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 Equation --- NAND circuit | 10 | ✓ |  |
| 16 | Circuit diagram of 2:1 MUX Circuit using POS Equation --- NOR circuit | 10 | ✓ |  |
| 17 | Implementation steps for Task I and II using 4:1 Multiplexer | 11 | ✓ |  |
| 18 | Implementation steps for Task I and II using 8:1 Multiplexer | 11 | ✓ |  |
| 19 | Multisim live / Circuitverse.org Simulation link for Multiplexer based circuit implemented using 8:1 Multiplexer | 12 | ✓ |  |
| 20 | Multisim live / Circuitverse.org Simulation link for Multiplexer based circuit implemented using 4:1 Multiplexer | 13 | ✓ |  |
| 21 | Verilog code: SL | 13 | ✓ |  |
| 22 | Verilog code: BL | 14 | ✓ |  |
| 23 | Verilog code: DFL | 15 | ✓ |  |
| 24 | Verilog code: Conditional Operator | 16 | ✓ |  |
| 25 | Test Bench | 17 | ✓ |  |
| 26 | Snip of Output waveform with respective code | 18 | ✓ |  |
| 27 | Online Verilog code Simulation links | 18 | ✓ |  |
| 28 | Result | 18 | ✓ |  |
| 29 | Inference | 18 | ✓ |  |
|  | PART II - Demultiplexer / Decoder based Design |  |  |  |
| 1 | Aim | 19 | ✓ |  |
| 2 | Components Required and Tools Required | 19 | ✓ |  |
| 3 | Procedure | 19 | ✓ |  |
| 4 | Pin diagram | 19 | ✓ |  |
| 5 | Truth Table and Boolean expression of Encoder (4:2, 8:3), Decoder (2:4, 3:8) with active low and active high logic. | 19 | ✓ |  |
| 6 | Block diagram for implementing 3:8 Decoder using 2:4 Decoder | ✓ | 20 |  |
| 7 | Block diagram for implementing 2: 4 Decoder using 1:2 Decoder | ✓ | 20 |  |
| 8 | Decoder theory | ✓ | 20 |  |
| 9 | Encoder and Priority Encoder Theory | ✓ | 20 |  |
| 10 | AOI logic circuit of 2:4 Decoder | ✓ | 21 |  |
| 11 | NAND logic circuit of 2:4 Decoder | ✓ | 21 |  |
| 12 | AOI logic circuit of 4:2 encoder | ✓ | 22 |  |
| 13 | NAND logic circuit of 4:2 encoder | ✓ | 22 |  |
| 14 | Implementation steps for Task I & II using Decoder | ✓ | 23 |  |
| 15 | Proof of Task I (SOP) implemented using Decoder Active Low logic in simulator using snipping tool | ✓ | 24 |  |
| 16 | Proof of Task I (SOP) implemented using Decoder Active High logic in simulator using snipping tool | ✓ | 25 |  |
| 17 | Proof of Task I (POS) implemented using Decoder Active Low logic in simulator using snipping tool | ✓ | 24 |  |
| 18 | Proof of Task I (POS) implemented using Decoder Active High logic in simulator using snipping tool | ✓ | 25 |  |
| 19 | Verilog code for 2:4 Decoder (BL, DFL) | ✓ | 26 |  |
| 20 | Verilog code of 3:8 Decoder  (BL, DFL) | ✓ | 26 |  |
| 21 | Verilog code for 2:4 and 3:8 Decoder (SL) | ✓ | 26 |  |
| 22 | Verilog Test Bench of Decoder | ✓ | 27 |  |
| 23 | Snip of Verilog code output with links | ✓ | 27 |  |
| 24 | Result | ✓ | 27 |  |
| 25 | Inference | ✓ | 27 |  |

