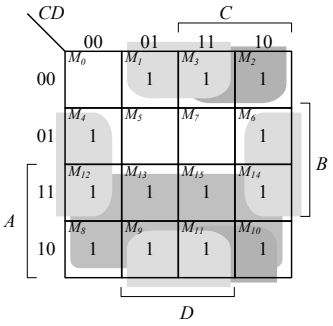
CHAPTER 4

4.1

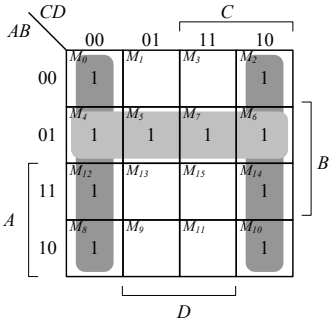
(a) $T_1 = B'C$, $T_2 = A'B$, $T_3 = A + T_1 = A + B'C$,
$T_4 = D \oplus T_2 = D \oplus (A'B) = A'BD' + D(A + B') = A'BD' + AD + B'D$
$F_1 = T_3 + T_4 = A + B'C + A'BD' + AD + B'D$
With $A + AD = A$ and $A + A'BD' = A + BD'$:
$F_1 = A + B'C + BD' + B'D$
Alternative cover: $F_1 = A + CD' + BD' + B'D$

$F_2 = T_2 + D' = A'B + D'$

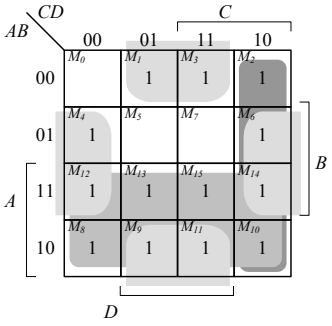
$ABCD$	T_1	T_2	T_3	T_4	F_1	F_2
0000	0	0	0	0	0	1
0001	0	0	0	1	1	0
0010	1	0	1	0	1	1
0011	1	0	1	1	1	0
0100	0	1	0	1	1	1
0101	0	1	0	0	0	1
0110	0	1	0	1	1	1
0111	0	1	0	0	0	1
1000	0	0	1	0	1	1
1001	0	0	1	1	1	0
1010	1	0	1	0	1	1
1011	1	0	1	1	1	0
1100	0	0	1	0	1	1
1101	0	0	1	1	1	0
1110	0	0	1	0	1	1
1111	0	0	1	1	1	0



$F_1 = A + B'C + B'D + BD'$



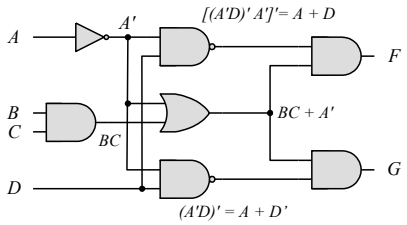
$F_2 = A'B + D'$



$F_1 = A + CD' + B'D + BD'$

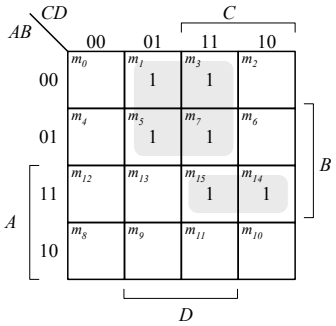
f2 = C'D' + A'B + CD'

4.2

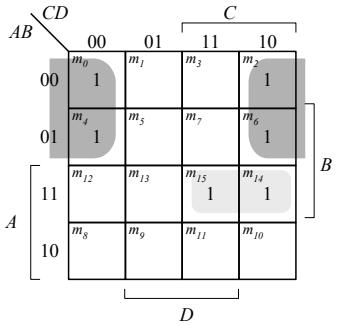


$F = (A + D)(A' + BC) = A'D + ABC + BCD + += A'D + ABC$

$F = (A + D')(A' + BC) = A'D' + ABC + BCD' = A'D' + ABC$



$F = A'D + ABC + BCD = A'D + ABC$



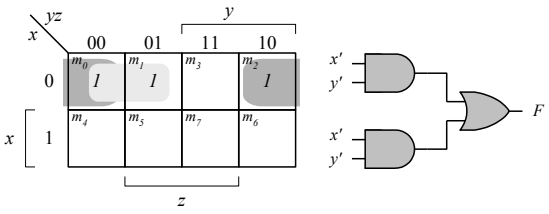
$G = A'D' + ABC + BCD' = A'D' + ABC$

4.3 **(a)** $Y_i = (A_iS' + B_iS)E'$ *for* $i = 0, 1, 2, 3$

(b) 1024 rows and 14 columns

4.4 **(a)**

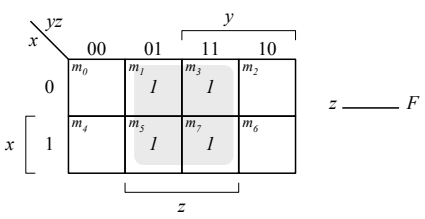
xyz	F
000	1
001	1
010	1
011	0
100	0
101	0
110	0
111	0



$F = x'y' + x'z'$

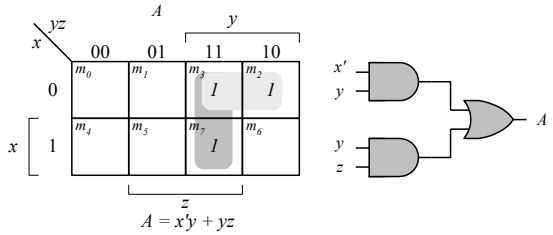
(b)

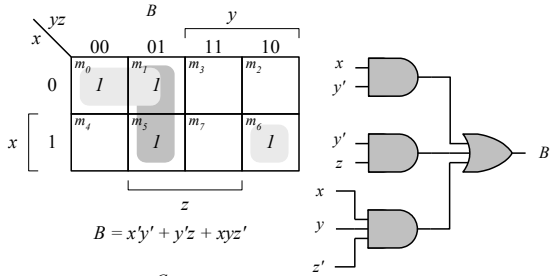
xyz	F
000	0
001	1
010	0
011	0
100	0
101	0
110	0
111	0

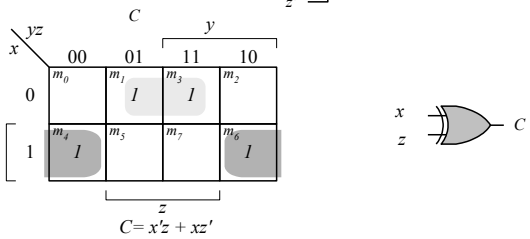


4.5

xyz	ABC
000	010
001	011
010	100
011	101
100	001
101	010
110	011
111	100

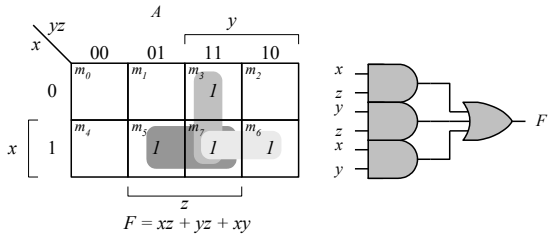






4.6

xyz	F
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

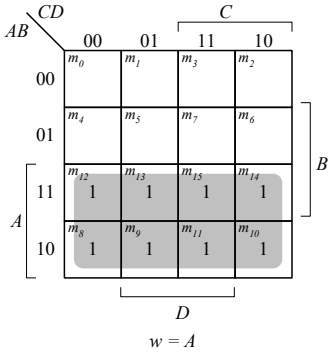


$F = xz + yz + xy$

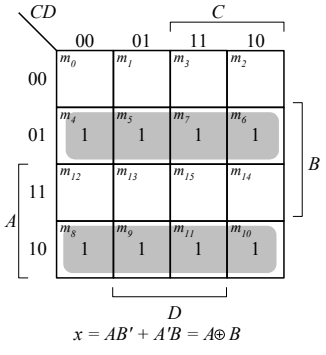
```
module Prob_4_6 (output F, input x, y, z);
  assign F = (x & z) | (y & z) | (x & y);
endmodule
```

4.7 (a)

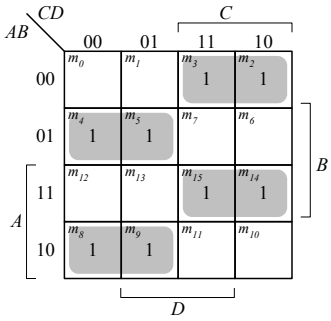
ABCD	*wxyz*
0000	0000
0001	0001
0011	0010
0010	0011
0110	0100
0111	0101
0101	0110
0100	0111
1100	1000
1101	1001
1111	1010
1110	1011
1010	1100
1011	1101
1001	1110
1000	1111



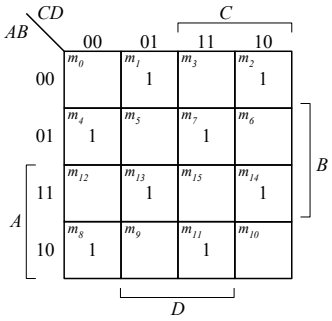
$w = A$



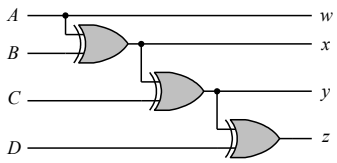
$x = AB' + A'B = A \oplus B$



$y = A'B'C\ \ A'BC' + ABC + AB'C'$
$= A'(A \oplus B) + A(B \oplus C)'$
$= A \oplus B \oplus C$
$= X \oplus C$



$z = A \oplus B \oplus C \oplus D$
$= y \oplus D$



(b)

```
module Prob_4_7(output w, x, y, z, input A, B, C, D);
  always @ (A, B, C, D)
    case ({A, B, C, D})
       4'b0000:      {w, x, y, z} = 4'b0000;
       4'b0001:      {w, x, y, z} = 4'b1111;
       4'b0010:      {w, x, y, z} = 4'b1110;
       4'b0011:      {w, x, y, z} = 4'b1101;
       4'b0100:      {w, x, y, z} = 4'b1100;
       4'b0101:      {w, x, y, z} = 4'b1011;
       4'b0110:      {w, x, y, z} = 4'b1010;
       4'b0111:      {w, x, y, z} = 4'b1001;

       4'b1000:      {w, x, y, z} = 4'b1000;
       4'b1001:      {w, x, y, z} = 4'b0111;
       4'b1010:      {w, x, y, z} = 4'b0110;
       4'b1011:      {w, x, y, z} = 4'b0101;
       4'b1100:      {w, x, y, z} = 4'b0100;
       4'b1101:      {w, x, y, z} = 4'b0011;
       4'b1110:      {w, x, y, z} = 4'b0010;
       4'b1111:      {w, x, y, z} = 4'b0001;
    endcase
endmodule
```

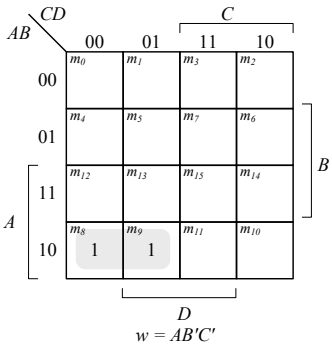
Alternative model:

```
module Prob_4_7(output w, x, y, z, input A, B, C, D);
  assign w = A;
  assign x = A ^ B);
  assign y = x ^ C;
  assign z = y ^ D;
endmodule
```

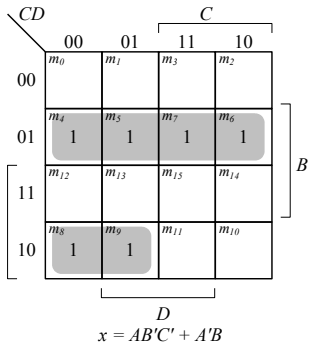
4.8 **(a)** The 8-4-2-1 code (Table 1.5) and the BCD code (Table 1.4) are identical for digits 0 – 9.

(b)

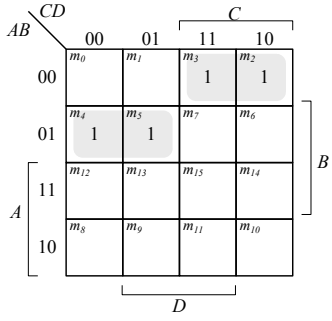
8421 *ABCD*	*Gray* *wxyz*
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101



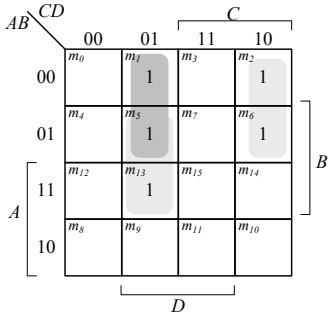
$w = AB'C'$



$x = AB'C' + A'B$



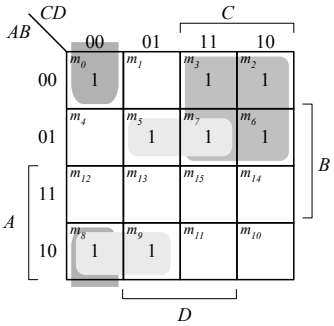
$y = A'BD' + A'B'D$



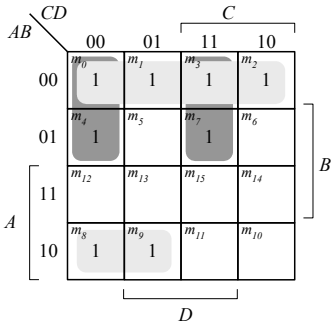
$z = A'C'D + BC'D + A'CD'$

4.9

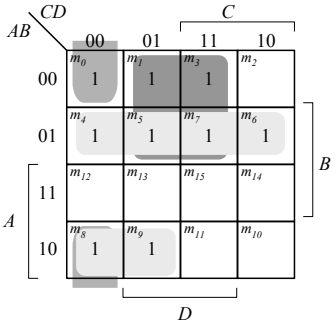
ABCD	a	b	c	d	e	f	g
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1	1	1	0	0	1
0100	0	1	1	0	0	1	1
0101	1	0	1	1	0	1	1
0110	1	0	1	1	1	1	1
0111	1	1	1	0	0	0	0
1000	1	1	1	1	1	1	1
1001	1	1	1	1	0	1	1



