DSLab. 10 Magnitude Comparator, Decoder and Multiplexer

Lab. 10 Magnitude Comparator, Decoder & Multiplexer

- Design and verify the following circuits using Verilog HDL
  - Magnitude Comparator
  - Decoder
  - Multiplexer
- Verilog
  - Behavioral level modeling
  - Dataflow modeling
  - Structural level (Gate-level) modeling
- Please write and upload the lab report (Lab10) -- Due on 2022/11/11 23:59

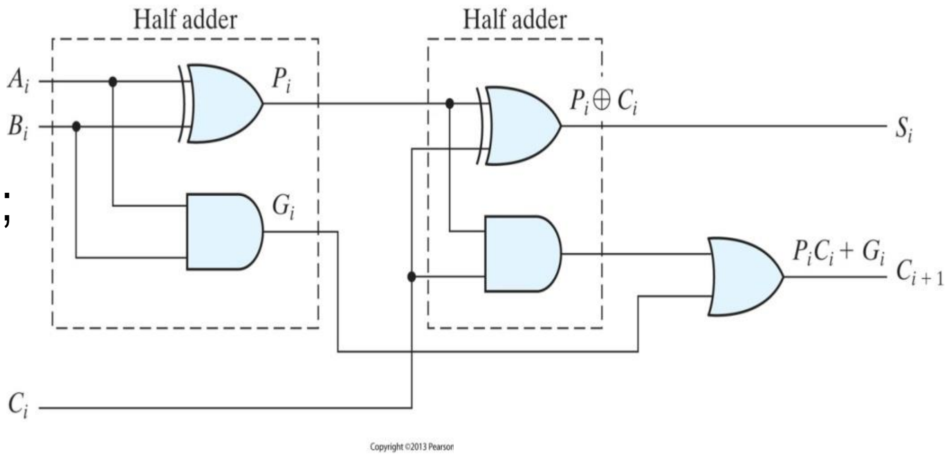
Dataflow Description of 4-bit Adder

```
// Dataflow description of 4-bit adder
module adder_4_bit_df (
    output [3:0]        Sum,
    output              C_out,
    input [3: 0]        A, B,
    input C_in
    );
     assign {C_out, Sum} = A + B + C_in;
endmodule
```

Gate-level Description of 4-bit Ripple-Carry Adder

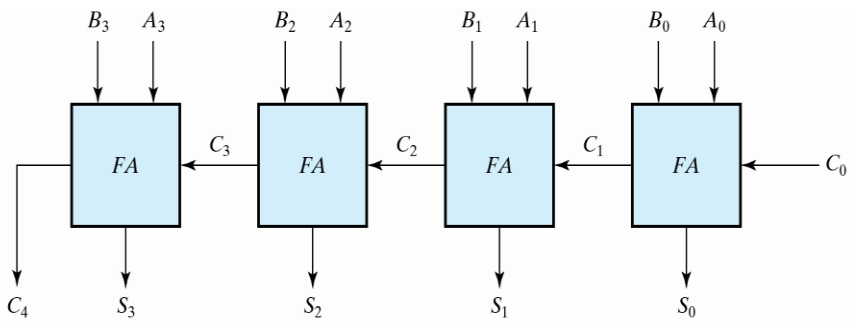
```
module half_adder (output S, C, input x, y);
   xor (S, x, y);
   and (C, x, y);
endmodule

module full_adder (output S, C, input x, y, z);
   wire     S1, C1, C2;
   half_adder HA1 (S1, C1, x, y);
   half_adder HA2 (S, C2, S1, z);
   or G1 (C, C2, C1);
endmodule
```



```
module ripple_carry_4_bit_adder ( output [3: 0] Sum, output C4, input [3:0]  A, B, input C0);
    wire          C1, C2, C3;      // Intermediate carries

    full_adder     FA0 (Sum[0], C1, A[0], B[0], C0),
                   FA1 (Sum[1], C2, A[1], B[1], C1),
                   FA2 (Sum[2], C3, A[2], B[2], C2),
                   FA3 (Sum[3], C4, A[3], B[3], C3);
endmodule
```