**Part I – Multiplexer Based Design**

**Aim**

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

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

**Components Required**

1. AND, OR, NOT, NAND and NOR gates
2. 5V voltage source
3. Led indicator

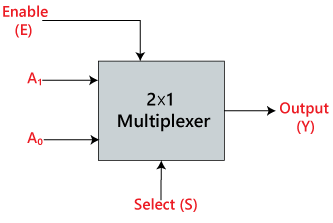
**Tools Required**

1. 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. **Diagram, schematic

   Description automatically generated**Simplify registration number using implementation table and represent with mux

**Pin Diagram**

**Diagram

Description automatically generatedDiagram, schematic

Description automatically generated**

**Truth Table and Boolean expression of Multiplexer, Demultiplexer**

Multiplexer:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Binary Equivalent | S0 | S1 | S2 | S3 | Y |
| 0 | 0 | 0 | 0 | 0 | A0 |
| 1 | 0 | 0 | 0 | 1 | A1 |
| 2 | 0 | 0 | 1 | 0 | A2 |
| 3 | 0 | 0 | 1 | 1 | A3 |
| 4 | 0 | 1 | 0 | 0 | A4 |
| 5 | 0 | 1 | 0 | 1 | A5 |
| 6 | 0 | 1 | 1 | 0 | A6 |
| 7 | 0 | 1 | 1 | 1 | A7 |
| 8 | 1 | 0 | 0 | 0 | A8 |
| 9 | 1 | 0 | 0 | 1 | A9 |
| 10 | 1 | 0 | 1 | 0 | A10 |
| 11 | 1 | 0 | 1 | 1 | A11 |
| 12 | 1 | 1 | 0 | 0 | A12 |
| 13 | 1 | 1 | 0 | 1 | A13 |
| 14 | 1 | 1 | 1 | 0 | A14 |
| 15 | 1 | 1 | 1 | 1 | A15 |

Y = A0.S0’S1’S2’S3’ + A1.S0’S1’S2’S3 + A2.S0’S1’S2S3’ + A3.S0’S1’S2S3 + A4.S0’S1S2’S3’ + A5.S0’S1S2’S3 + A6.S0’S1S2S3’ + A7.S0’S1S2S3 + A8.S0S1’S2’S3’ + A9.S0S1’S2’S3 + A10.S0S1’S2S3’ + A11.S0S1’S2S3 + A12.S0S1S2’S3’ + A13.S0S1S2’S3 + A14.S0S1S2S3’ + A15.S0S1S2S3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S0 | S1 | S2 | S3 | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 |
| 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 |

A0 = S0’S1’S2’S3’

A1 = S0’S1’S2’S3

A2 = S0’S1’S2S3’

A3 = S0’S1’S2S3

A4 = S0’S1S2’S3’

A5 = S0’S1S2’S3

A6 = S0’S1S2S3’

A7 = S0’S1S2S3

A8 = S0S1’S2’S3’

A9 = S0S1’S2’S3

A10 = S0S1’S2S3’

A11 = S0S1’S2S3

A12 = S0S1S2’S3’

A13 = S0S1S2’S3

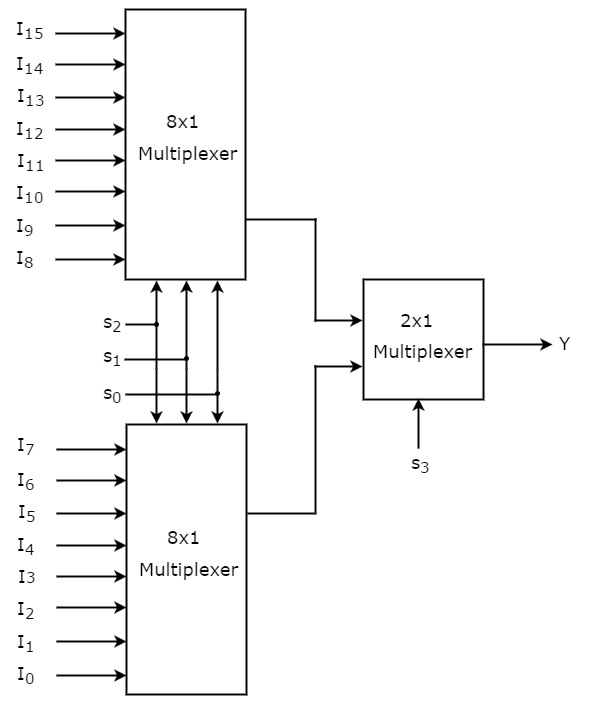
A14 = S0S1S2S3’

