an n-bit code. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with ‘n’ bits.

A priority encoder is a circuit or algorithm that compresses multiple binary inputs into a smaller number of outputs. The output of a priority encoder is the binary representation of the index of the most significant activated line, starting from zero.

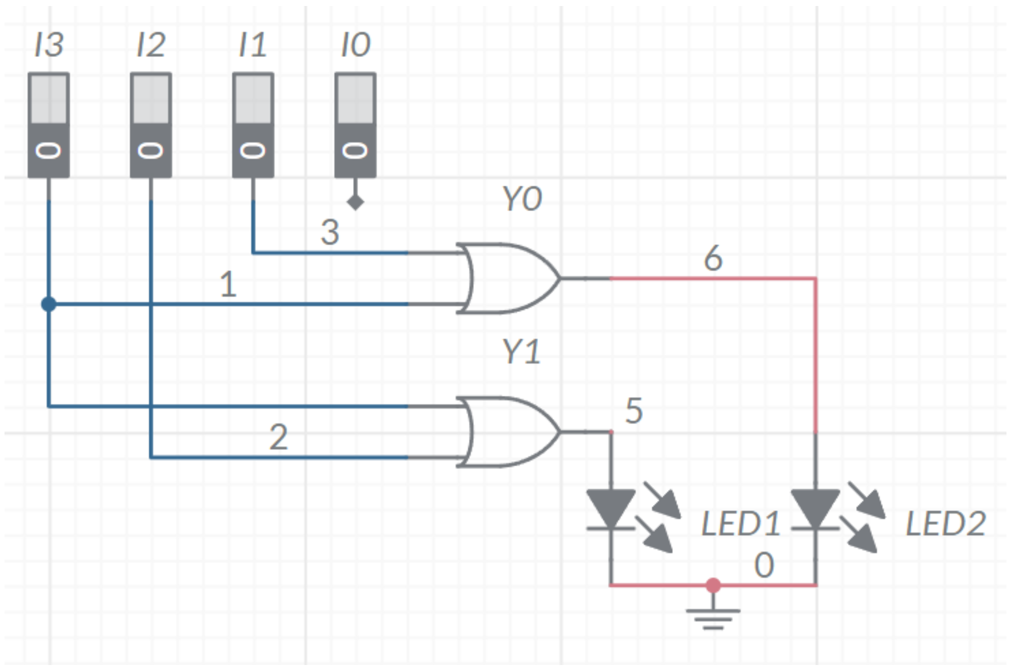