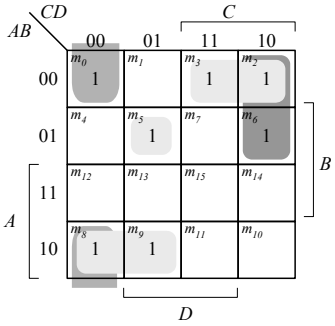
$a = A'C + A'BD + B'C'D' + AB'C'$



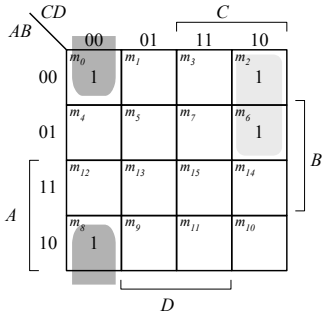
$b = A'B' + A'C'D' + A'CD + AB'C'$



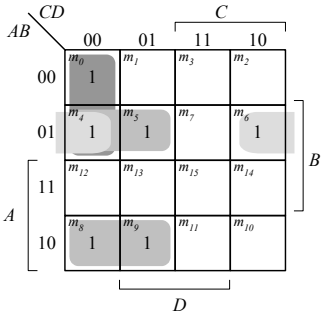
$c = A'B + A'D + B'C'D' + AB'C'$



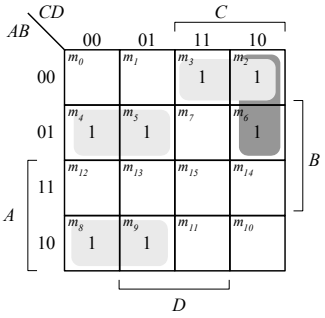
$d = A'CD' + A'B'C + B'C'D' + AB'C' + A'BC'D$



$e = A'CD' + B'C'D'$



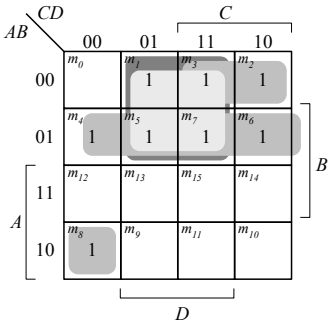
$f = A'BC' + A'C'D' + A'BD + AB'C'$



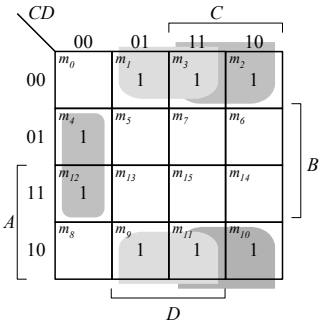
$g = A'CD' + A'B'C + A'BC' + AB'C'$

4.10

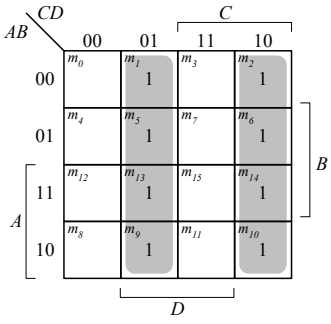
ABCD	wxyz
0000	0000
0001	1111
0010	1110
0011	1101
0100	1100
0101	1011
0110	1001
0111	1000
1000	1000
1001	0111
1010	0110
1011	0101
1100	0100
1101	0011
1110	0010
1111	0001



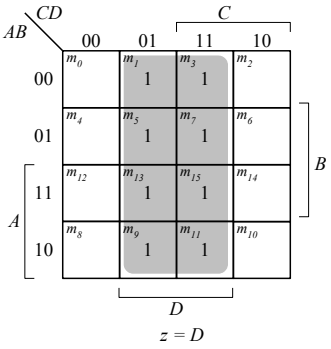
$w = A'(B + C + D) + AB'C'D'$
$= A \oplus (B + C + D)$



$x = B'(C + D) + CB'D'$
$= B \oplus (C + D)$



$y = CD' + C'D = C \oplus D$

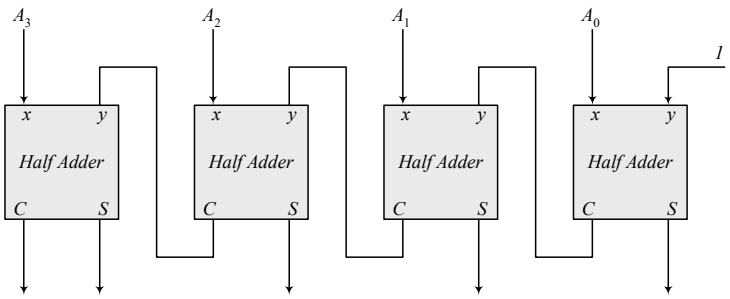


$z = D$

For a 5-bit 2's complementer with input E and output v:

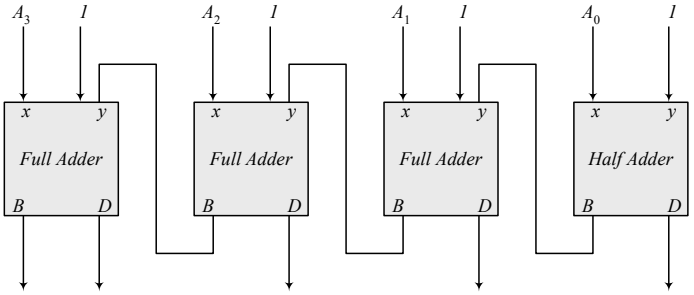
$$v = E \oplus (A + B + C + D)$$

4.11 **(a)**



Note: 5-bit output

(b)



Note: To decrement the 4-bit number, add -1 to the number. In 2's complement format (add F_h) to the number. An attempt to decrement 0 will assert the borrow bit. For waveforms, see solution to Problem 4.52.

4.12

(a)

x y	B D
0 0	0 0
0 1	1 1
1 0	0 1
1 1	0 0

$D = x'y + xy'$
$B = x'y$

(b)

x y B_{in}	B D
0 0 0	0 0
0 0 1	1 1
0 1 0	1 1
0 1 1	1 0
1 0 0	0 1
1 0 1	0 0
1 1 0	0 0
1 1 1	1 1

$Diff = x \oplus y \oplus z$
$B_{out} = x'y + x'z + yz$

4.13

	Sum	*C*	*V*
(a)	1101	0	1
(b)	0001	1	1
(c)	0100	1	0
(d)	1011	0	1
(e)	1111	0	0

4.14 xor AND OR XOR

10 + 5 + 5 + 10 = 30 ns

4.15 $C_4 = G_3 + P_3C_3 = G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0)$

$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$

4.16 **(a)**

$(C'G'_i + p'_i)' = (C_i + G_i)P_i = G_iP_i + P_iC_i$
$= A_iB_i(A_i + B_i) + P_iC_i$
$= A_iB_i + P_iC_i = G_i + P_iC_i$
$= A_iB_i + (A_i + B_i)C_i = A_iB_i + A_iC_i + B_iC_i = C_{i+1}$
$(P_iG'_i) \oplus C_i = (A_i + B_i)(A_iB_i)' \oplus C_i = (A_i + Bi)(A'_i + B'_i) \oplus C_i$
$= (A'_iB_i + A_iB'_i) \oplus C_i = A_i \oplus B_i \oplus C_i = S_i$

(b)

Output of NOR gate = $(A_0 + B_0)' = P'_0$
Output of NAND gate = $(A_0B_0)' = G'_0$
$S_1 = (P_0G'_0) \oplus C_0$
$C_1 = (C'_0G'_0 + P'_0)'$ as defined in part (a)

4.17 **(a)**

$(C'_iG'_i + P'_i)' = (C_i + G_i)P_i = G_iP_i + P_iC_i = A_iB_i(A_i + B_i) + P_iC_i$
$= A_iB_i + P_iC_i = G_i + P_iC_i$
$= A_iB_i + (A_i + B_i)C_i = A_iB_i + A_iC_i + B_iC_i = C_{i+1}$

$(P_iG'_i) \oplus C_i = (A_i + B_i)(A_iB_i)' \oplus C_i = (A_i + B_i)(A'_i + B'_i) \oplus C_i$
$= (A'_iB_i + A_iB'_i) \oplus C_i = A_i \oplus B_i \oplus C_i = S_i$

(b)

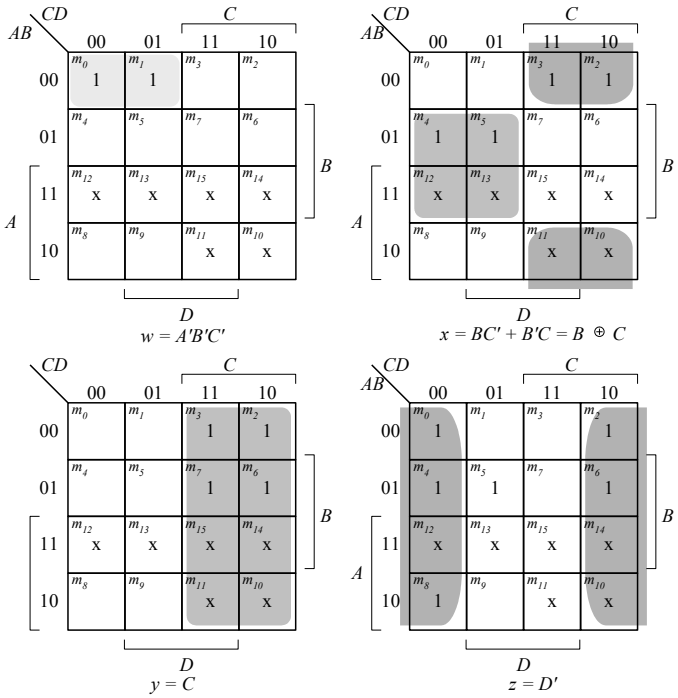
Output of NOR gate = $(A_0 + B_0)' = P'_0$
Output of NAND gate = $(A_0B_0)' = G'_0$

$S_0 = (P_0G'_0) \oplus C_0$
$C_1 = (C'_0G'_0 + P'_0)'$ as defined in part (a)

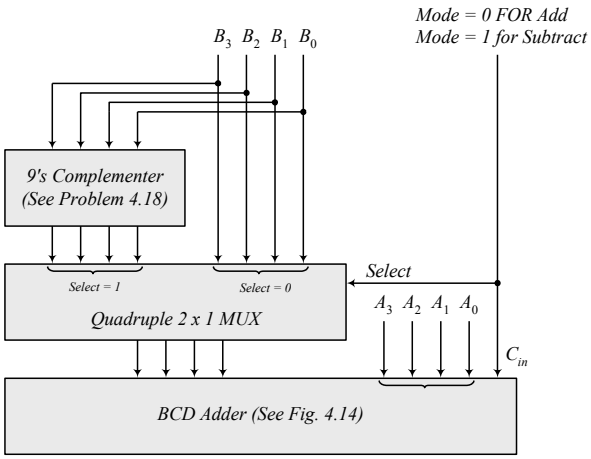
4.18

Inputs *ABCD*	Outputs *wxyz*
0000	1001
0001	1000
0010	0111
0011	0110
0100	0101
0101	0100
0110	0011
0111	0010
1000	0001
1001	0000

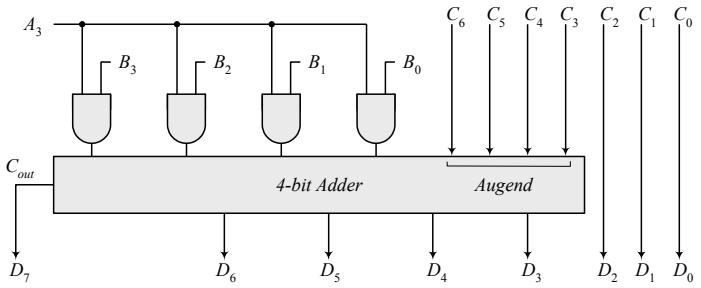
$d(A, b, c, d) = \Sigma(10, 11, 12, 13, 14, 15)$



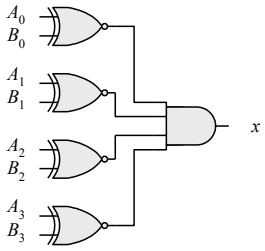
4.19



4.20 Combine the following circuit with the 4-bit binary multiplier circuit of Fig. 4.16.



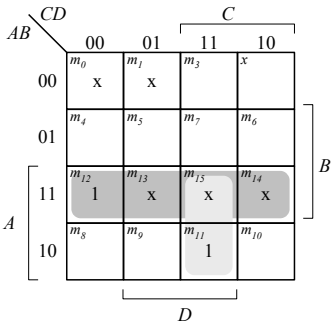
4.21

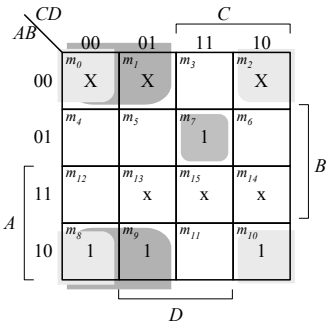


$x = (A_0 \oplus B_0)'(A_1 \oplus B_1)'(A_2 \oplus B_2)'(A_3 \oplus B_3)'$

4.22

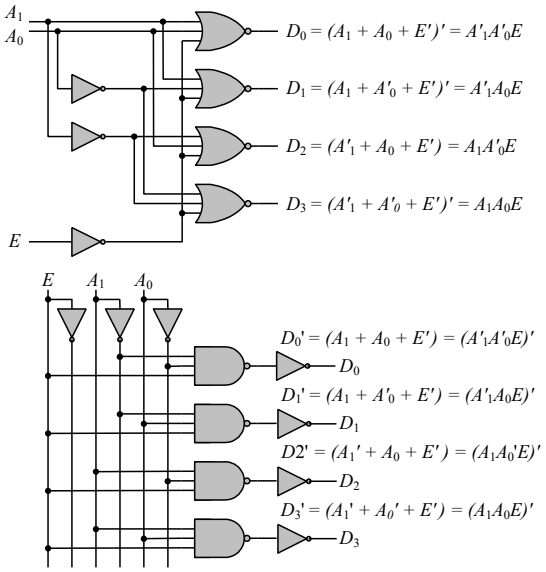
XS-3 *ABCD*	Binary *wxyz*
0011	0000
0100	0001
0101	0010
0110	0011
0111	0100
1000	0101
1001	0110
1010	0111
1011	1000
1100	1001





4.23

D0 = A1'A0' = (A1 + A0)' (NOR) D0' = (A1'A0')' (NAND)
D1 = A1'A0 = (A1 + A0')' (NOR) D1' = (A1'A0)' (NAND)
D2 = A1A0' = (A1' + A0)' (NOR) D2' = (A1A0')' (NAND)
D3 = A1A0 = (A1' + A0)' (NOR) D0' = (A1A0)' (NAND)