A15 = S0S1S2S3

**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**

**Diagram, schematic

Description automatically generatedBlock diagram and Circuit diagram for implementing 8:1 MUX using 4:1 MUX**

**Chart, diagram

Description automatically generated**

**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 = S0’S1’S2’S3’ + S0’S1’S2’S3 + S0’S1’S2S3’ + S0’S1’S2S3 + S0’S1S2’S3’ + A11.S0S1’S2S3 + A13.S0S1S2’S3

F’ = S0’S1S2’S3 + S0’S1S2S3’ + S0’S1S2S3 + S0S1’S2’S3’ + S0S1’S2’S3 + S0S1’S2S3’+ S0S1S2’S3’ + S0S1S2S3’ + S0S1S2S3

**POS canonical expression of MUX and circuit implemented in MUX**

F = (S0+S1+S2+S3) (S0+S1+S2+S3’) (S0+S1+S2’+S3) (S0+S1+S2’+S3’) (S0+S1’+S2+S3) (S0’+S1+S2’+S3’) (S0’+S1’+S2+S3’)

F’ = (S0+S1’+S2+S3’) (S0+S1’+S2’+S3) (S0+S1’+S2’+S3’) (S0’+S1+S2+S3) (S0’+S1+S2+S3’) (S0’+S1+S2’+S3) (S0’+S1’+S2+S3) (S0’+S1’+S2’+S3) (S0’+S1’+S2’+S3’)

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

**Diagram

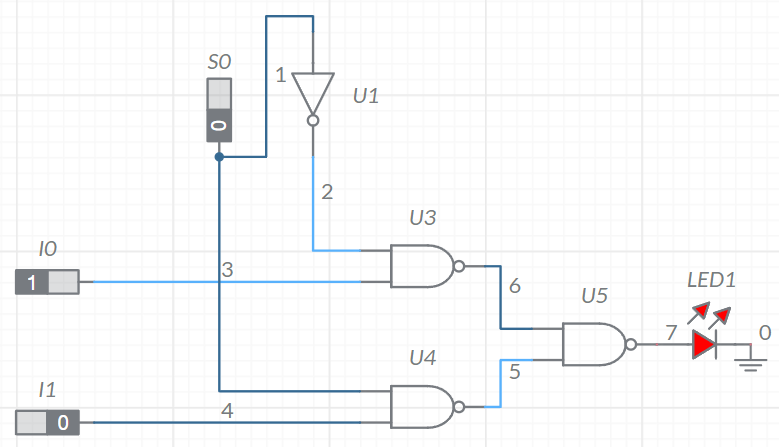
Description automatically generated**

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

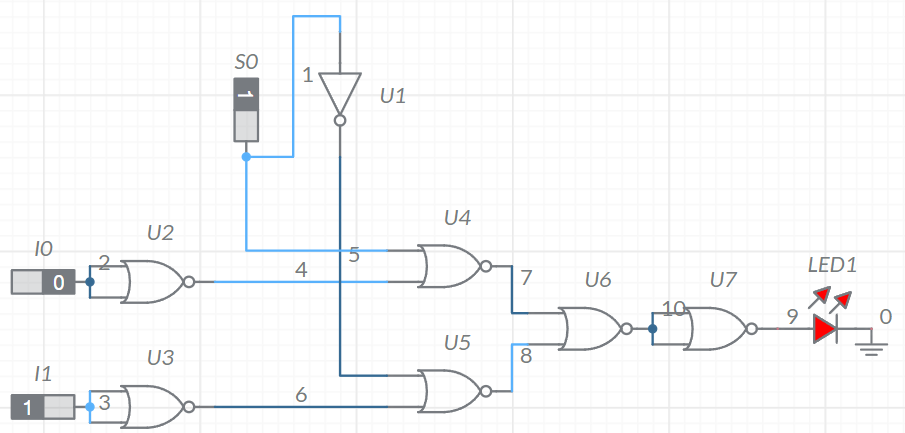
Diagram

Description automatically generated

**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 |
|  | I0 | I1 | I2 | I3 |
| 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:

I0 = A’

I1 = (A ^ B)’

I2 = A’B’

I3 = B’

F’ implementation:

I0 = A

I1 = A ^ B

I2 = A + B

I3 = 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 |
|  | I0 | I1 | I2 | I3 | I4 | I5 | I6 | I7 |
| 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:

I0 = I1 = I2 = I4 = A’

I3 = 1

I­5 = A

I6 = I7 = 0

F’ implementation:

I0 = I1 = I2 = I4 = A

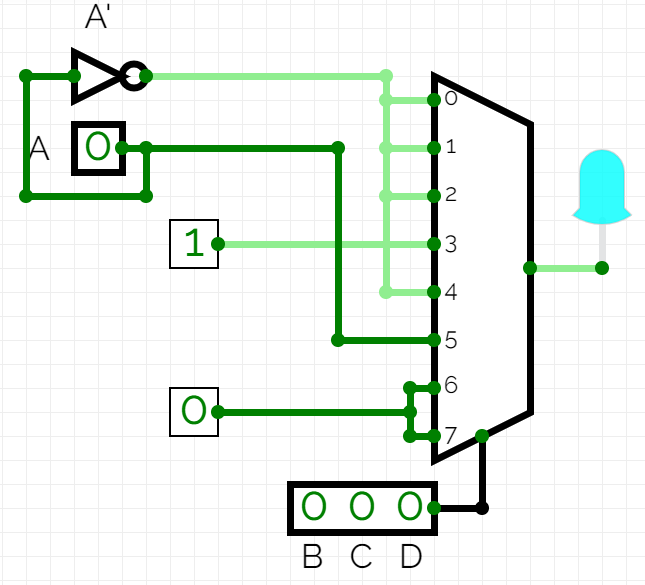
I3 = 0

I­5 = A’

I6 = I7 = 1

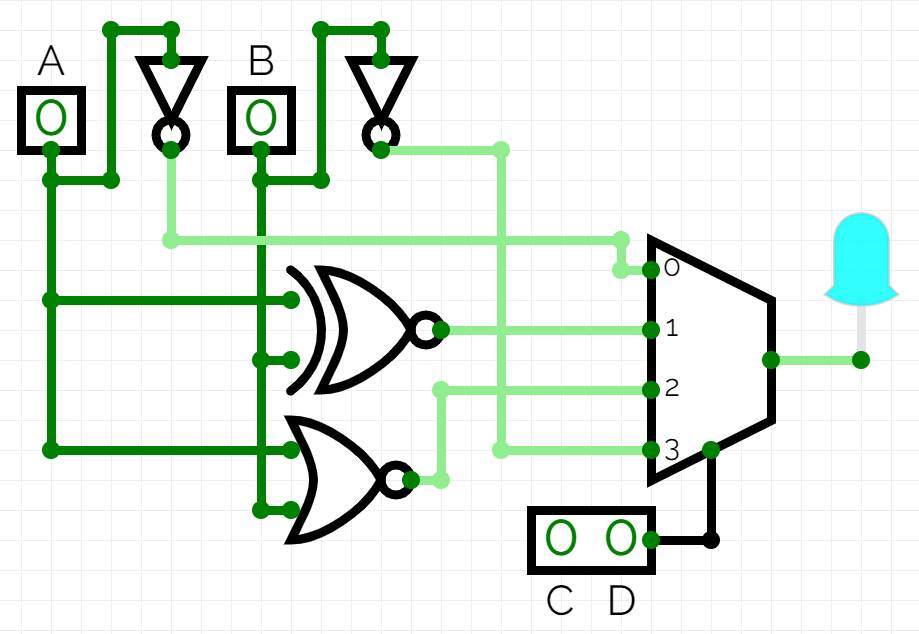
**Multisim live / Circuitverse.org Simulation link for Multiplexer based circuit implemented using 8:1 Multiplexer**