AOI logic circuit of 2:4 Decoder



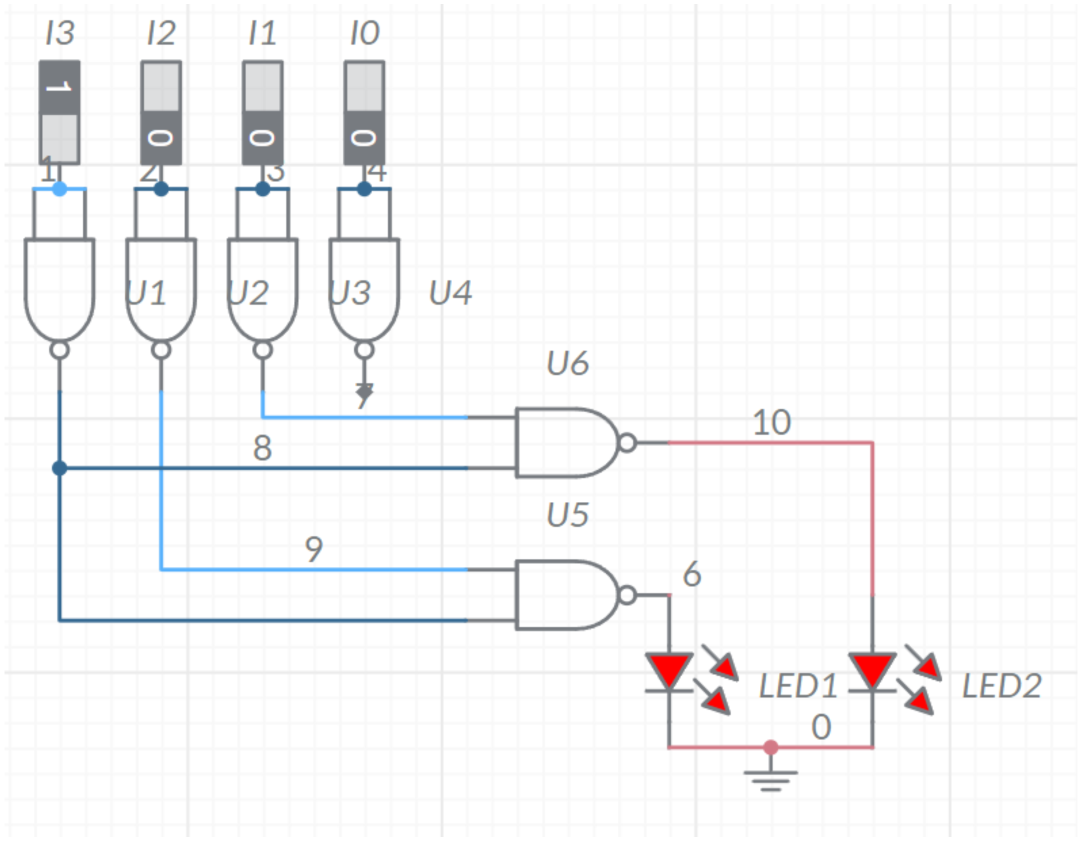
NAND logic circuit of 2:4 Decoder



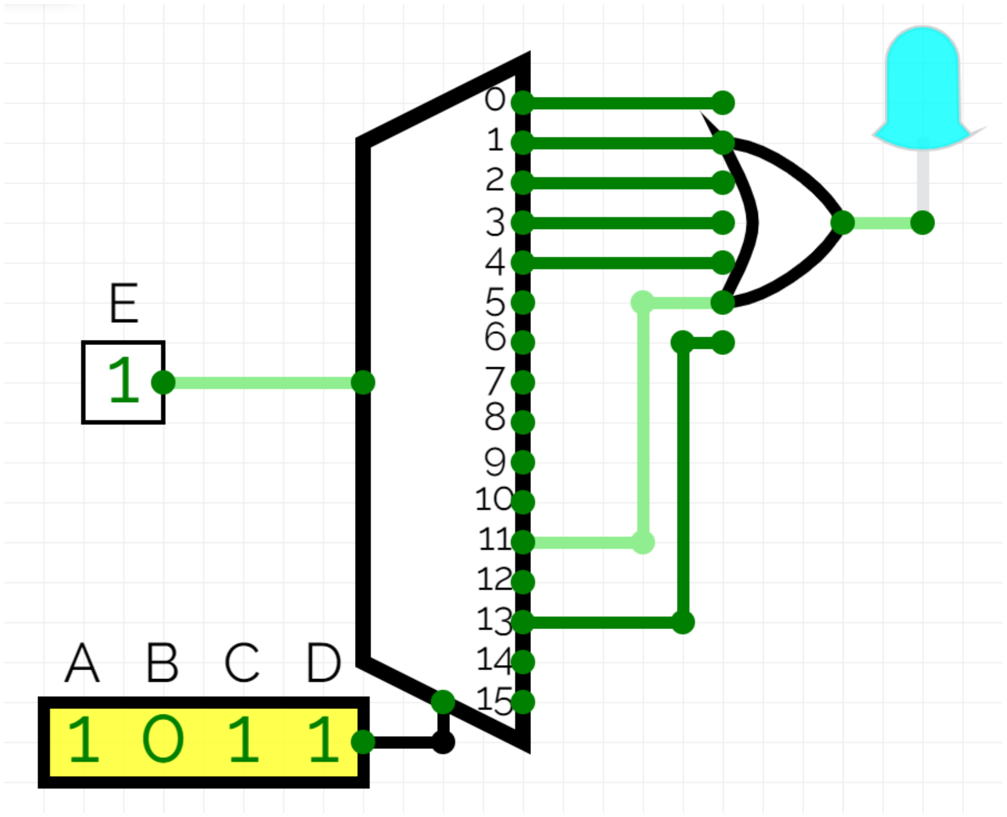
AOI logic circuit of 4:2 encoder