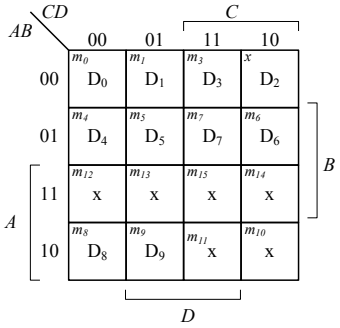


4.24

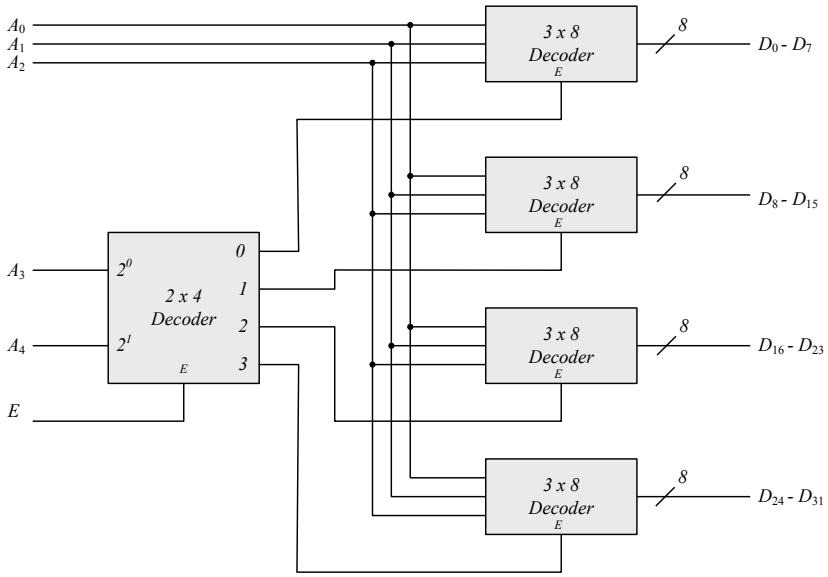
Inputs: *A, B, C, D*

Outputs: D_0, D_1, ... D_9

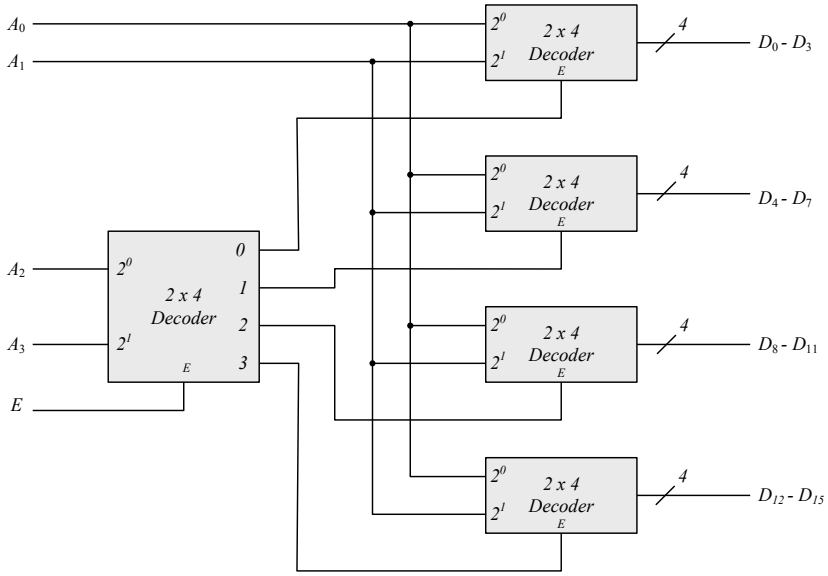
$D_0 = A'B'C'D'$
$D_1 = A'B'C'D$
$D_2 = B'CD'$
$D_3 = B'CD$
$D_4 = BC'D'$
$D_5 = BC'D$
$D_6 = BCD'$
$D_7 = BCD$
$D_8 = AD'$
$D_9 = AD$



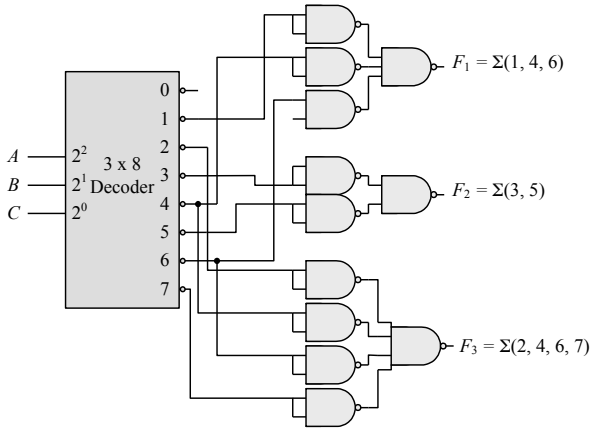
4.25



4.26



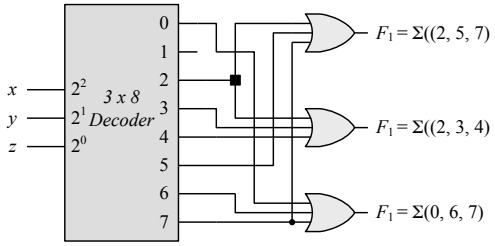
4.27



4.28 **(a)**

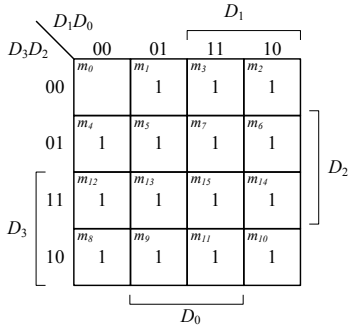
$F_1 = x(y + y')z + x'yz' = xyx + xy'z + x'yz' = \Sigma(2, 5, 7)$
$F_2 = xy'z' + x'y = xy'z' + x'yz + x'yz' = \Sigma(2, 3, 4)$
$F_3 = x'y'z' + xy(z + z') = x'y'z' + xyz + xyz' = \Sigma(0, 6, 7)$

(b)

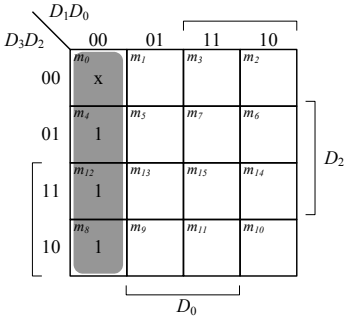


4.29

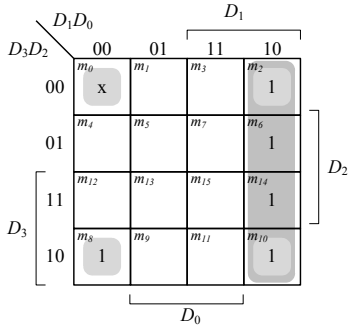
Inputs $D_3\ D_2\ D_1\ D_0$	Outputs $x\ y\ V$
0 0 0 0	x x 0
x x x 1	0 0 1
x x 1 0	0 1 1
x 1 0 0	1 0 1
1 0 0 0	1 1 1



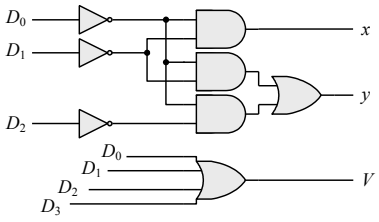
$V = D_0 + D_1 + D_2 + D_3$



$x = D_1'D_0'$



$y = D_0'D_2' + D_1D_0'$

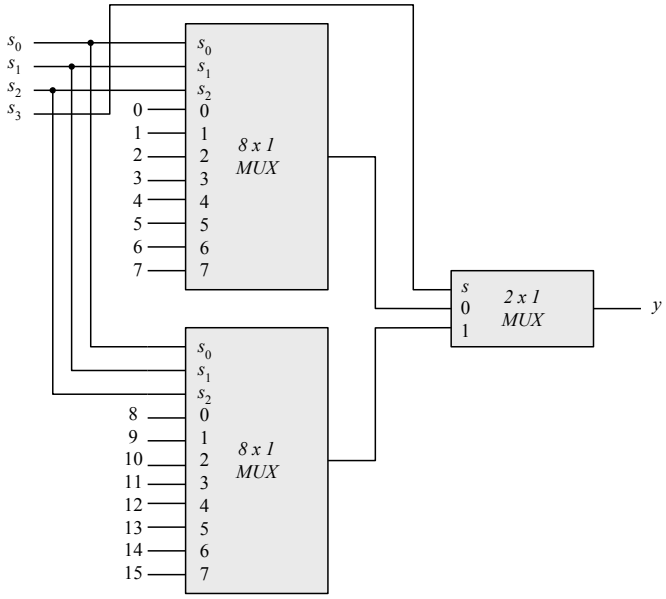


4.30

Inputs								*Outputs*			
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z	V
0	0	0	0	0	0	0	0	x	x	x	0
1	0	0	0	0	0	0	0	0	0	0	1
x	1	0	0	0	0	0	0	0	0	1	1
x	x	1	0	0	0	0	0	0	1	0	1
x	x	x	1	0	0	0	0	0	1	1	1
x	x	x	x	1	0	0	0	1	0	0	1
x	x	x	x	x	1	0	0	1	0	1	1
x	x	x	x	x	x	1	0	1	0	0	1
x	x	x	x	x	x	x	1	1	1	1	1

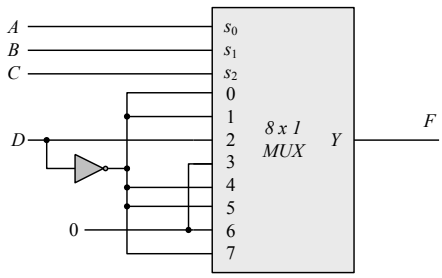
If $D_2 = 1$, $D_6 = 1$, all others = 0
Output xyz = 100 and V = 1

4.31



4.32 **(a)** $F = \Sigma\,(0, 2, 5, 8, 10, 14)$

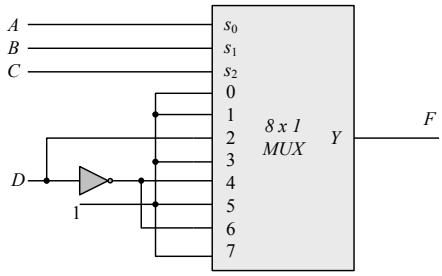
Inputs ABCD	*Mux input line (ABC)*	*Value*	*F = Σ(0, 2, 5, 8, 10, 14)*	
000 0	0	**0**	1	$F = D'$
000 1	0	1	0	
001 0	1	**2**	1	$F = D'$
001 1	1	3	0	
010 0	2	4	0	$F = D$
010 1	2	**5**	1	
011 0	3	6	0	$F = 0$
011 1	3	7	0	
100 0	4	**8**	1	$F = D'$
100 1	4	9	0	
101 0	5	**10**	1	$F = D'$
101 1	5	11	0	
110 0	6	12	0	$F = 0$
110 1	6	13	0	
111 0	7	**14**	1	$F = D'$
111 1	7	15	0	



(b)

$F = \Pi(2, 6, 11) = (A' + B' + C + D')(A' + B + C + D')(A + B' + C + D)$
$F' = (A' + B' + C + D')' + (A' + B + C + D')' + (A + B' + C + D)'$
$F' = (ABC'D) + (AB'C'D) + (A'BC'D') = \Sigma(13, 9, 4)$
$F = \Sigma(0, 1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 14, 15)$

Inputs ABCD	Mux input line (ABC)	Value	
000 0	0	**0**	1 $F = 1$
000 1	0	**1**	1
001 0	1	**2**	1 $F = 1$
001 1	1	**3**	1
010 0	2	4	0 $F = D$
010 1	2	**5**	1
011 0	3	**6**	1 $F = 1$
011 1	3	**7**	1
100 0	4	**8**	1 $F = D'$
100 1	4	9	0
101 0	5	**10**	1 $F = 1$
101 1	5	**11**	1
110 0	6	**12**	1 $F = D'$
110 1	6	13	0
111 0	7	**14**	1 $F = 1$
111 1	7	**15**	1

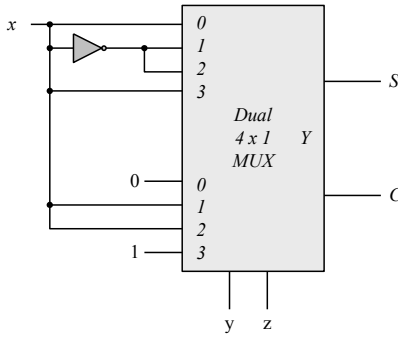


4.33

$S(x, y, z) = \Sigma(1, 2, 4, 7)$
$C(x, y, z) = \Sigma(3, 5, 6, 7)$

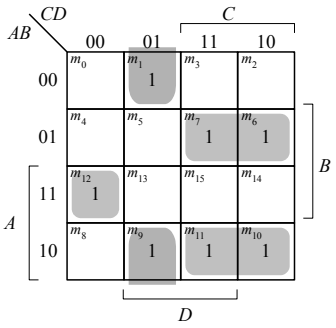
S	I_0	I_1	I_2	I_3
x′	0	1	2	3
x	4	5	6	7
	x	x′	x′	x

C	I_0	I_1	I_2	I_3
x′	0	1	2	3
x	4	5	6	7
	0	x′	x′	1



4.34 **(a)**

	A	B	C	D	F
$I_3 = 1$	0	1	1	0	1
	0	1	1	1	1
$I_5 = 1$	1	0	1	0	1
	1	0	1	1	1
$I_0 = D$	0	0	0	0	0
	0	0	0	1	1
$I_4 = D$	1	0	0	0	0
	1	0	0	1	1
$I_6 = D'$	1	1	0	0	1
	1	1	0	1	0

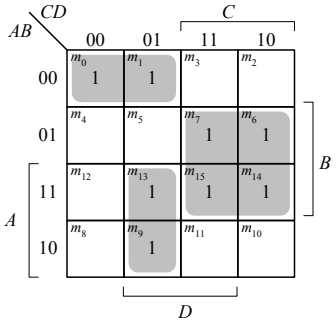


Other minterms = 0
since $I_1 = I_2 = I_7 = 0$

F(A, B, C, D) = Σ(1, 6, 7, 9, 10, 11, 12)

(b)