Dataflow Description of 2-to-4-line Decoder

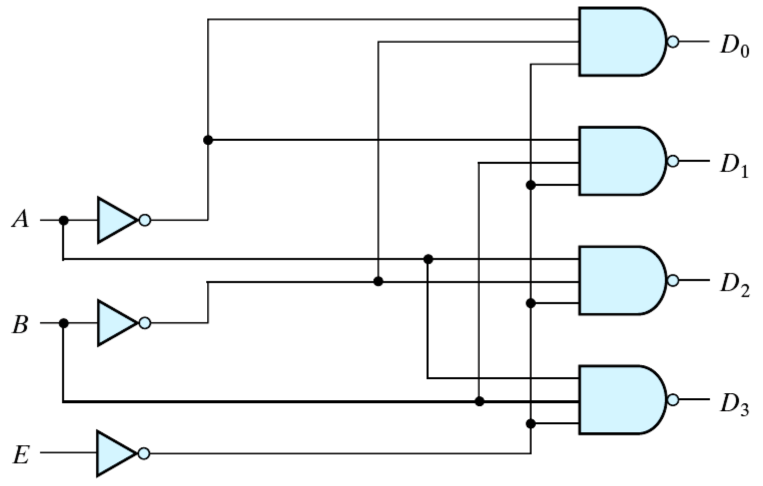
```
module decoder_2x4_df (D, A, B, enable);
  output [0: 3]     D;
  input             A, B;
  input             enable;


  assign D[0] = !((~A) && (!B) && (!enable)),
         D[1] = !((!A) && B & !enable),
         D[2] = !(A & (!B) && !enable),
         D[3] = !(A & B && !enable);

endmodule
```

A	B	D_0	D_1	D_2	D_3
0	0	E	1	1	1
0	1	1	E	1	1
1	0	1	1	E	1
1	1	1	1	1	E

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0



(a) Logic diagram

Gate-level Description of 2-to-4-line Decoder

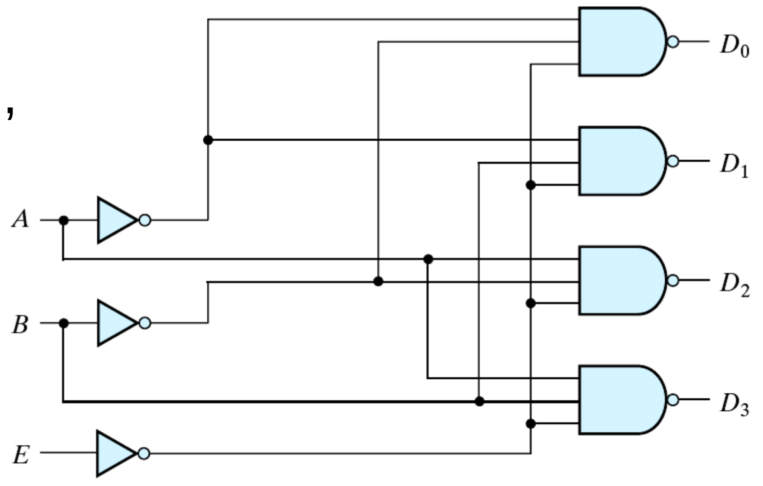
```
module decoder_2x4_gates (D, A, B, enable);
   output [0: 3]    D;
   input            A, B;
   input            enable;
   wire             A_not, B_not, enable_not;

    not
      G1  (A_not, A),
      G2  (B_not, B),
      G3  (enable_not, enable);
    nand
      G4  (D[0], A_not, B_not, enable_not),
      G5  (D[1], A_not, B, enable_not),
      G6  (D[2], A, B_not, enable_not),
      G7  (D[3], A, B, enable_not);

endmodule
```