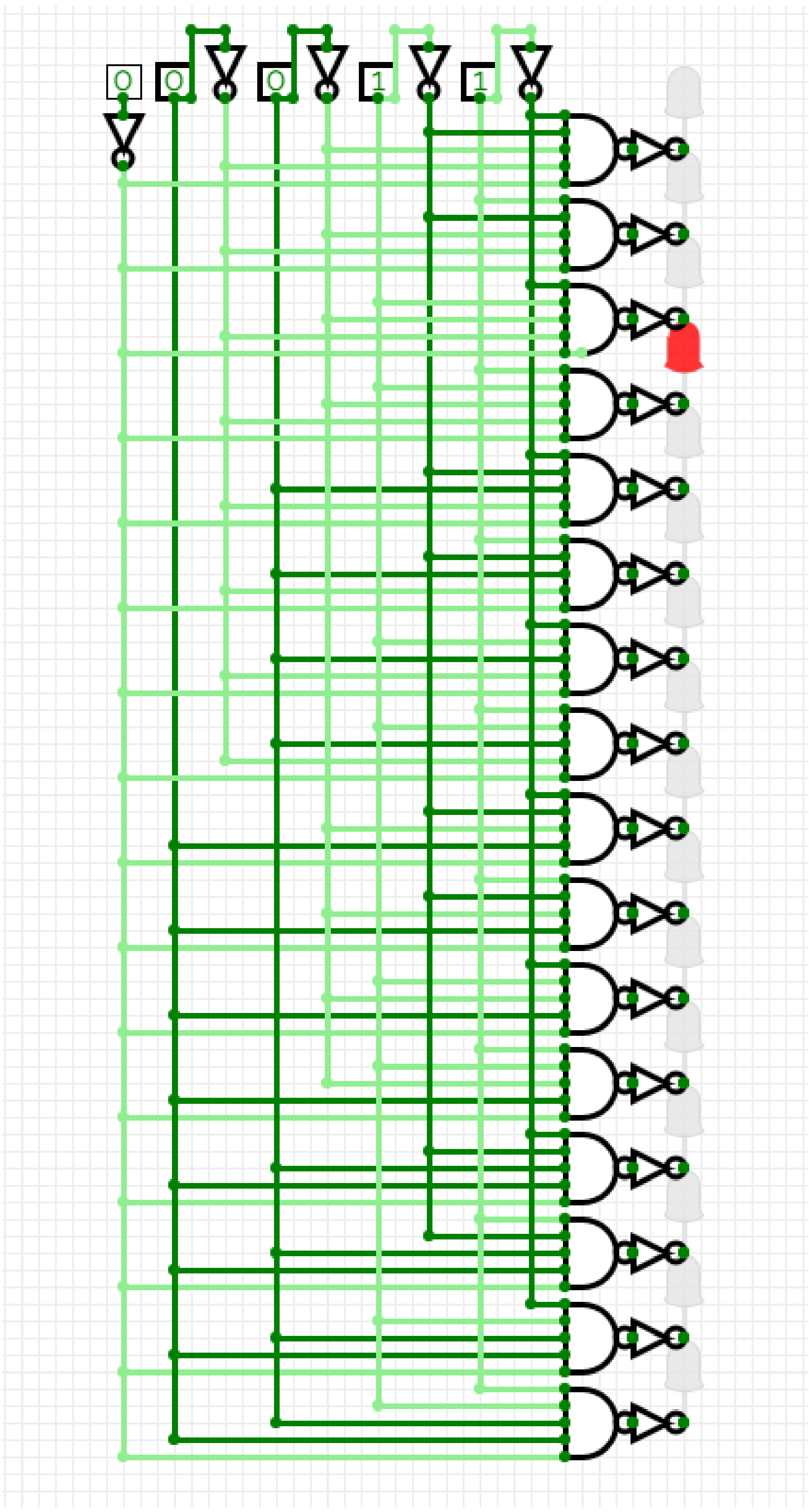
NAND logic circuit of 4:2 encoder



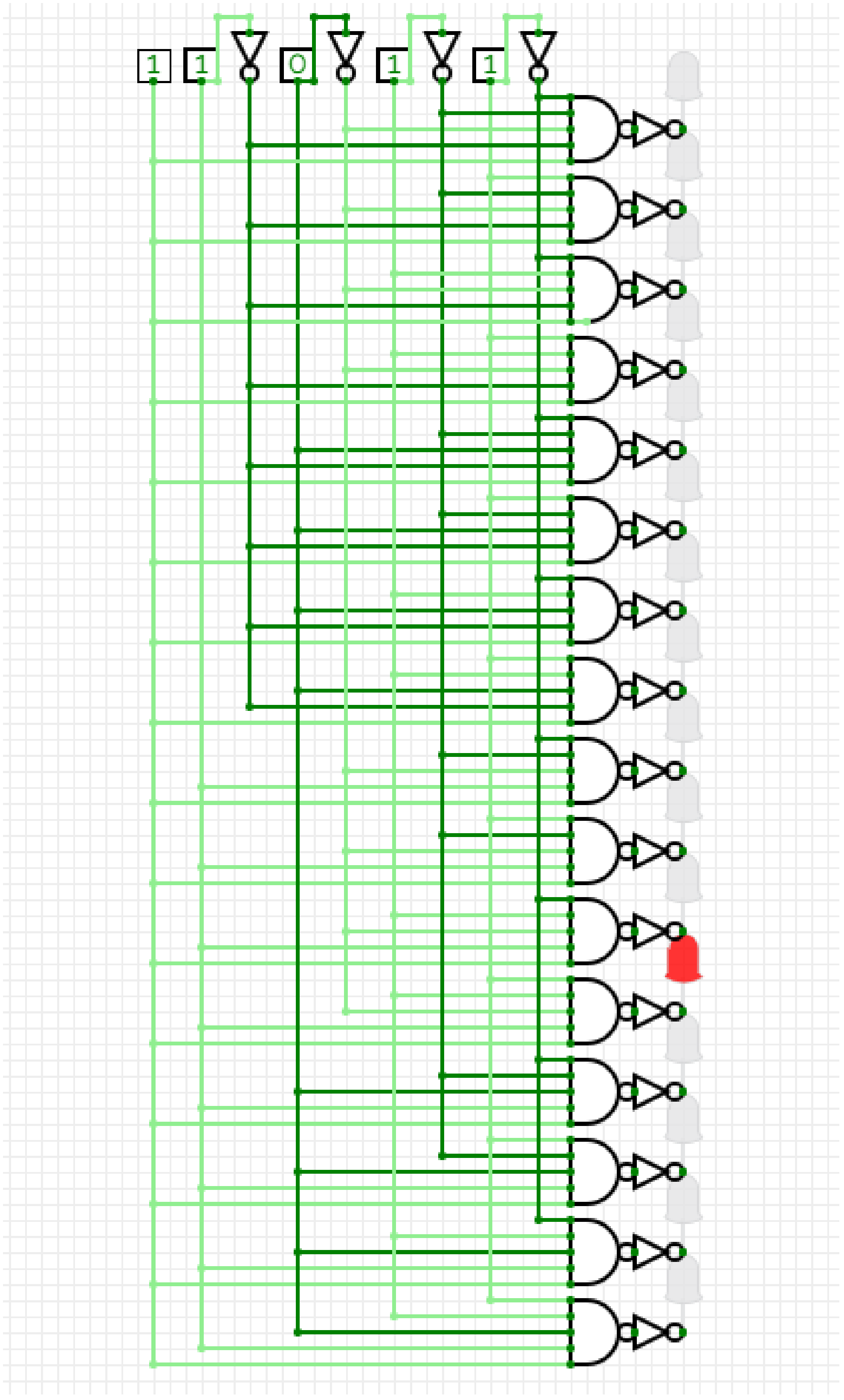
<u>Implementation steps for Task I using Decoder</u>