<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

Graphical user interface, text, application

Description automatically generated**Snip of Output waveform with respective code**

**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

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

**Components Required**

1. AND, OR, NOT, NAND and NOR gates
2. 5V voltage source
3. Led indicator

**Tools Required**

1. Multisim simulator

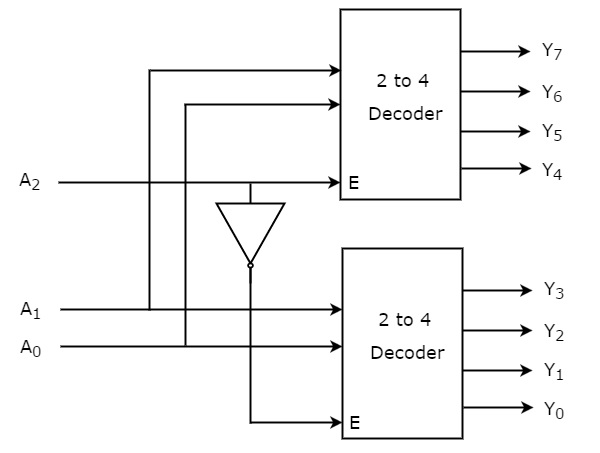
**Procedure**

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

**Pin Diagram**

Text, whiteboard

Description automatically generated

**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-2n, binary-coded decimal decoders.

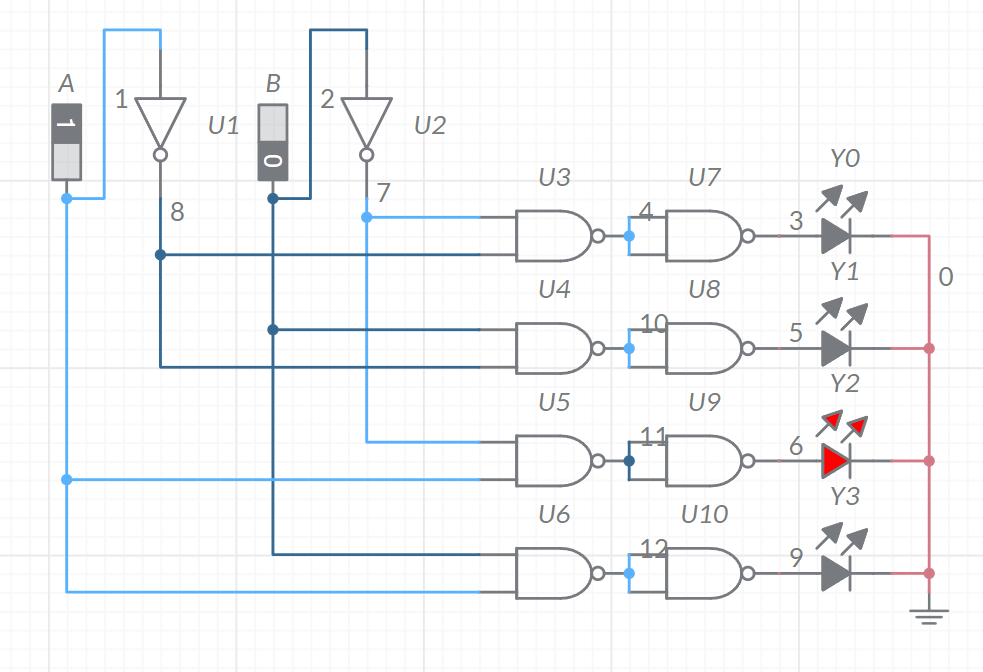