A	B	D_0	D_1	D_2	D_3
0	0	E	1	1	1
0	1	1	E	1	1
1	0	1	1	E	1
1	1	1	1	1	E

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

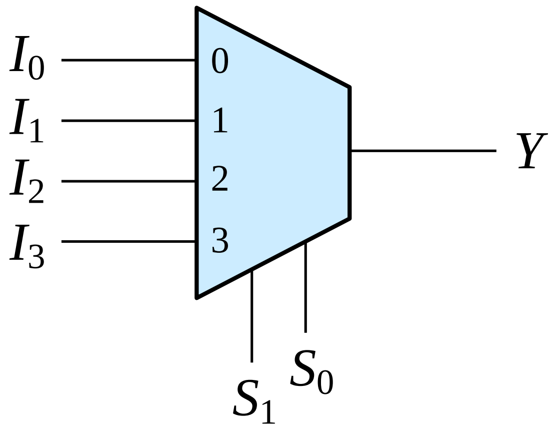


(a) Logic diagram

Behavioral Description of 4-to-1 line Multiplexer

```
module mux_4x1_beh(
   output reg     Y,
   input          I0, I1, I2, I3,
   input [1: 0]   Select
);

  always @ (I0, I1, I2, I3, Select)
    case (Select)
        2'b00:    Y = I0;
        2'b01:    Y = I1;
        2'b10:    Y = I2;
        2'b11:    Y = I3;
    endcase
  endmodule
```



S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

Exercise 1: 8-bit Magnitude Comparator (1/2)

- Design and verify the 8-bit magnitude comparator composed of two 4-bit magnitude comparators
- Algorithm -> logic
  - $A = A_3A_2A_1A_0$; $B = B_3B_2B_1B_0$
  - $A = B$ if $A_3=B_3$, $A_2=B_2$, $A_1=B_1$ and $A_1=B_1$
    - equality: $x_i= A_iB_i+A_i'B_i'$
    - $(A=B) = x_3x_2x_1x_0$
  - $(A>B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1 A_0B_0'$
  - $(A<B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1 A_0'B_0$
- Implementation
  - $x_i = (A_iB_i' + A_i'B_i)'$

A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	L	G	E
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	1	0	1	0	0
				⋮					⋮	