	A	B	C	D	F
$I_1 = 0$	0	0	1	0	0
	0	0	1	1	0
$I_2 = 0$	0	1	0	0	0
	0	1	0	1	0
$I_3 = 1$	0	1	1	0	1
	0	1	1	1	1
$I_7 = 1$	1	1	1	0	1
	1	1	1	1	1
$I_4 = D$	1	0	0	0	0
	1	0	0	1	1
$I_0 = D'$	0	0	0	0	1
	0	0	0	1	0
$I_6 = D'$	1	1	0	0	1
	1	1	0	1	0

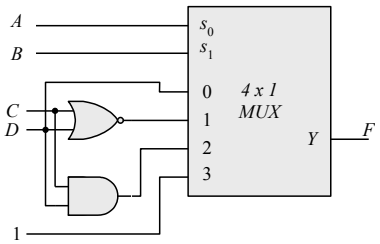


Other minterms = 0
since $I_1 = I_2 = 0$

F(A, B, C, D) = Σ(0, 1, 6, 7, 9, 13, 14, 15)

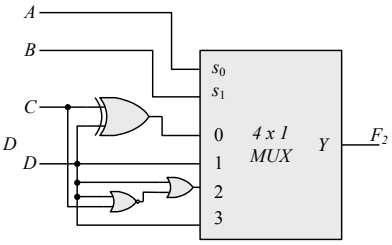
4.35 **(a)**

Inputs ABCD	*F*	
0000	0	
0001	1	*AB* = 00
0010	0	*F = D*
0011	1	
0100	1	
0101	0	*AB* = 01
0110	0	*F = C'D'*
0111	0	*= (C + D)'*
1000	0	
1001	0	*AB* = 10
1010	0	*F = CD*
1011	1	
1100	1	
1101	1	*AB* = 11
1110	1	*F = 1*
1111	1	



(b) F = S(1, 2, 5, 7, 8, 10, 11, 13, 15)

Inputs ABCD	$F_2 = \Sigma(1, 2, 5, 7, 8, 10, 11, 13, 15)$	
0000	0	
0001	1	$AB = 00$
0010	1	$F = C'D + CD'$
0011	0	
0100	0	
0101	1	$AB = 01$
0110	0	$F = C'D + CD = D$
0111	1	
1000	1	
1001	0	
1010	1	$AB = 10$
1011	1	$F = C'D' + C'D + CD = C'D' + D$
1100	0	
1101	1	$AB = 11$
1110	0	$F = D$
1111	1	



4.36

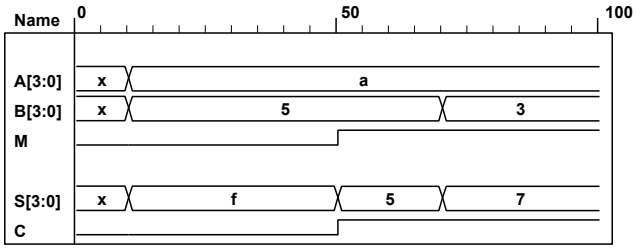
```
module priority_encoder_gates (output x, y, V, input D0, D1, D2, D3);  // V2001
  wire  w1, D2_not;
  not   (D2_not, D2);
  or    (x, D2, D3);
  or    (V, D0, D1, x);
  and   (w1, D2_not, D1);
  or    (y, D3, w1);
endmodule
```

Note: See Problem 4.45 for testbench)

4.37

```
module Add_Sub_4_bit (
  output [3: 0] S,
  output C,
  input [3: 0] A, B,
  input M
);
  wire  [3: 0] B_xor_M;
  wire  C1, C2, C3, C4;
  assign C = C4;        // output carry
  xor (B_xor_M[0], B[0], M);
  xor (B_xor_M[1], B[1], M);
  xor (B_xor_M[2], B[2], M);
  xor (B_xor_M[3], B[3], M);
  // Instantiate full adders
  full_adder FA0 (S[0], C1, A[0], B_xor_M[0], M);
  full_adder FA1 (S[1], C2, A[1], B_xor_M[1], C1);
  full_adder FA2 (S[2], C3, A[2], B_xor_M[2], C2);
  full_adder FA3 (S[3], C4, A[3], B_xor_M[3], C3);
endmodule

module full_adder (output S, C, input x, y, z); // See HDL Example 4.2
  wire S1, C1, C2;
  // instantiate half adders
  half_adder HA1 (S1, C1, x, y);
  half_adder HA2 (S, C2, S1, z);
  or G1 (C, C2, C1);
endmodule
```

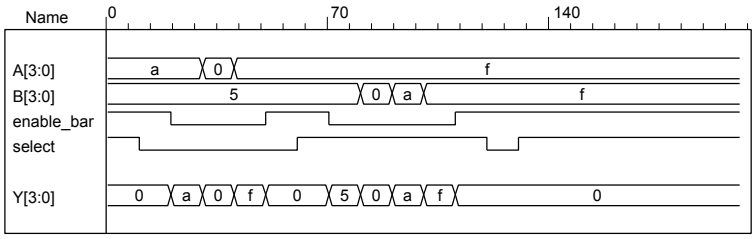
```
module half_adder (output S, C, input x, y);        // See HDL Example 4.2
  xor   (S, x, y);
  and   (C, x, y);
endmodule

module t_Add_Sub_4_bit ();
  wire [3: 0] S;
  wire C;
  reg [3: 0] A, B;
  reg M;

  Add_Sub_4_bit M0 (S, C, A, B, M);

  initial #100 $finish;
  initial fork
    #10 M = 0;
    #10 A = 4'hA;
    #10 B = 4'h5;
    #50 M = 1;
    #70 B = 4'h3;
  join
endmodule
```



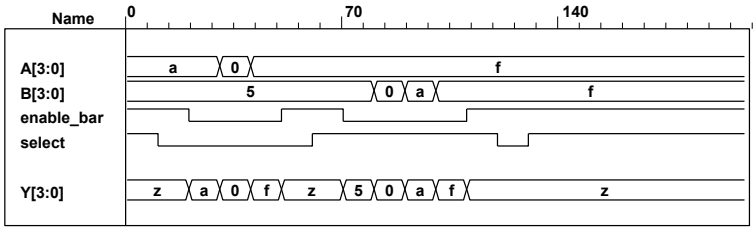
4.38

```
module quad_2x1_mux (                   // V2001
  input     [3: 0]   A, B,              // 4-bit data channels
  input              enable_bar, select,  // enable_bar is active-low)
  output    [3: 0]   Y                  // 4-bit mux output
);
  //assign Y = enable_bar ? 0 : (select ? B : A);          // Grounds output
  assign Y = enable_bar ? 4'bzzzz : (select ? B : A);    // Three-state output
endmodule
// Note that this mux grounds the output when the mux is not active.

module t_quad_2x1_mux ();
  reg     [3: 0] A, B, C;                 // 4-bit data channels
  reg            enable_bar, select;      // enable_bar is active-low)
  wire  [3: 0]   Y;                       // 4-bit mux

  quad_2x1_mux M0 (A, B, enable_bar, select, Y);

  initial #200 $finish;
  initial fork
    enable_bar = 1;
    select = 1;
    A = 4'hA;
    B = 4'h5;
    #10 select = 0;       // channel A
```

```
  #20 enable_bar = 0;
  #30 A = 4'h0;
  #40 A = 4'hF;
  #50 enable_bar = 1;
  #60 select = 1;       // channel B
  #70 enable_bar = 0;
  #80 B = 4'h00;
  #90 B = 4'hA;
  #100 B = 4'hF;
  #110 enable_bar = 1;
  #120 select = 0;
  #130 select = 1;
  #140 enable_bar = 1;
 join
endmodule
```



With three-state output:



4.39 // Verilog 1995

```
module Compare (A, B, Y);
  input [3: 0]   A, B;   // 4-bit data inputs.
  output   [5: 0]   Y;      // 6-bit comparator output.
  reg      [5: 0]   Y;      // EQ, NE, GT, LT, GE, LE

  always @ (A or B)
    if (A==B)          Y = 6'b10_0011;        // EQ, GE, LE
    else if (A < B)    Y = 6'b01_0101;        // NE, LT, LE
    else               Y = 6'b01_1010;        // NE, GT, GE
endmodule
```

// Verilog 2001, 2005

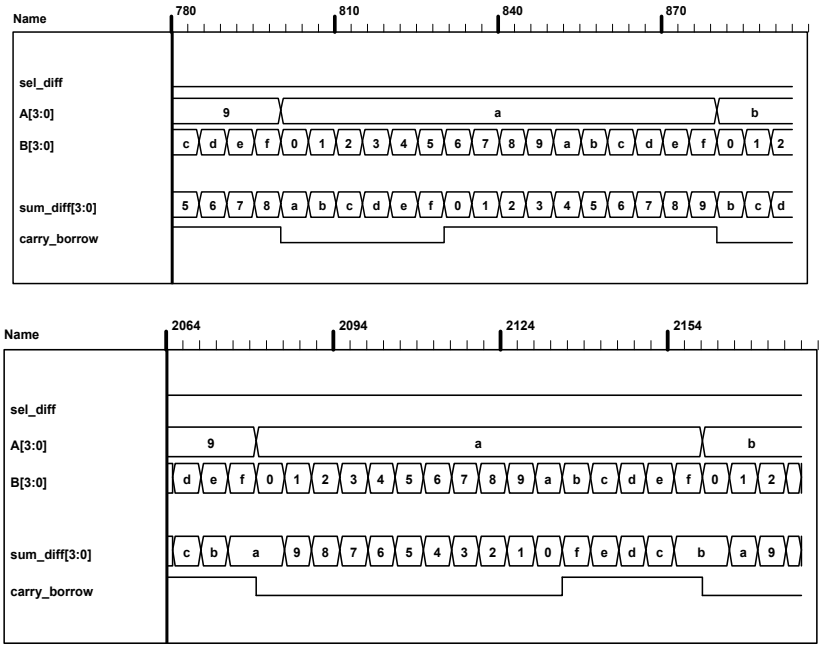
```
module Compare (input [3: 0] A, B, output reg [5:0] Y);
  always @ (A, B)
    if (A==B)          Y = 6'b10_0011;        // EQ, GE, LE
    else if (A < B)    Y = 6'b01_0101;        // NE, LT, LE
    else               Y = 6'b01_1010;        // NE, GT, GE
endmodule
```

4.40

```
 module Prob_4_40 (
 output [3: 0]  sum_diff, output carry_borrow,
 input [3: 0] A, B, input sel_diff
 );

   always @(sel_diff, A, B)    {carry_borrow, sum_diff} = sel_diff ? A - B : A + B;
endmodule

module t_Prob_4_40;
 wire  [3: 0] sum_diff;
 wire carry_borrow;
 reg [3:0] A, B;
 reg sel_diff;

 integer I, J, K;
 Prob_4_40 M0 (  sum_diff, carry_borrow,  A, B, sel_diff);
 initial #4000 $finish;
 initial begin
   for (I = 0; I < 2; I = I + 1) begin
     sel_diff = I;
     for (J = 0; J < 16; J = J + 1) begin
       A = J;
       for (K = 0; K < 16; K = K + 1) begin B = K; #5 ; end
     end
   end
 end
endmodule
```

4.41

```
 module Prob_4_41 (
 output reg [3: 0]  sum_diff, output reg carry_borrow,
 input [3: 0] A, B, input sel_diff
 );

   always @ (A, B, sel_diff)
   {carry_borrow, sum_diff} = sel_diff ? A - B : A + B;

endmodule


module t_Prob_4_41;
 wire  [3: 0] sum_diff;
 wire carry_borrow;
 reg [3:0] A, B;
 reg sel_diff;

 integer I, J, K;
 Prob_4_46 M0 (  sum_diff, carry_borrow,  A, B, sel_diff);
 initial #4000 $finish;
 initial begin
   for (I = 0; I < 2; I = I + 1) begin
     sel_diff = I;
     for (J = 0; J < 16; J = J + 1) begin
       A = J;
       for (K = 0; K < 16; K = K + 1) begin B = K; #5 ; end
     end
   end
 end
endmodule
```



4.42 **(a)**

```
module Xs3_Gates (input A, B, C, D, output w, x, y, z);
  wire  B_bar, C_or_D_bar;
  wire  CD, C_or_D;
  or    (C_or_D, C, D);
  not   (C_or_D_bar, C_or_D);
  not   (B_bar, B);
  and   (CD, C, D);
  not   (z, D);
  or    (y, CD,  C_or_D_bar);
  and   (w1, C_or_D_bar, B);
  and   (w2, B_bar, C_or_D);
  and   (w3, C_or_D, B);
  or    (x, w1, w2);
  or    (w, w3, A);
endmodule
```

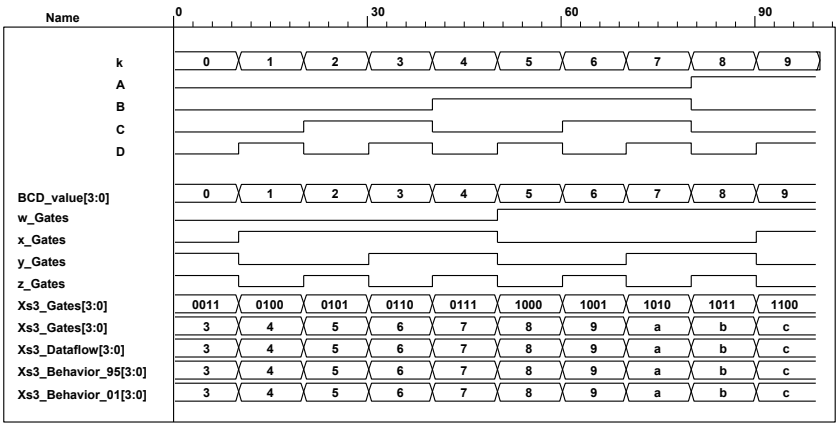
(b)

```
module Xs3_Dataflow (input A, B, C, D, output w, x, y, z);
assign {w, x, y, z} = {A, B, C, D} + 4'b0011;
endmodule
```