**Encoder and Priority Encoder Theory**

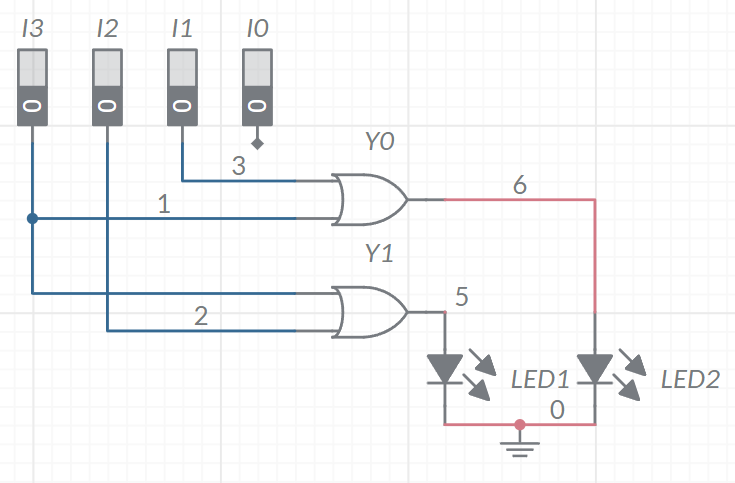
An Encoder is a combinational circuit that performs the reverse operation of decoder. It has maximum of 2^n input lines and ‘n’ output lines, hence it encodes the information from 2^n inputs into an n-bit code. It will produce a binary code equivalent to the input, which is active High. Therefore, the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

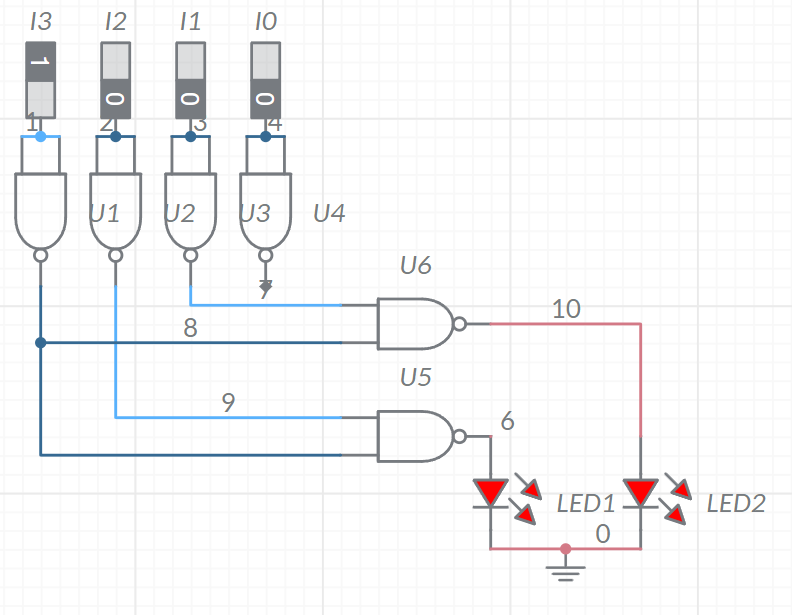
A priority encoder is a circuit or algorithm that compresses multiple binary inputs into a smaller number of outputs. The output of a priority encoder is the binary representation of the index of the most significant activated line, starting from zero.