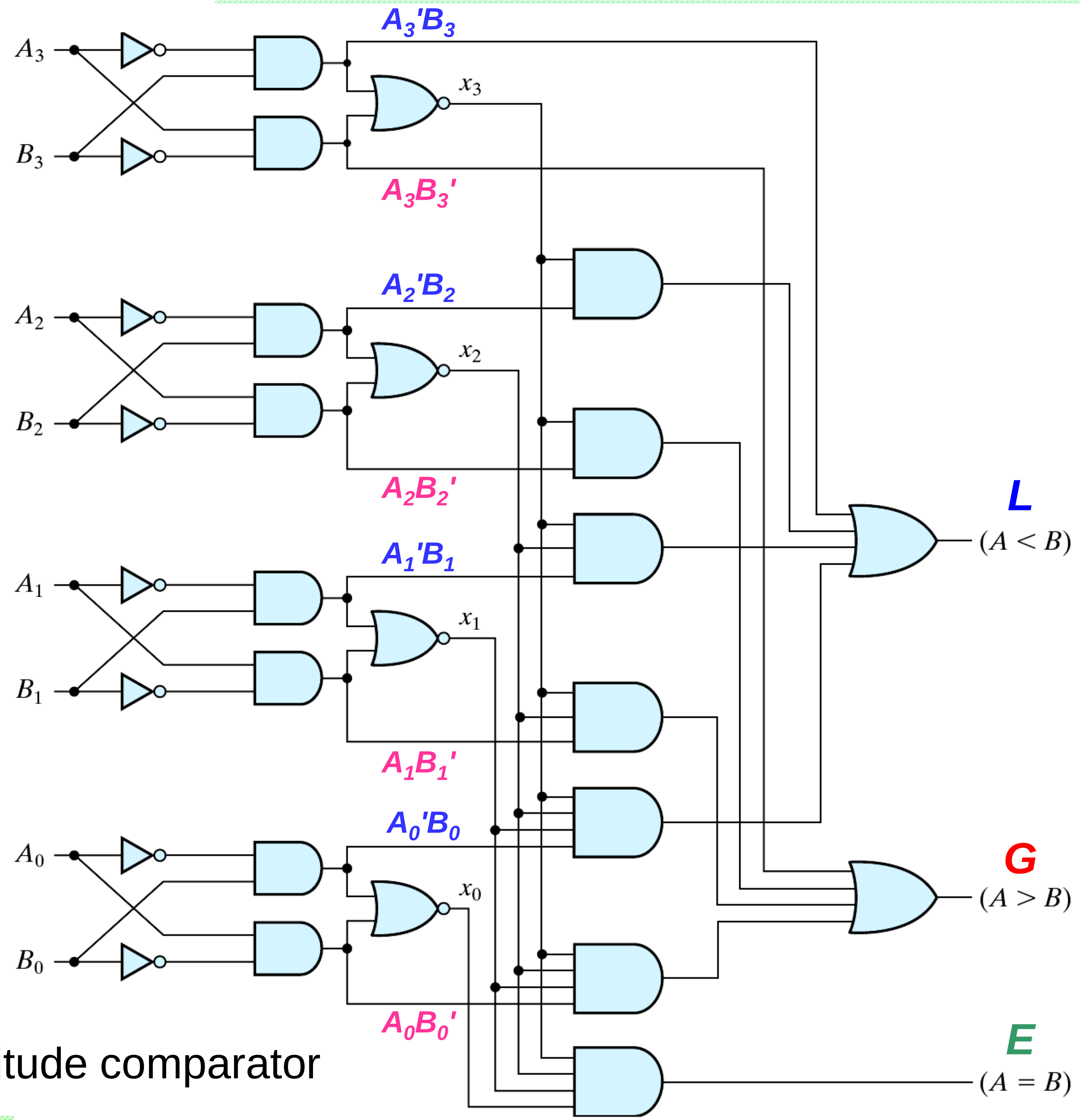


Fig. 4.17
Four-bit magnitude comparator

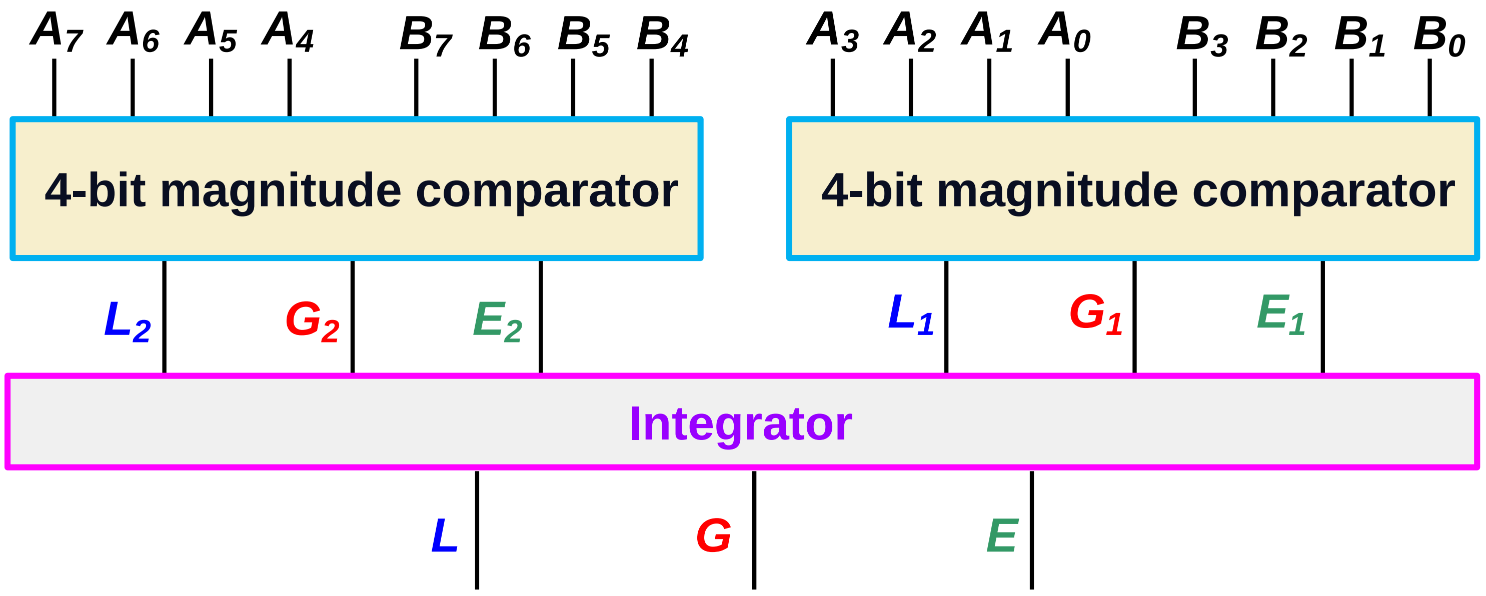
Exercise 1: 8-bit Magnitude Comparator (2/2)