(c)

```
module Xs3_Behavior_95 (A, B, C, D, w, x, y, z);
  input     A, B, C, D;
  output    w, x, y, z;
  reg   w, x, y, z;

 always @ (A or B or C or D) begin {w, x, y, z} = {A, B, C, D} + 4'b0011; end
endmodule

module Xs3_Behavior_01 (input A, B, C, D, output reg w, x, y, z);
```

```
  always @ (A, B, C, D) begin {w, x, y, z} = {A, B,C, D} + 4'b0011; end
endmodule

module t_Xs3_Converters ();
  reg A, B, C, D;
  wire w_Gates, x_Gates, y_Gates, z_Gates;
  wire w_Dataflow, x_Dataflow, y_Dataflow, z_Dataflow;
  wire w_Behavior_95, x_Behavior_95, y_Behavior_95, z_Behavior_95;
  wire w_Behavior_01, x_Behavior_01, y_Behavior_01, z_Behavior_01;
  integer k;
  wire [3: 0] BCD_value;
  wire [3: 0] Xs3_Gates = {w_Gates, x_Gates, y_Gates, z_Gates};
  wire [3: 0] Xs3_Dataflow = {w_Dataflow, x_Dataflow, y_Dataflow, z_Dataflow};
  wire [3: 0] Xs3_Behavior_95 = {w_Behavior_95, x_Behavior_95, y_Behavior_95, z_Behavior_95};
  wire [3: 0] Xs3_Behavior_01 = {w_Behavior_01, x_Behavior_01, y_Behavior_01, z_Behavior_01};

  assign BCD_value = {A, B, C, D};
  Xs3_Gates   M0 (A, B, C, D, w_Gates, x_Gates, y_Gates, z_Gates);
  Xs3_Dataflow    M1 (A, B, C, D, w_Dataflow, x_Dataflow, y_Dataflow, z_Dataflow);
  Xs3_Behavior_95 M2 (A, B, C, D, w_Behavior_95, x_Behavior_95, y_Behavior_95, z_Behavior_95);
  Xs3_Behavior_01 M3 (A, B, C, D, w_Behavior_01, x_Behavior_01, y_Behavior_01, z_Behavior_01);

  initial #200 $finish;
  initial begin
    k = 0;
    repeat (10) begin {A, B, C, D} = k; #10 k = k + 1; end
  end
endmodule
```

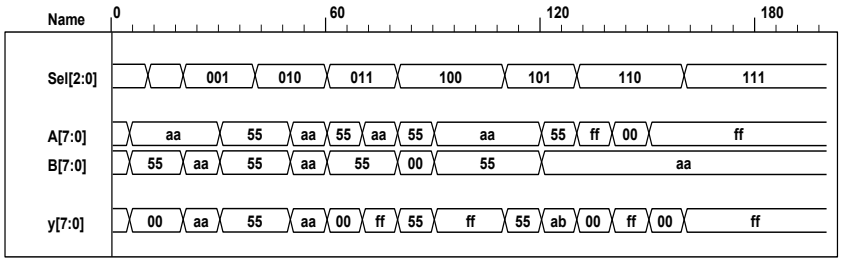


4.43 Two-channel mux with 2-bit data paths, enable, and three-state output.

4.44

```
module ALU (output reg [7: 0] y, input [7: 0] A, B, input [2: 0] Sel);
  always @ (A, B, Sel) begin
    y = 0;
    case (Sel)
      3'b000:    y = 8'b0;
      3'b001:    y = A & B;
      3'b010:    y = A | B;
      3'b011:    y = A ^ B;
      3'b100:    y = A + B;
```

```
      3'b101:    y = A - B;
      3'b110:    y = ~A;
      3'b111:    y = 8'hFF;
    endcase
  end

endmodule

module t_ALU ();
  wire[7: 0]y;
  reg [7: 0] A, B;
  reg [2: 0] Sel;

  ALU M0 (y, A, B, Sel);

initial #200 $finish;
  initial fork
     #5 begin A = 8'hAA; B = 8'h55; end      // Expect y = 8'd0
    #10 begin Sel = 3'b000; A = 8'hAA; B = 8'h55; end // y = 8'b000   Expect y = 8'd0
    #20 begin Sel = 3'b001; A = 8'hAA; B = 8'hAA; end // y = A & B    Expect y = 8'hAA = 8'1010_1010
    #30 begin Sel = 3'b001; A = 8'h55; B = 8'h55; end  // y = A & B   Expect y = 8'h55 = 8'b0101_0101
    #40 begin Sel = 3'b010; A = 8'h55; B = 8'h55; end  // y = A | B   Expect y = 8'h55 = 8'b0101_0101
    #50 begin Sel = 3'b010; A = 8'hAA; B = 8'hAA; end // y = A | B    Expect y = 8'hAA = 8'b1010_1010
    #60 begin Sel = 3'b011; A = 8'h55; B = 8'h55; end  // y = A ^ B   Expect y = 8'd0
    #70 begin Sel = 3'b011; A = 8'hAA; B = 8'h55; end // y = A ^ B    Expect y = 8'hFF = 8'b1111_1111
    #80 begin Sel = 3'b100; A = 8'h55; B = 8'h00; end  // y = A + B   Expect y = 8'h55 = 8'b0101_0101
    #90 begin Sel = 3'b100; A = 8'hAA; B = 8'h55; end // y = A + B    Expect y = 8'hFF = 8'b1111_1111
  #110 begin Sel = 3'b101; A = 8'hAA; B = 8'h55; end // y = A – B    Expect y = 8'h55 = 8'b0101_0101
  #120 begin Sel = 3'b101; A = 8'h55; B = 8'hAA; end // y = A – B    Expect y = 8'hab = 8'b1010_1011
  #130 begin Sel = 3'b110; A = 8'hFF; end            // y = ~A       Expect y = 8'd0
  #140 begin Sel = 3'b110; A = 8'd0; end             // y = ~A       Expect y = 8'hFF = 8'b1111_1111
  #150 begin Sel = 3'b110; A = 8'hFF; end            // y = ~A       Expect y = 8'd0
  #160 begin Sel = 3'b111;   end                     // y = 8'hFF    Expect y = 8'hFF = 8'b1111_1111
  join
endmodule
```

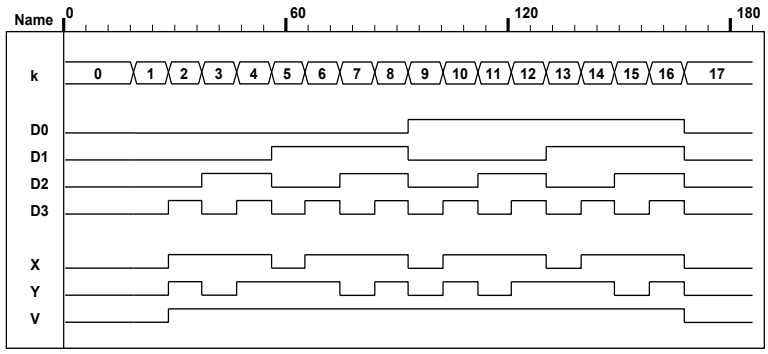


Note that the subtraction operator performs 2's complement subtraction. So 8'h55 – 8'hAA adds the 2's complement of 8'hAA to 8'h55 and gets 8'hAB. The sign bit is not included in the model, but hand calculation shows that the 9th bit is 1, indicating that the result of the operation is negative. The magnitude of the result can be obtained by taking the 2's complement of 8'hAB.

4.45

```
module priority_encoder_beh (output reg X, Y, V, input D0, D1, D2, D3);   // V2001
  always @ (D0, D1, D2, D3) begin
    X = 0;
    Y = 0;
    V = 0;
    casex ({D0, D1, D2, D3})
```

```
    4'b0000:   {X, Y, V} = 3'bxx0;
    4'b1000:   {X, Y, V} = 3'b001;
    4'bx100:   {X, Y, V} = 3'b011;
    4'bxx10:   {X, Y, V} = 3'b101;
    4'bxxx1:   {X, Y, V} = 3'b111;
    default:   {X, Y, V} = 3'b000;
   endcase
  end
endmodule

module t_priority_encoder_beh ();  // V2001
  wire X, Y, V;
  reg D0, D1, D2, D3;
  integer k;

  priority_encoder_beh M0 (X, Y, V, D0, D1, D2, D3);

  initial #200 $finish;
  initial begin
    k = 32'bx;
    #10 for (k = 0; k <= 16; k = k + 1)  #10 {D0, D1, D2, D3} = k;
  end
endmodule
```



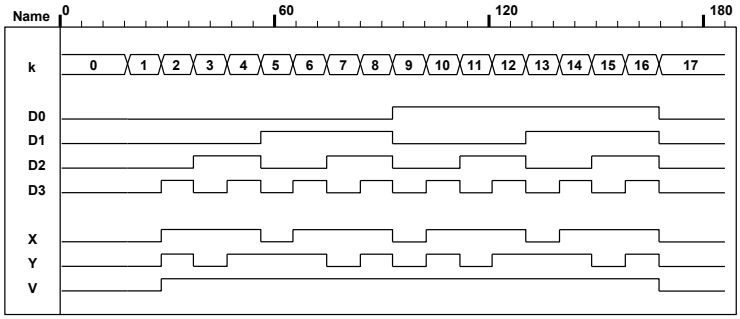
4.46 **(a)**

$F = \Sigma(0, 2, 5, 7, 11, 14)$
See code below.

(b) From prob 4.32:

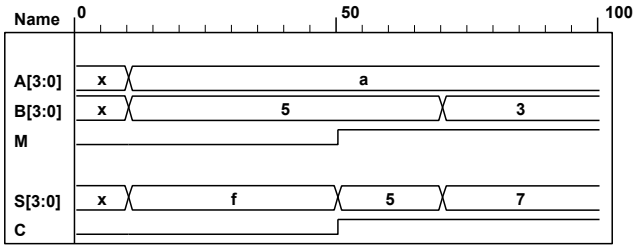
$F = \Pi(3, 8, 12) = (A' + B' + C + D)(A + B' + C' + D')(A + B + C' + D')$
$F' = ABC'D' + A'BCD + A'B'CD = \Sigma(12, 7, 3)$
$F = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 10, 11, 13, 14, 15)$

```
module Prob_4_46a (output F, input A, B, C, D);
assign F = (~A&~B&~C&~D) | (~A&~B&C&~D) | (~A&B&~C&D) | (~A&B&C&D) | (A&~B&C&D) |
(A&B&C&~D);
endmodule


module Prob_4_46b (output F, input A, B, C, D);
assign F = (~A&~B&~C&~D) | (~A&~B&~C&D) | (~A&~B&C&~D) | (~A&B&~C&~D) | (~A&B&~C&D) |
(~A&B&C&~D) | (A&~B&~C&~D) | (A&~B&~C&D) | (A&~B&C&~D) | (A&~B&C&D) | (A&B&~C&D) |
(A&B&C&~D) | (A&B&C&D);
endmodule

module t_Prob_4_46a ();
wire F_a, F_b;
reg A, B, C, D;
integer k;
  Prob_4_46a M0 (F_a, A, B, C, D);
  Prob_4_46b M1 (F_b, A, B, C, D);

 initial #200 $finish;
 initial begin
   k = 0;
   #10 repeat (15) begin {A, B, C, D} = k;  #10  k = k + 1; end
 end
endmodule
```



4.47

```
module Add_Sub_4_bit_Dataflow (
  output   [3: 0]       S,
  output        C, V,
  input    [3: 0]       A, B,
  input         M
);
  wire          C3;

  assign {C3, S[2: 0]} = A[2: 0] + ({M, M, M} ^ B[2: 0]) + M;
  assign {C, S[3]} = A[3] + M ^ B[3] + C3;
  assign V = C ^ C3;
endmodule

module t_Add_Sub_4_bit_Dataflow ();
  wire [3: 0] S;
  wire C, V;
  reg [3: 0] A, B;
  reg M;

  Add_Sub_4_bit_Dataflow M0 (S, C, V, A, B, M);

  initial #100 $finish;
  initial fork
    #10 M = 0;
    #10 A = 4'hA;
    #10 B = 4'h5;
    #50 M = 1;
    #70 B = 4'h3;
  join
endmodule
```



4.48

```
module ALU_3state (output [7: 0] y_tri, input [7: 0] A, B, input [2: 0] Sel, input En);
  reg [7: 0] y;
  assign y_tri = En ? y: 8'bz;
  always @ (A, B, Sel) begin
    y = 0;
    case (Sel)
      3'b000:    y = 8'b0;
      3'b001:    y = A & B;
      3'b010:    y = A | B;
      3'b011:    y = A ^ B;
      3'b100:    y = A + B;
      3'b101:    y = A - B;
      3'b110:    y = ~A;
      3'b111:    y = 8'hFF;
    endcase
  end
```

```
endmodule

module t_ALU_3state ();
  wire[7: 0] y;
  reg [7: 0] A, B;
  reg [2: 0] Sel;
  reg En;

  ALU_3state M0 (y, A, B, Sel, En);

initial #200 $finish;
  initial fork
     #5 En = 1;

     #5 begin A = 8'hAA; B = 8'h55; end      // Expect y = 8'd0
    #10 begin Sel = 3'b000; A = 8'hAA; B = 8'h55; end // y = 8'b000   Expect y = 8'd0
    #20 begin Sel = 3'b001; A = 8'hAA; B = 8'hAA; end // y = A & B    Expect y = 8'hAA = 8'1010_1010
    #30 begin Sel = 3'b001; A = 8'h55; B = 8'h55; end  // y = A & B    Expect y = 8'h55 = 8'b0101_0101
    #40 begin Sel = 3'b010; A = 8'h55; B = 8'h55; end  // y = A | B Expect y = 8'h55 = 8'b0101_0101
    #50 begin Sel = 3'b010; A = 8'hAA; B = 8'hAA; end // y = A | BExpect y = 8'hAA = 8'b1010_1010
    #60 begin Sel = 3'b011; A = 8'h55; B = 8'h55; end  // y = A ^ B    Expect y = 8'd0
    #70 begin Sel = 3'b011; A = 8'hAA; B = 8'h55; end // y = A ^ B    Expect y = 8'hFF = 8'b1111_1111
    #80 begin Sel = 3'b100; A = 8'h55; B = 8'h00; end  // y = A + B    Expect y = 8'h55 = 8'b0101_0101
    #90 begin Sel = 3'b100; A = 8'hAA; B = 8'h55; end // y = A + B    Expect y = 8'hFF = 8'b1111_1111
    #100 En = 0;
    #115 En = 1;
#110 begin Sel = 3'b101; A = 8'hAA; B = 8'h55; end   // y = A – B    Expect y = 8'h55 = 8'b0101_0101
  #120 begin Sel = 3'b101; A = 8'h55; B = 8'hAA; end // y = A – B    Expect y = 8'hab = 8'b1010_1011
  #130 begin Sel = 3'b110; A = 8'hFF; end              // y = ~A       Expect y = 8'd0
  #140 begin Sel = 3'b110; A = 8'd0; end               // y = ~A       Expect y = 8'hFF = 8'b1111_1111
  #150 begin Sel = 3'b110; A = 8'hFF; end              // y = ~A       Expect y = 8'd0
  #160 begin Sel = 3'b111;    end                      // y = 8'hFF    Expect y = 8'hFF = 8'b1111_1111
  join
endmodule
```