Diagram

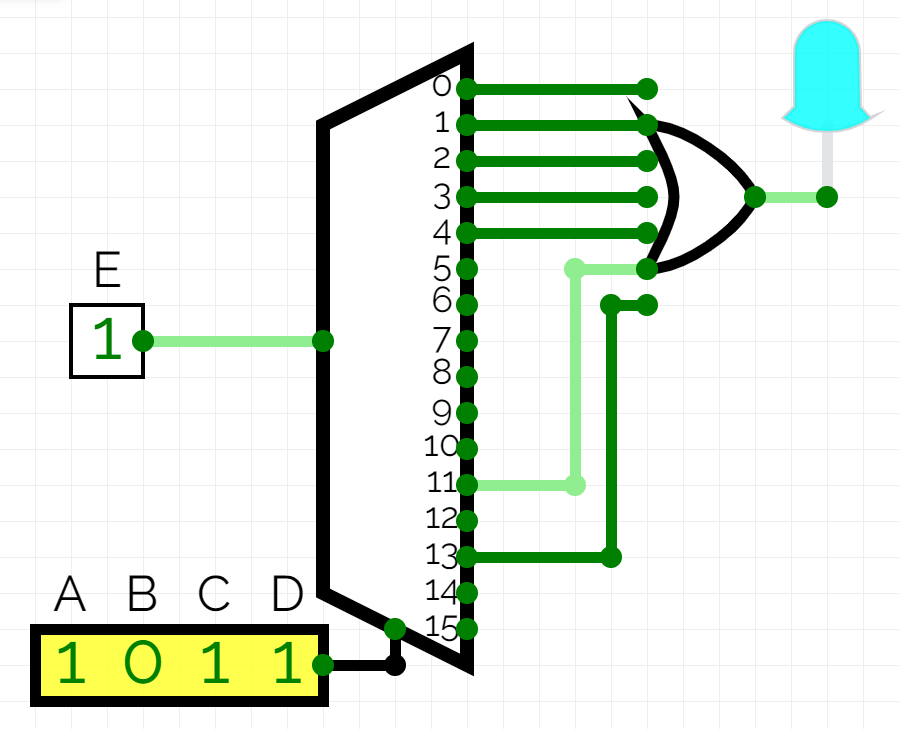
Description automatically generated**AOI logic circuit of 2:4 Decoder**

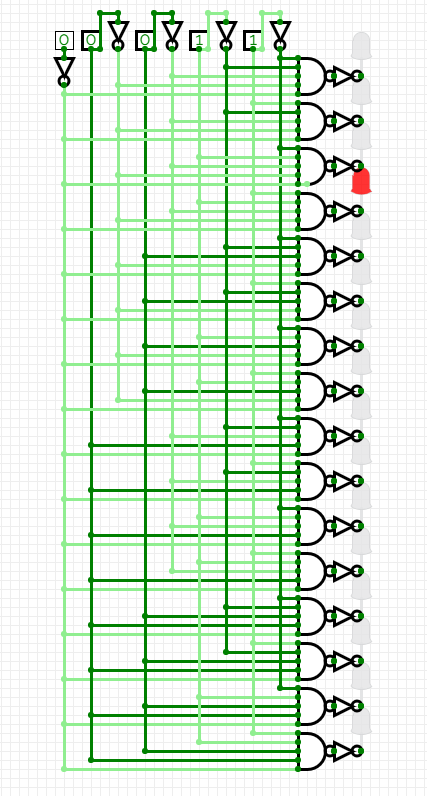
**NAND logic circuit of 2:4 Decoder**

**AOI logic circuit of 4:2 encoder**

**NAND logic circuit of 4:2 encoder**

**Implementation steps for Task I using Decoder**



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

A close-up of a circuit board

Description automatically generated with low confidence**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;

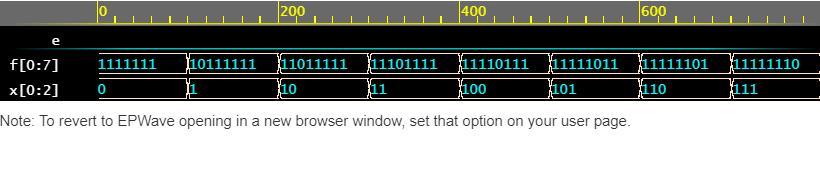
        #100

        x = 2'b00;

    end

endmodule

**Snip of Verilog code output with links**



<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.