- Algorithm -> logic
  - $A = A_7A_6A_5A_4A_3A_2A_1A_0$
  - $B = B_7B_6B_5B_4B_3B_2B_1B_0$
  - $A=B$: if $E_1 = 1$ and $E_2 = 1 \Rightarrow E = 1$
  - $A>B$: if $G_2 + E_2G_1 = 1 \Rightarrow G = 1$
  - $A<B$: if $L_2 + E_2L_1 = 1 \Rightarrow L = 1$

10101010, 10101010

11010011, 10100000
01010111, 01010011

01111111, 10000000
10010011, 10010100

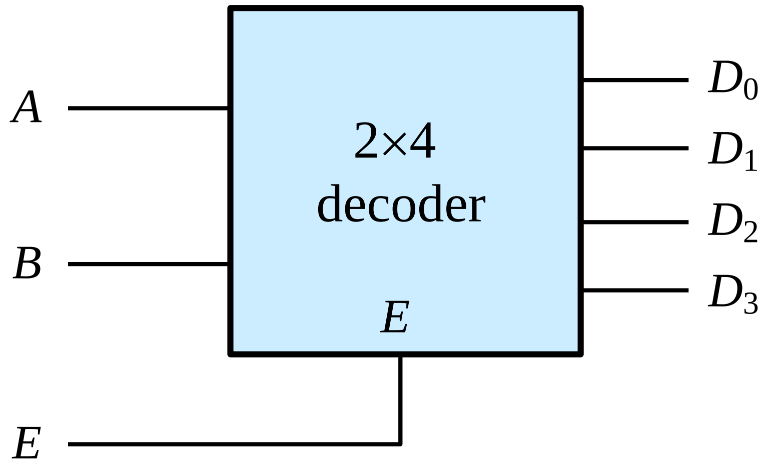


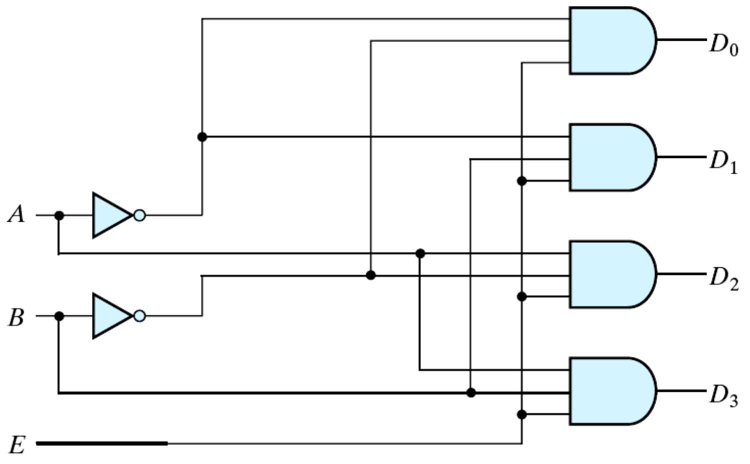
Exercise 2: 3-to-8 Decoder (1/2)