4.49

```
// See Problem 4.1
module Problem_4_49_Gates (output F1, F2, input A, B, C, D);
  wire A_bar = !A;
  wire B_bar = !B;

  and (T1, B_bar, C);
  and (T2, A_bar, B);
  or (T3, A, T1);
  xor (T4, T2, D);
  or (F1, T3, T4);
  or (F2, T2, D);
endmodule

module Problem_4_49_Boolean_1 (output F1, F2, input A, B, C, D);
  wire A_bar = !A;
  wire B_bar = !B;
  wire T1 = B_bar && C;
  wire T2 = A_bar && B;
  wire T3 = A || T1;
  wire T4 = T2 ^ D;
  assign F1 = T3 || T4;
  assign F2 = T2 || D;
endmodule

module Problem_4_49_Boolean_2(output F1, F2, input A, B, C, D);
  assign F1 = A || (!B && C) || (B && (!D)) || (!B && D);
  assign F2 = ((!A) && B) || D;
endmodule
```

```
module t_Problem_4_49;
  reg A, B, C, D;
  wire F1_Gates, F2_Gates;
  wire F1_Boolean_1, F2_Boolean_1;
  wire F1_Boolean_2, F2_Boolean_2;

  Problem_4_48_Gates       M0 (F1_Gates, F2_Gates, A, B, C, D);
  Problem_4_48_Boolean_1   M1 (F1_Boolean_1, F2_Boolean_1, A, B, C, D);
  Problem_4_48_Boolean_2   M2 (F1_Boolean_2, F2_Boolean_2, A, B, C, D);

  initial #100 $finish;
  integer K;
  initial begin
    for (K = 0; K < 16; K = K + 1) begin {A, B, C, D} = K; #5; end
  end
endmodule
```

4.50 (a) 84-2-1 to BCD code converter

```
// See Problem 4.8 and Table 1.5.
// Verilog 1995

// module Prob_4_50a (Code_BCD,  Code84_m2_m1);
// output [3: 0]   Code_BCD;
// input [3:0];
// reg [3: 0]   Code_BCD;
// ...

// Verilog 2001, 2005

module Prob_4_50a (output reg [3: 0] Code_BCD,  input [3: 0] Code_84_m2_m1);

  always @ (Code_84_m2_m1)        // always @ (A or B or C or D)
    case (Code_84_m2_m1)
      4'b0000: Code_BCD = 4'b0000;       // 0
      4'b0111: Code_BCD = 4'b0001;       // 1
      4'b0110: Code_BCD = 4'b0010;       // 2
      4'b0101: Code_BCD = 4'b0011;       // 3
      4'b0100: Code_BCD = 4'b0100;       // 4
      4'b1011: Code_BCD = 4'b0101;       // 5
      4'b1010: Code_BCD = 4'b0110;       // 6
      4'b1001: Code_BCD = 4'b0111;       // 7
      4'b1000: Code_BCD = 4'b1000;       // 8
      4'b1111: Code_BCD = 4'b1001;       // 9

      4'b0001: Code_BCD = 4'b1010;       // 10
      4'b0010: Code_BCD = 4'b1011;       // 11
      4'b0011: Code_BCD = 4'b1100;       // 12
      4'b1100: Code_BCD= 4'b1101;        // 13
      4'b1101: Code_BCD = 4'b1110;       // 14
      4'b1110: Code_BCD  = 4'b1111;      // 15
    endcase
endmodule

module t_Prob_4_50a;
  wire [3: 0]  Code_BCD;
  reg [3: 0]; Code_84_m2_m1;
  integer      K;

  Prob_4_50a M0 ( Code_BCD,  Code_84_m2_m1); // Unit under test (UUT)

  initial #100 $finish;
  initial begin
    for (K = 0; K < 16; K = K + 1) begin Code_84_m2_m1 = K; #5 ; end
  end
```

```
endmodule
```

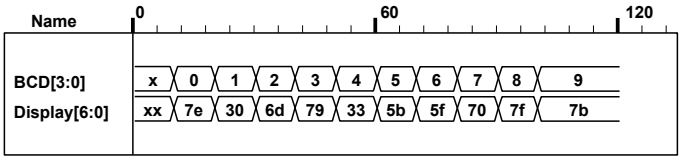
(b) 84-2-1 to Gray code converter

```
module Prob_4_50b (output reg [3: 0] Code_BCD,  input [3: 0] Code_84_m2_m1);

 always @ (Code_84_m2_m1)
  case (Code_84_m2_m1)
    4'b0000: Code_Gray = 4'b0000;       // 0
    4'b0111: Code_Gray = 4'b0001;       // 1
    4'b0110: Code_Gray = 4'b0011;       // 2
    4'b0101: Code_Gray = 4'b0010;       // 3
    4'b0100: Code_Gray = 4'b0110;       // 4
    4'b1011: Code_Gray = 4'b0111;       // 5
    4'b1010: Code_Gray = 4'b0101;       // 6
    4'b1001: Code_Gray = 4'b0100;       // 7
    4'b1000: Code_Gray = 4'b1100;       // 8
    4'b1111: Code_Gray = 4'b1101;       // 9

    4'b0001: Code_Gray = 4'b1111;       // 10
    4'b0010: Code_Gray = 4'b1110;       // 11
    4'b0011: Code_Gray = 4'b1010;       // 12
    4'b1100: Code_Gray= 4'b1011;        // 13
    4'b1101: Code_Gray = 4'b1001;       // 14
    4'b1110: Code_Gray  = 4'b1000;      // 15
  endcase
endmodule

module t_Prob_4_50b;
 wire [3: 0]  Code_Gray;
 reg [3: 0] Code_84_m2_m1;
 integer     K;

 Prob_4_50b M0 (Code_Gray,  Code_84_m2_m1); // Unit under test (UUT)

 initial #100 $finish;
 initial begin
  for (K = 0; K < 16; K = K + 1) begin Code_84_m2_m1 = K; #5 ; end
 end
endmodule
```

4.51 Assume that that the LEDs are asserted when the output is high.

```
module Seven_Seg_Display_V2001 (
 output reg  [6: 0]  Display,
 input       [3: 0]  BCD
);

 //                    abc_defg
 parameter  BLANK      = 7'b000_0000;
 parameter  ZERO       = 7'b111_1110;      // h7e
 parameter  ONE        = 7'b011_0000;      // h30
 parameter  TWO        = 7'b110_1101;      // h6d
 parameter  THREE      = 7'b111_1001;      // h79
 parameter  FOUR       = 7'b011_0011;      // h33
```

```
  parameter   FIVE          = 7'b101_1011;      // h5b
  parameter   SIX           = 7'b101_1111;      // h5f
  parameter   SEVEN         = 7'b111_0000;      // h70
  parameter   EIGHT         = 7'b111_1111;      // h7f
  parameter   NINE          = 7'b111_1011;      // h7b

  always @ (BCD)
    case (BCD)
      0:      Display = ZERO;
      1:      Display = ONE;
      2:      Display = TWO;
      3:      Display = THREE;
      4:      Display = FOUR;
      5:      Display = FIVE;
      6:      Display = SIX;
      7:      Display = SEVEN;
      8:      Display = EIGHT;
      9:      Display = NINE;
      default:    Display = BLANK;
  endcase
endmodule

module t_Seven_Seg_Display_V2001 ();
  wire  [6: 0]   Display;
  reg   [3: 0]   BCD;

  parameter   BLANK         = 7'b000_0000;
  parameter   ZERO          = 7'b111_1110;      // h7e
  parameter   ONE           = 7'b011_0000;      // h30
  parameter   TWO           = 7'b110_1101;      // h6d
  parameter   THREE         = 7'b111_1001;      // h79
  parameter   FOUR          = 7'b011_0011;      // h33
  parameter   FIVE          = 7'b101_1011;      // h5b
  parameter   SIX           = 7'b001_1111;      // h1f
  parameter   SEVEN         = 7'b111_0000;      // h70
  parameter   EIGHT         = 7'b111_1111;      // h7f
  parameter   NINE          = 7'b111_1011;      // h7b

  initial #120 $finish;
  initial fork
    #10 BCD = 0;
    #20 BCD = 1;
    #30 BCD = 2;
    #40 BCD = 3;
    #50 BCD = 4;
    #60 BCD = 5;
    #70 BCD = 6;
    #80 BCD = 7;
    #90 BCD = 8;
    #100 BCD = 9;

  join

  Seven_Seg_Display_V2001 M0 (Display, BCD);
endmodule
```



Alternative with continuous assignments (dataflow):

```
module Seven_Seg_Display_V2001_CA (
  output      [6: 0]   Display,
  input       [3: 0]   BCD
);

  //                     abc_defg
  parameter BLANK    = 7'b000_0000;
  parameter ZERO     = 7'b111_1110;      // h7e
  parameter ONE      = 7'b011_0000;      // h30
  parameter TWO      = 7'b110_1101;      // h6d
  parameter THREE    = 7'b111_1001;      // h79
  parameter FOUR     = 7'b011_0011;      // h33
  parameter FIVE     = 7'b101_1011;      // h5b
  parameter SIX      = 7'b101_1111;      // h5f
  parameter SEVEN    = 7'b111_0000;      // h70
  parameter EIGHT    = 7'b111_1111;      // h7f
  parameter NINE     = 7'b111_1011;      // h7b
  wire        A, B, C, D, a, b, c, d, e, f, g;

  assign A = BCD[3];
  assign B = BCD[2];
  assign C = BCD[1];
  assign D = BCD[0];
  assign Display = {a,b,c,d,e,f,g};
  assign a  = (~A)&C | (~A)&B&D | (~B)&(~C)&(~D) | A & (~B)&(~C);
  assign b = (~A)&(~B) | (~A)&(~C)&(~D) | (~A)&C&D | A&(~B)&(~C);
  assign c = (~A)&B | (~A)&D | (~B)&(~C)&(~D) | A&(~B)&(~C);
  assign d = (~A)&C&(~D) | (~A)&(~B)&C | (~B)&(~C)&(~D) | A&(~B)&(~C) | (~A)&B&(~C)&D;
  assign e = (~A)&C&(~D) | (~B)&(~C)&(~D);
  assign f = (~A)&B&(~C) | (~A)&(~C)&(~D) | (~A)&B&(~D) | A&(~B)&(~C);
  assign g = (~A)&C&(~D) | (~A)&(~B)&C | (~A)&B&(~C) | A&(~B)&(~C);
endmodule

module t_Seven_Seg_Display_V2001_CA ();
  wire     [6: 0]   Display;
  reg      [3: 0]   BCD;

  parameter      BLANK          = 7'b000_0000;
  parameter ZERO     = 7'b111_1110;      // h7e
  parameter ONE      = 7'b011_0000;      // h30
  parameter TWO      = 7'b110_1101;      // h6d
  parameter THREE    = 7'b111_1001;      // h79
  parameter FOUR     = 7'b011_0011;      // h33
  parameter FIVE     = 7'b101_1011;      // h5b
  parameter SIX      = 7'b001_1111;      // h1f
  parameter SEVEN    = 7'b111_0000;      // h70
  parameter EIGHT    = 7'b111_1111;      // h7f
  parameter NINE     = 7'b111_1011;      // h7b

  initial #120 $finish;
  initial fork
```

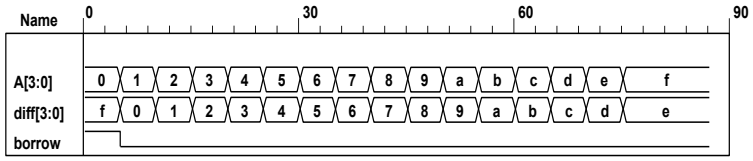
```
    #10 BCD = 0;
    #20 BCD = 1;
    #30 BCD = 2;
    #40 BCD = 3;
    #50 BCD = 4;
    #60 BCD = 5;
    #70 BCD = 6;
    #80 BCD = 8;
    #90 BCD = 8;
    #100 BCD = 9;

  join

   Seven_Seg_Display_V2001_CA M0 (Display,   BCD);
endmodule
```

4.52 **(a)** Incrementer for unsigned 4-bit numbers

```
module Problem_4_52a_Data_Flow  (output [3: 0] sum, output carry, input [3: 0] A);
  assign {carry, sum} = A + 1;
endmodule

module t_Problem_4_52a_Data_Flow;
  wire [3: 0]  sum;
  wire         carry;
  reg [3: 0]   A;

  Problem_4_52a_Data_Flow  M0 (sum, carry, A);

  initial # 100 $finish;
  integer K;
  initial begin
    for (K = 0; K < 16; K = K + 1) begin A = K; #5; end
  end
endmodule
```

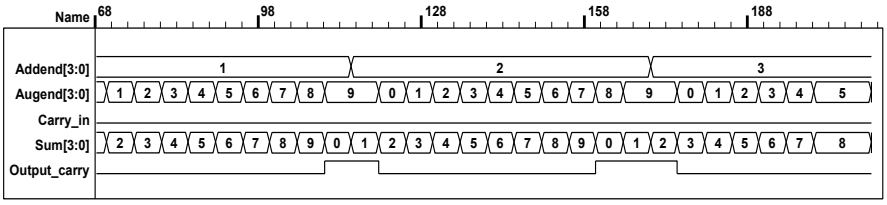
(b) Decrementer for unsigned 4-bit numbers

```
module Problem_4_52b_Data_Flow  (output [3: 0] diff, output borrow, input [3: 0] A);
  assign {borrow, diff} = A - 1;
endmodule

module t_Problem_4_52b_Data_Flow;
  wire [3: 0]      diff;
  wire             borrow;
  reg [3: 0]       A;

  Problem_4_52b_Data_Flow  M0 (diff, borrow, A);

  initial # 100 $finish;
  integer K;
  initial begin
    for (K = 0; K < 16; K = K + 1) begin A = K; #5; end
  end
endmodule
```