Proof of Task I (SOP) implemented using Decoder Active Low logic in simulator using snipping tool



Proof of Task I (SOP) implemented using Decoder Active High logic in simulator using snipping tool



Verilog code for 2:4 Decoder (BL, DFL)

```
module decoder2to4(e, x, f);
    input e;
    input [0:1]x;
    output [0:3]f;
    assign f[0] = ~(~e & ~x[0] & ~x[1]);
    assign f[1] = ~(~e & ~x[0] & x[1]);
    assign f[2] = ~(~e & x[0] & ~x[1]);
    assign f[3] = ~(~e & x[0] & x[1]);
endmodule
```

Verilog code of 3:8 Decoder (BL, DFL)

```
module decoder3to8(e, x, f);
    input e;
    input [0:2]x;
    output [0:7]f;
    assign f[0] = ~(~e & ~x[0] & ~x[1] & ~x[2]);
    assign f[1] = ~(~e & ~x[0] & ~x[1] & x[2]);
    assign f[2] = ~(~e & ~x[0] & x[1] & ~x[2]);
    assign f[3] = ~(~e & ~x[0] & x[1] & x[2]);
    assign f[4] = ~(~e & x[0] & ~x[1] & ~x[2]);
    assign f[5] = ~(~e & x[0] & ~x[1] & x[2]);
    assign f[6] = ~(~e & x[0] & x[1] & ~x[2]);
    assign f[7] = ~(~e & x[0] & x[1] & x[2]);
endmodule
```

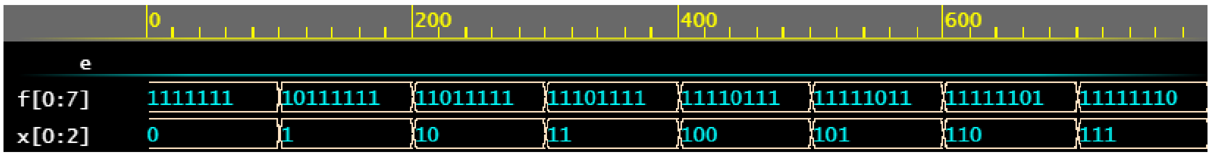
Verilog code for 2:4 and 3:8 Decoder (SL)

```
module decoder2to4(e, x, f);
    input e;
    input [0:1]x;
    output [0:3]f;
    assign f[0] = ~(~e & ~x[0] & ~x[1]);
    assign f[1] = ~(~e & ~x[0] & x[1]);
    assign f[2] = ~(~e & x[0] & ~x[1]);
    assign f[3] = ~(~e & x[0] & x[1]);
endmodule

module decoder3to8(e, x, f);
    input e;
    input [0:2]x;
    output [0:7]f;
    decoder2to4 d2t41(x[0], x[1:2], f[0:3]);
    decoder2to4 d2t42(~x[0], x[1:2], f[4:7]);
endmodule
```

Verilog Test Bench of Decoder

```
module testdecoder;
    reg e;
    reg [0:2]x;
    wire [0:7]f;
    decoder3to8 d1(e, x, f);
    initial begin
        e = 0;
        x = 3'b000;
        #100
        x = 3'b001;
        #100
        x = 3'b010;
        #100
        x = 3'b011;
        #100
        x = 3'b100;
        #100
        x = 3'b101;
        #100
        x = 3'b110;
        #100
        x = 3'b111;
        #100
        x = 2'b00;
    end
endmodule
```

Snip of Verilog code output with links



```
e
f[0:7]  1111111  10111111  11011111  11101111  11110111  11111011  11111101  11111110
x[0:2]  0        1         10        11        100       101       110       111
```

Note: To revert to EPWave opening in a new browser window, set that option on your user page.

https://www.edaplayground.com/x/kG_r

Result and Inference

The circuit above has been created to decode any input with an active low input enabled.