- Design and verify a 3-to-8 decoder composed of two 2-to-4 decoders

A	B	D_0	D_1	D_2	D_3
0	0	*E*	0	0	0
0	1	0	*E*	0	0
1	0	0	0	*E*	0
1	1	0	0	0	*E*

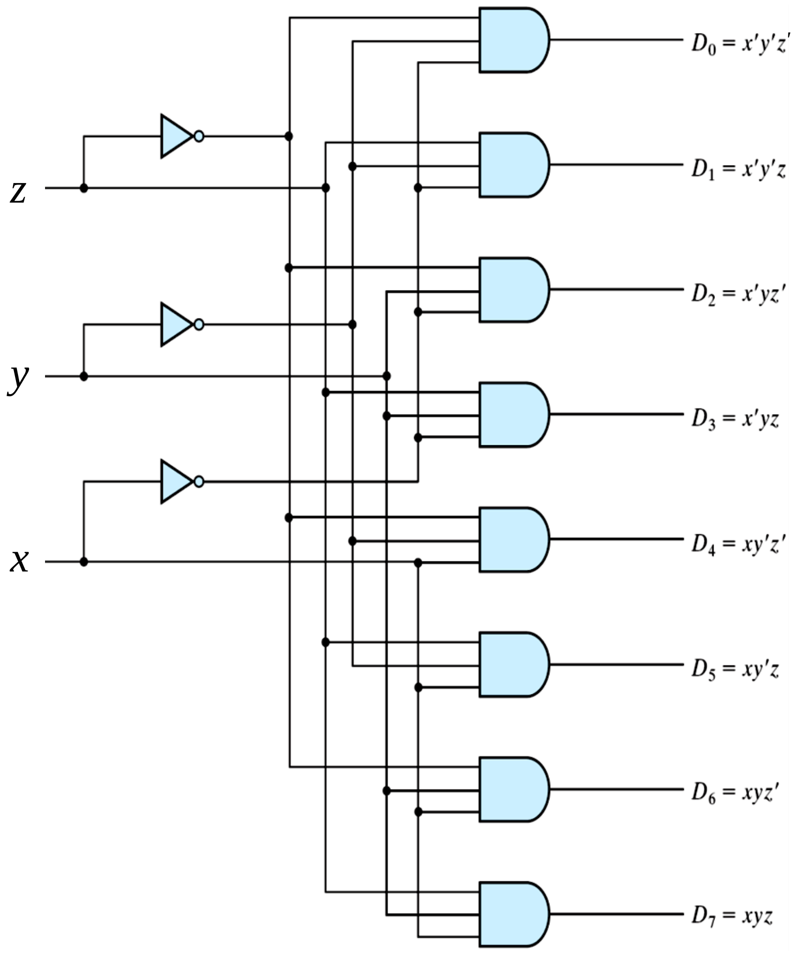
E	A	B	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