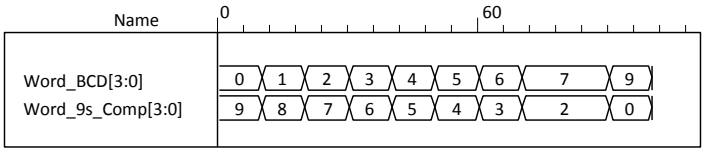
4.53 // BCD Adder

```
module Problem_4_53_BCD_Adder (
  output       Output_carry,
  output [3: 0] Sum,
   input [3: 0]  Addend, Augend,
   input         Carry_in);
  supply0      gnd;
  wire [3: 0]   Z_Addend;
  wire          Carry_out;
  wire          C_out;
  assign  Z_Addend = {1'b0, Output_carry, Output_carry, 1'b0};
  wire [3: 0] Z_sum;

  and (w1, Z_sum[3], Z_sum[2]);
  and (w2, Z_sum[3], Z_sum[1]);
  or (Output_carry, Carry_out, w1, w2);

  Adder_4_bit M0 (Carry_out, Z_sum, Addend, Augend, Carry_in);
  Adder_4_bit M1 (C_out, Sum, Z_Addend, Z_sum, gnd);
endmodule

module Adder_4_bit (output carry, output [3:0] sum, input [3: 0] a, b, input c_in);
  assign {carry, sum} = a + b + c_in;
endmodule

module t_Problem_4_53_Data_Flow;
  wire [3: 0]   Sum;
  wire          Output_carry;
  reg [3: 0]    Addend, Augend;
  reg           Carry_in;

  Problem_4_53_BCD_Adder M0 (Output_carry, Sum, Addend, Augend, Carry_in);

  initial # 1500 $finish;
  integer i, j, k;
  initial begin
    for (i = 0; i <= 1; i = i + 1) begin Carry_in = i; #5;
      for (j = 0; j <= 9; j = j +1) begin Addend = j; #5;
        for (k = 0; k <= 9; k = k + 1) begin Augend = k; #5;
       end
      end
    end
  end
endmodule
```



4.54 **(a)** 9s Complement of BCD

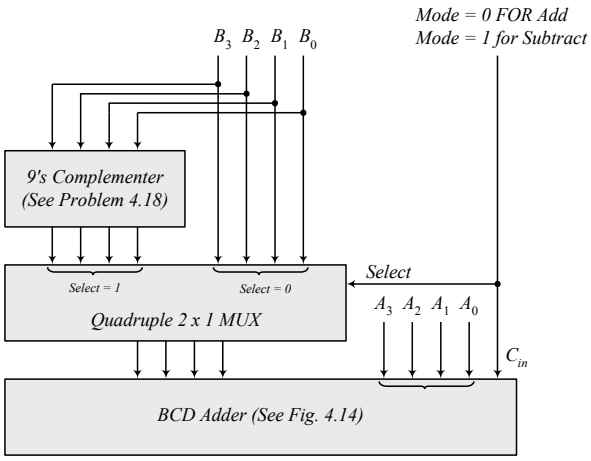
```
module Nines_Complementer (                 // V2001
  output reg   [3: 0]   Word_9s_Comp,
  input        [3: 0]   Word_BCD
);
  always @ (Word_BCD) begin
    Word_9s_Comp = 4'b0;
    case (Word_BCD)
      4'b0000:   Word_9s_Comp = 4'b1001;      // 0 to 9
      4'b0001:   Word_9s_Comp = 4'b1000;      // 1 to 8
      4'b0010:   Word_9s_Comp = 4'b0111;      // 2 to 7
      4'b0011:   Word_9s_Comp = 4'b0110;      // 3 to 6
      4'b0100:   Word_9s_Comp = 4'b0101;      // 4 to 5
      4'b0101:   Word_9s_Comp = 4'b0100;      // 5 to 4
      4'b0110:   Word_9s_Comp = 4'b0011;      // 6 to 3
      4'b0111:   Word_9s_Comp = 4'b0010;      // 7 to 2
      4'b1000:   Word_9s_Comp = 4'b0001;      // 8 to 1
      4'b1001:   Word_9s_Comp = 4'b0000;      // 9 to 0
      default:   Word_9s_Comp = 4'b1111;      // Error detection
    endcase
  end
endmodule

module t_Nines_Complementer ();
  wire  [3: 0]   Word_9s_Comp;
  reg   [3: 0]   Word_BCD;

  Nines_Complementer M0 (Word_9s_Comp, Word_BCD);

  initial #11$finish;
  initial fork
        Word_BCD = 0;
    #10 Word_BCD = 1;
    #20 Word_BCD = 2;
    #30 Word_BCD = 3;
    #40 Word_BCD = 4;
    #50 Word_BCD = 5;
    #60 Word_BCD = 6;
    #70 Word_BCD = 7;
    #20 Word_BCD = 8;
    #90 Word_BCD = 9;
    #100 Word_BCD = 4'b1100;        // Confirm error detection
  join
endmodule
```



(b) 9s complement of Gray Code

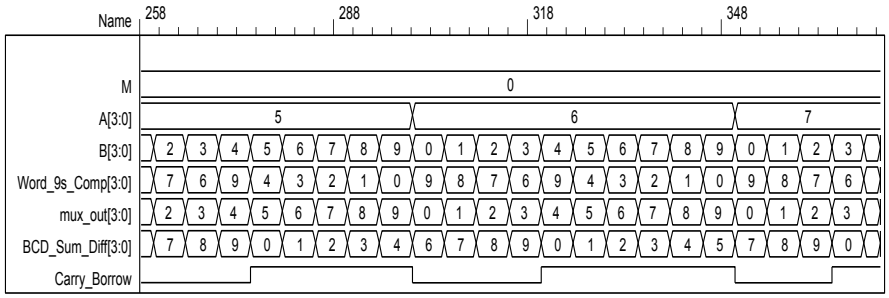
```
module Nines_Complementer (                 // V2001
  output reg   [3: 0]   Word_9s_Comp,
  input        [3: 0]   Word_Gray
);
  always @ (Word_Gray) begin
    Word_9s_Comp = 4'b0;
    case (Word_BCD)
      4'b0000:   Word_9s_Comp = 4'b1101;       // 0 to 9
      4'b0001:   Word_9s_Comp = 4'b1100;       // 1 to 8
      4'b0010:   Word_9s_Comp = 4'b0100;       // 2 to 7
      4'b0011:   Word_9s_Comp = 4'b0101;       // 3 to 6
      4'b0100:   Word_9s_Comp = 4'b0111;       // 4 to 5
      4'b0101:   Word_9s_Comp = 4'b0110;       // 5 to 4
      4'b0110:   Word_9s_Comp = 4'b0010;       // 6 to 3
      4'b0111:   Word_9s_Comp = 4'b0011;       // 7 to 2
      4'b1000:   Word_9s_Comp = 4'b0001;       // 8 to 1
      4'b1001:   Word_9s_Comp = 4'b0000;       // 9 to 0
      default:   Word_9s_Comp = 4'b1111;       // Error detection
    endcase
  end
endmodule

module t_Nines_Complementer ();
  wire  [3: 0]   Word_9s_Comp;
  reg   [3: 0]   Word_Gray;

  Nines_Complementer M0 (Word_9s_Comp, Word_Gray);

  initial #11$finish;
  initial fork
          Word_Gray = 0;
    #10 Word_Gray = 1;
    #20 Word_Gray = 2;
    #30 Word_Gray = 3;
    #40 Word_Gray = 4;
    #50 Word_Gray = 5;
    #60 Word_Gray = 6;
    #70 Word_Gray = 7;
    #20 Word_Gray = 8;
    #90 Word_Gray = 9;
    #100 Word_Gray = 4'b1100;        // Confirm error detection
  join
endmodule
```

4.55 From Problem 4.19:

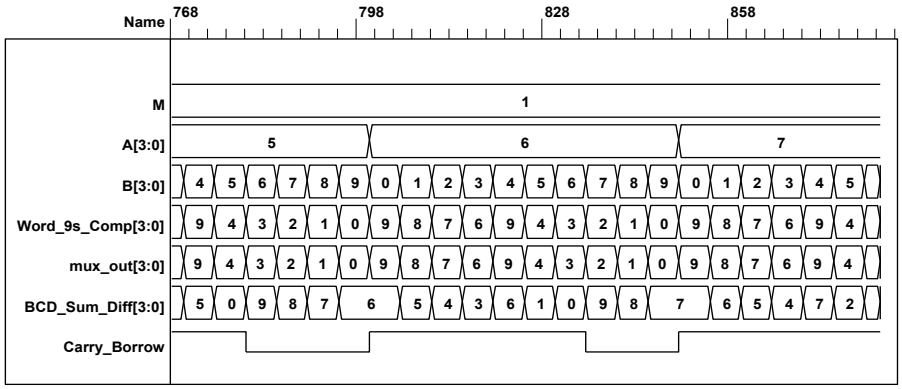



```
// BCD Adder – Subtractor
module Problem_4_55_BCD_Adder_Subtractor (
  output [3: 0]     BCD_Sum_Diff,
  output            Carry_Borrow,
  input [3: 0]      B, A,
  input             Mode

);
  wire [3: 0] Word_9s_Comp, mux_out;
  Nines_Complementer   M0 (Word_9s_Comp, B);
  Quad_2_x_1_mux       M2 (mux_out, Word_9s_Comp, B, Mode);
  BCD_Adder            M1 (Carry_Borrow,  BCD_Sum_Diff, mux_out, A, Mode);
endmodule

module Nines_Complementer (               // V2001
  output reg   [3: 0]  Word_9s_Comp,
  input        [3: 0]  Word_BCD
);
  always @ (Word_BCD) begin
    Word_9s_Comp = 4'b0;
    case (Word_BCD)
      4'b0000:   Word_9s_Comp = 4'b1001;      // 0 to 9
      4'b0001:   Word_9s_Comp = 4'b1000;      // 1 to 8
      4'b0010:   Word_9s_Comp = 4'b0111;      // 2 to 7
      4'b0011:   Word_9s_Comp = 4'b0110;      // 3 to 6
      4'b0100:   Word_9s_Comp = 4'b1001;      // 4 to 5
      4'b0101:   Word_9s_Comp = 4'b0100;      // 5 to 4
      4'b0110:   Word_9s_Comp = 4'b0011;      // 6 to 3
      4'b0111:   Word_9s_Comp = 4'b0010;      // 7 to 2
      4'b1000:   Word_9s_Comp = 4'b0001;      // 8 to 1
      4'b1001:   Word_9s_Comp = 4'b0000;      // 9 to 0
      default:   Word_9s_Comp = 4'b1111;          // Error detection
    endcase
  end
endmodule
```

```
module Quad_2_x_1_mux (output reg [3: 0] mux_out, input [3: 0] b, a,  input select);
  always @ (a, b,  select)
    case (select)
      0:  mux_out = a;
      1:  mux_out = b;
    endcase
endmodule

module BCD_Adder (
  output            Output_carry,
  output    [3: 0]  Sum,
  input     [3: 0]  Addend, Augend,
  input             Carry_in);
  supply0           gnd;
  wire      [3: 0]  Z_Addend;
  wire              Carry_out;
  wire              C_out;
  assign  Z_Addend = {1'b0, Output_carry, Output_carry, 1'b0};
  wire [3: 0] Z_sum;

  and (w1, Z_sum[3], Z_sum[2]);
  and (w2, Z_sum[3], Z_sum[1]);
  or (Output_carry, Carry_out, w1, w2);

  Adder_4_bit M0 (Carry_out, Z_sum, Addend, Augend, Carry_in);
  Adder_4_bit M1 (C_out, Sum, Z_Addend, Z_sum, gnd);
endmodule

module Adder_4_bit (output carry, output [3:0] sum, input [3: 0] a, b, input c_in);
  assign {carry, sum} = a + b + c_in;
endmodule

module t_Problem_4_55_BCD_Adder_Subtractor();
  wire  [3: 0]  BCD_Sum_Diff;
  wire          Carry_Borrow;
  reg   [3: 0]  B, A;
  reg           Mode;

  Problem_4_55_BCD_Adder_Subtractor M0 (BCD_Sum_Diff, Carry_Borrow, B, A, Mode);

  initial #1000 $finish;

  integer J, K, M;
  initial begin
    for (M = 0; M < 2; M = M + 1) begin
      for (J = 0; J < 10; J = J + 1) begin
        for (K = 0; K < 10; K = K + 1) begin
          A = J; B = K; Mode = M; #5 ;
       end
      end
    end
  end
endmodule
```



Note: For subtraction, Carry_Borrow = 1 indicates a positive result; Carry_Borrow = 0 indicates a negative result.



4.56

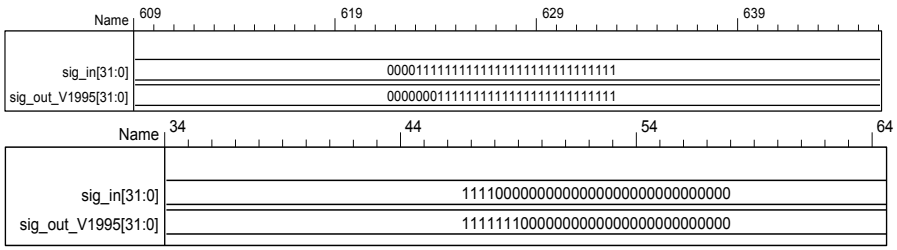
```
assign match = (A == B);     // Assumes reg [3: 0] A, B;
```

4.57

```
// Priority encoder (See Problem 4.29)
// Caution: do not confuse logic value x with identifier x.
// Verilog 1995

module Prob_4_57 (x, y, v, D3, D2, D1, D0);
output   x, y, v;
input    D3, D2, D1, D0;
reg      x, y, v;
...

// Verilog 2001, 2005

module Prob_4_57 (output reg x, y, v, input D3, D2, D1, D0);
 always @ (D3, D2, D1, D0) begin     // always @ (D3 or D2 or D1 or D0)
   x = 0;
   y = 0;
   v = 0;
   casex ({D3, D2, D1, D0})
     4'b0000: {x, y, v} = 3'bxx0;
     4'bxxx1: {x, y, v} = 3'b001;
     4'bxx10: {x, y, v} = 3'b011;
     4'bx100: {x, y, v} = 3'b101;
     4'b1000: {x, y, v} = 3'b110;
   endcase
 end
endmodule

module t_Prob_4_57;
 wire        x, y, v;
 reg     D3, D2, D1, D0;
 integer     K;
 Prob_4_57 M0 (x, y, v, D3, D2, D1, D0);
 initial #100 $finish;
 initial begin
   for (K = 0; K < 16; K = K + 1) begin {D3, D2, D1, D0} = K; #5 ; end
 end
endmodule
```