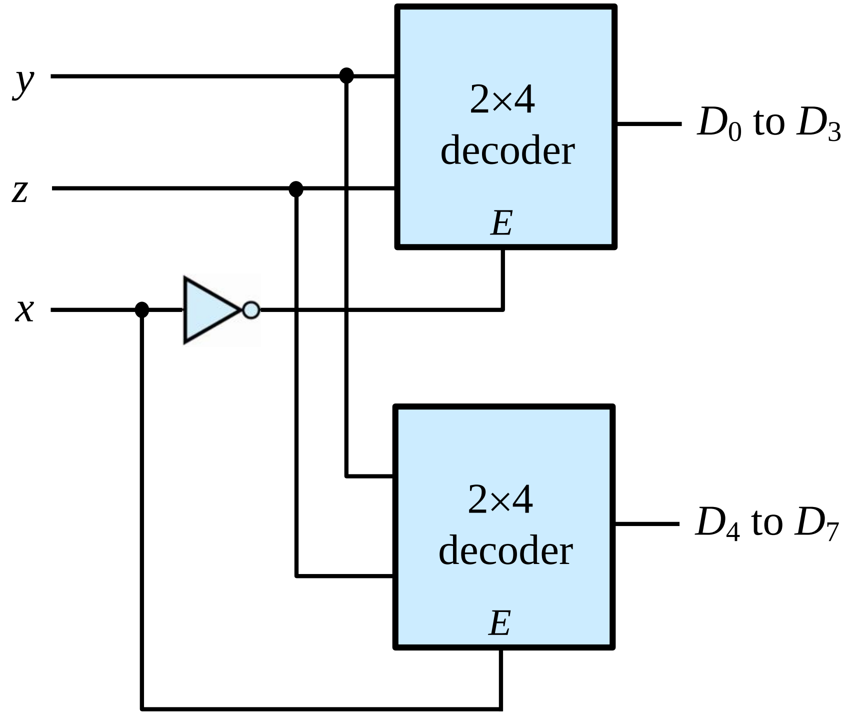




(a) Logic diagram

Exercise 2: 3-to-8 Decoder (2/2)





Exercise 3: Boolean Function Implementation (1/2)

- Design and verify the 8-to-1 multiplexer using Verilog HDL
  - Behavioral level modeling
- Implement and verify the following Boolean function of 4 input variable using 8-to-1 MUX
  - $F(A, B, C, D) = \Sigma(1, 2, 5, 8, 9, 10, 12, 13)$

Exercise 3: Boolean Function Implementation (2/2)

- Example: $F(A, B, C, D) = \Sigma(1, 3, 4, 11, 12, 13, 14, 15)$

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

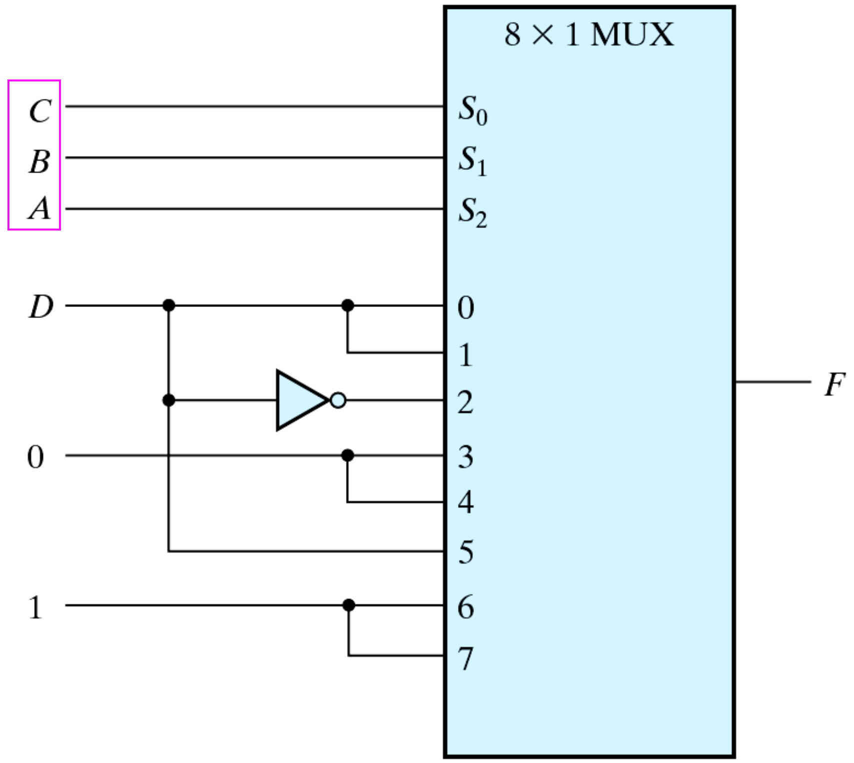


Fig. 4.28 Implementing a four-input function with a multiplexer