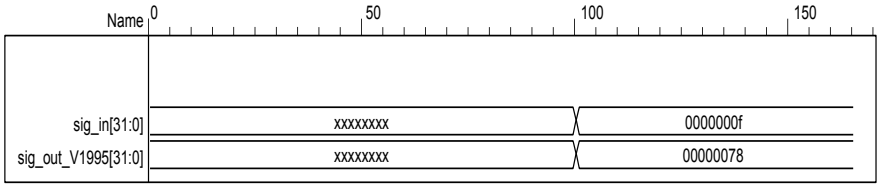
4.58 **(a)**

```
//module shift_right_by_3_V2001 (output [31: 0]  sig_out, input [31: 0] sig_in);
 // assign sig_out = sig_in >>> 3;
//endmodule

module shift_right_by_3_V1995 (output reg [31: 0]  sig_out, input [31: 0] sig_in);
  always @ (sig_in)
    sig_out = {sig_in[31], sig_in[31], sig_in[31],  sig_in[31: 3]};
endmodule


module t_shift_right_by_3 ();
  wire [31: 0]  sig_out_V1995;
  wire [31: 0]  sig_out_V2001;

  reg [31: 0] sig_in;

  //shift_right_by_3_V2001 M0 (sig_out_V2001, sig_in);

  shift_right_by_3_V1995 M1 (sig_out_V1995, sig_in);
  integer k;
  initial #1000 $finish;
  initial begin
    sig_in = 32'hf000_0000;
    #100 sig_in = 32'h8fff_ffff;
    #500 sig_in = 32'h0fff_ffff;
  end
endmodule
```



(b)

```
//module shift_left_by_3_V2001 (output [31: 0]  sig_out, input [31: 0] sig_in);
  assign sig_out = sig_in <<< 3;

//module shift_left_by_3_V1995 (output reg [31: 0]  sig_out, input [31: 0] sig_in);
  //always @ (sig_in)
   // sig_out = {sig_in[31: 3], 3'b0};
endmodule


module t_shift_left_by_3 ();
  wire [31: 0]  sig_out_V1995;
  wire [31: 0]  sig_out_V2001;

  reg [31: 0] sig_in;

  shift_left_by_3_V2001 M0 (sig_out_V2001, sig_in);

integer k;
  initial #1000 $finish;
  initial begin
    sig_in = 32'hf000_0000;
    #100 sig_in = 32'h8fff_ffff;
    #500 sig_in = 32'h0fff_ffff;
  end
endmodule
```



4.59

```
module BCD_to_Decimal (output reg [3: 0] Decimal_out, input [3: 0] BCD_in);
  always @ (BCD_in) begin
    Decimal_out = 0;
    case (BCD_in)
      4'b0000:   Decimal_out = 0;
      4'b0001:   Decimal_out = 1;
      4'b0010:   Decimal_out = 2;
      4'b0011:   Decimal_out = 3;
      4'b0100:   Decimal_out = 4;
      4'b0101:   Decimal_out = 5;
      4'b0110:   Decimal_out = 6;
      4'b0111:   Decimal_out = 7;
      4'b1000:   Decimal_out = 8;
      4'b1001:   Decimal_out = 9;
      default:   Decimal_out = 4'bxxxx;
    endcase
  end
endmodule
```

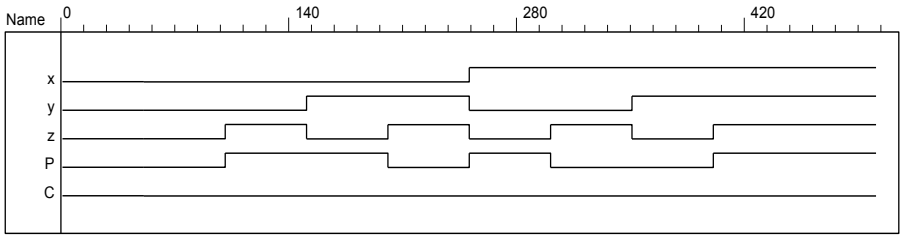
4.60

```
module Even_Parity_Checker_4 (output  P, C, input x, y, z);
  xor (w1, x,  y);
  xor (P, w1, z);
  xor (C, w1, w2);
  xor (w2, z, P);
endmodule
```

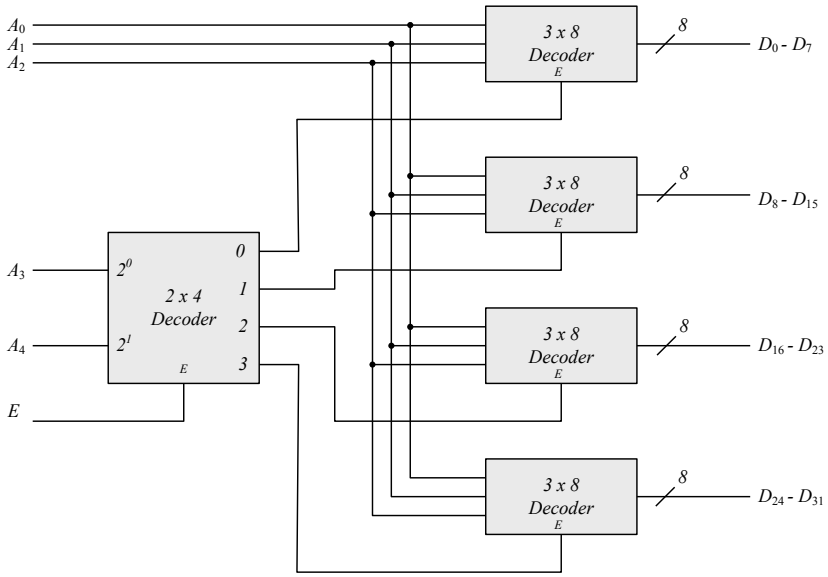
See Problem 4.62 for testbench and waveforms.

4.61

```
module Even_Parity_Checker_4 (output  P, C, input x, y, z);
  assign w1 = x ^ y;
  assign P = w1 ^ z;
  assign C = w1 ^ w2;
  assign w2 = z ^ P;
endmodule
```

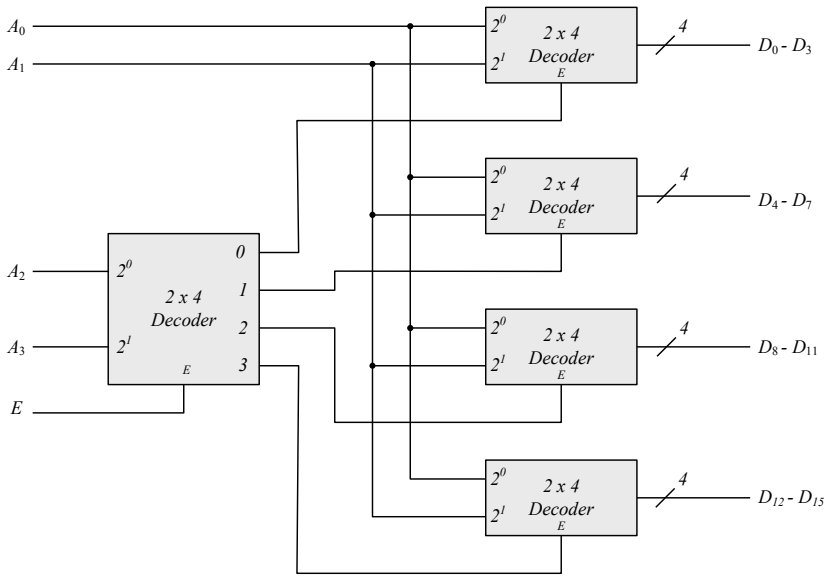


4.62




```
module Decoder_3x8 (output D7, D6, D5, D4, D3, D2, D1, D0, input in2, in1, in0, E);
  not (in2_bar, in2);
  not (in1_bar, in1);
  not (in0_bar, in0);
  and (D0, in2_bar, in1_bar, in0_bar, E);
  and (D1, in2_bar, in1_bar, in0, E);
  and (D2, in2_bar, in1, in0_bar, E);
  and (D3, in2_bar, in1, in0, E);
  and (D4, in2, in1_bar, in0_bar, E);
  and (D5, in2, in1_bar, in0, E);
  and (D6, in2, in1, in0_bar, E);
  and (D7, in2, in1, in0, E);
endmodule

module Decoder_5x32 (
  output D31, D30, D29, D28, D27, D26, D25, D24, D23, D22, D21, D20, D19, D18, D17, D16,
         D15, D14, D13, D12, D11, D10, D9, D8, D7, D6, D5, D4, D3, D2, D1, D0,
  input A4, A3, A2, A1, A0, E;
  wire E3, E2, E1, E0;
  Decoder_3x8 M0 (D7, D6, D5, D4, D3, D2, D1, D0, A2, aA1, A0, E0);
  Decoder_3x8 M1 (D15, D14, D13, D12, D11, D10, D9, D8, A2, A1, A0, E1);
  Decoder_3x8 M2 (D23, D22, D21, D20, D19, D18, D17, D16, in2, in1, in0, E2);
  Decoder_3x8 M3 (D31, D30, D29, D28, D27, D26, D25, D24, A2, A1, A0, E3);
  Decoder_2x4 M4 (E3, E2, E1, E0, A4, A3, E);
endmodule
```

4.63




```
module Decoder_2x4 (output D3, D2, D1, D0, input in1, in0, E);
  not (in1_bar, in1);
  not (in0_bar, in0);
  and (D0, in1_bar, in0_bar, E);
  and (D1, in1_bar, in0, E);
  and (D2, in1, in0_bar, E);
  and (D3, in1, in0, E);
endmodule

module Decoder_4x16 (
  output D15, D14, D13, D12, D11, D10, D9, D8, D7, D6, D5, D4, D3, D2, D1, D0,
  input A3, A2, A1, A0, E);
  wire E3, E2, E1, E0;
  Decoder_2x4 M0 (output D3, D2, D1, D0, input in1, in0, E0);
  Decoder_2x4 M1 (output D7, D6, D5, D4, input in1, in0, E1);
  Decoder_2x4 M2 (output D11, D10, D9, D8, input in1, in0, E2);
  Decoder_2x4 M3 (output D15, D14, D13, D12, input in1, in0, E3);
  Decoder_2x4 M4 (output E3, E2, E1, E0, input A3, A2, E);
endmodule
```

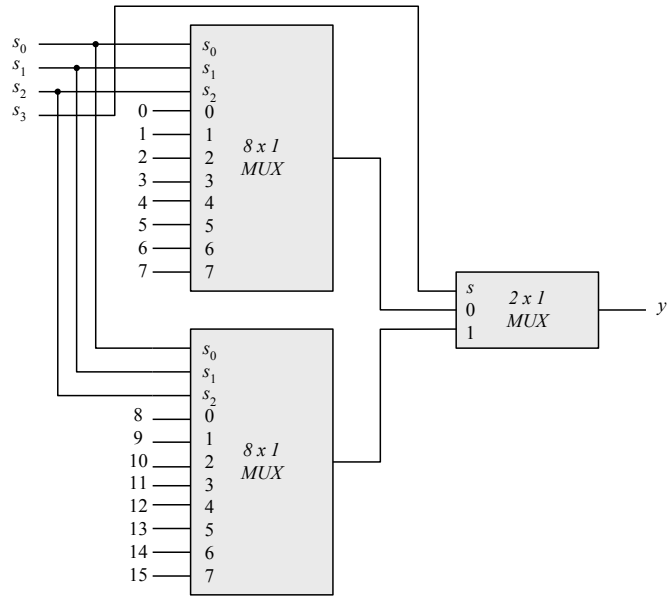
4.64

Inputs								*Outputs*			
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	*x*	*y*	*z*	*V*
0	0	0	0	0	0	0	0	x	x	x	0
1	0	0	0	0	0	0	0	0	0	0	1
x	1	0	0	0	0	0	0	0	0	1	1
x	x	1	0	0	0	0	0	0	1	0	1
x	x	x	1	0	0	0	0	0	1	1	1
x	x	x	x	1	0	0	0	1	0	0	1
x	x	x	x	x	1	0	0	1	0	1	1
x	x	x	x	x	x	1	0	1	0	0	1
x	x	x	x	x	x	x	1	1	1	1	1

If D_2 = 1, D_6 = 1, all others = 0
Output xyz = 100 and V = 1

```
module Prob_4_64 (output x, y, x, V, input, D0, D1, D2, D3, D4,D5 D6, D7);

  always @( D0, D1, D2, D3, D4,D5 D6, D7)
   case({D0, D1, D2, D3, D4,D5 D6, D7})
     8'b0000_0000:    {x, y, x, V} = 4'bxxx0;
     8'b1000_0000:    {x, y, x, V} = 4'b0001;
     8'b0100_0000:    {x, y, x, V} = 4'b0011;
     8'b0010_0000:    {x, y, x, V} = 4'b0101;

     8'b0001_0000:    {x, y, x, V} = 4'b0111;
     8'b0000_1000:    {x, y, x, V} = 4'b1001;
     8'b0000_0100:    {x, y, x, V} = 4'b1011;
     8'b0000_0010:    {x, y, x, V} = 4'b1001;
     8'b0000_0001:    {x, y, x, V} = 4'b1111;
     default:         {x, y, x, V} = 4'b1010;        // Use for error detection
   endcase
endmodule
```

4.65




```
module Mux_2x1 (
  output y_out,
  input in1, in0, sel);
  not (sel_bar, sel);
  and (y0, in0, sel);
  and (y1, in1, sel);
  or (y_out, in0, in1, sel_bar
);
endmodule

module Mux_4x1 (
  output y_out,
  input in3, in2, in1, in0, sel1, sel0);
  not (sel_1_bar, sel1);
  and (s0, sel_1_bar, sel0);
  and (s1, sel[1], sel0);
  Mux_2x1 M0 (y_M0, in0, in1, s0);
  Mux_2x1   M1 (y_M1, in2, in3, s1);
  or (y_out, y_M0, y_M1
);
endmodule

module Mux_8x1 (
  output y_out,
  input in7, in6, in5, in4, in3, in2, in1, in0, sel2, sel1, sel0
);
  Mux_4x1 M0 (y_M0,  in3, in2, in1, in0, sel1, sel0);
  Mux_4x1 M1 (y_M1, in7, in6, in5, in4, sl1, sel0);
  Mux_2x1 M2 (y_out, y_M0, y_M1, sel2);
endmodule

module Mux_16x1 (
  output y_out,
```

```
  input in15, in14, in13, in12, in11, in10, in9, in8, in7, in6, in5, in4, in3, in2, in1, in0, sel3, sel2, sel1, sel0
);
  Mux_8x1 M0 (y_M0, in7, in6, in5, in4, in3, in2, in1, in0, sel2, sel1, sel0);
  Mux_8x1 M1 (y_M1, in15, in14, in13, in12, in11, in10, in9, in8, sel2, sel1, sel0);
  Mux_2x1 M2 (y_out, y_M0, y_M1, sel3);
endmodule
```