

# **SOLUTIONS MANUAL**

## **PART 1: CHAPTERS 1-5**

Rev 06/05/2017

# **DIGITAL DESIGN**

**WITH AN INTRODUCTION to the VERILOG HDL,  
VHDL, and SystemVerilog  
Sixth Edition**

**M. MORRIS MANO**  
Professor Emeritus  
California State University, Los Angeles

**MICHAEL D. CILETTI**  
Professor Emeritus

**University of Colorado, Colorado Springs**

Note: Solutions to problems requiring HDL code are presented in Verilog and VHDL

## CHAPTER 1

**1.1** Base-10: 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32  
 Octal: 16 17 20 21 22 23 24 25 26 27 30 31 32 33 34 35 36 37 40  
 Hex: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20

Base-10: 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

Base-12 8 9 0A 0B 10 11 12 13 14 15 16 17 18 19 1A 1B 20 21 22 23

24

**1.2** (a) 32,768 (b) 67,108,864 (c) 6,871,947,674

**1.3**  $(4310)_5 = 4 * 5^3 + 3 * 5^2 + 1 * 5^1 = 580_{10}$

$(198)_{12} = 1 * 12^2 + 9 * 12^1 + 8 * 12^0 = 260_{10}$

$(445)_8 = 4 * 8^2 + 4 * 8^1 + 5 * 8^0 = 293_{10}$

$(345)_6 = 3 * 6^2 + 4 * 6^1 + 5 * 6^0 = 137_{10}$

**1.4** 16-bit binary: 1111\_1111\_1111\_1111

Decimal equivalent:  $2^{16} - 1 = 65,535_{10}$

Hexadecimal equivalent: FFFF<sub>16</sub>

**1.5** Let b = base

(a)  $14/2 = (b + 4)/2 = 5$ , so b = 6

(b)  $56/4 = (5*b + 6)/4 = 15 = 1*b + 5$ , so  $5*b + 6 = 4*(1*b + 5) = 4*b + 20$  so b = 14

(c)  $32 + 12 = 28$ ,  $3*b + 2 + 1*b + 2 = 2*b + 8$   
 $4*b + 4 = 2*b + 8$ ,  $2*b = 4$ , so b = 2

**1.6**  $(x - 3)(x - 6) = x^2 - (6 + 3)x + 6*3 = x^2 - 11x + 22$

Therefore:  $6 + 3 = b + 1$ , so b = 8

Also,  $6*3 = (18)_{10} = (22)_8$

**1.7**  $64CD_{16} = 0110\_0100\_1100\_1101_2 = 110\_010\_011\_001\_101 = (62315)_8$

**1.8 (a)** Results of repeated division by 2 (quotients are followed by remainders):

$$431_{10} = 215(1); 107(1); 53(1); 26(1); 13(0); 6(1) \ 3(0) \ 1(1)$$

$$\text{Answer: } 1111\_1010_2 = FA_{16}$$

**(b)** Results of repeated division by 16:

$$431_{10} = 26(15); 1(10) \text{ (Faster)}$$

$$\text{Answer: } FA = 1111\_1010$$

**1.9 (a)**  $10110.0101_2 = 16 + 4 + 2 + .25 + .0625 = 22.3125$

$$\text{(b)} \quad 16.5_{16} = 16 + 6 + 5*(.0615) = 22.3125$$

$$\text{(c)} \quad 26.24_8 = 2 * 8 + 6 + 2/8 + 4/64 = 22.3125$$

$$\text{(d)} \quad DABA.B_{16} = 13*16^3 + 10*16^2 + 11*16 + 10 + 11/16 = 55,994.6875$$

$$\text{(e)} \quad 1011.1001_2 = 8 + 2 + 1 + .5 + .0625 = 11.5625$$

**1.10 (a)**  $1.10010_2 = 0001.1001_2 = 1.9_{16} = 1 + 9/16 = 1.563_{10}$

$$\text{(b)} \quad 110.010_2 = 0110.0100_2 = 6.4_{16} = 6 + 4/16 = 6.25_{10}$$

Reason:  $110.010_2$  is the same as  $1.10010_2$  shifted to the left by two places.

$$\begin{array}{r} 1011.11 \\ 101 \overline{) 111011.0000} \\ \underline{101} \phantom{0000} \\ 01001 \phantom{000} \\ \underline{101} \phantom{000} \\ 1001 \phantom{000} \\ \underline{101} \phantom{000} \\ 1000 \phantom{000} \\ \underline{101} \phantom{000} \\ 0110 \end{array}$$

The quotient is carried to two decimal places, giving  $1011.11$

$$\text{Checking: } 111011_2 / 101_2 = 59_{10} / 5_{10} \approx 1011.11_2 = 58.75_{10}$$

**1.12 (a)** 10000 and 110111

$$1011$$

$$1011$$

$$\begin{array}{r} +101 \\ 10000 = 16_{10} \end{array}$$

$$\begin{array}{r} \times 101 \\ 1011 \\ 1011 \\ \hline 110111 = 55_{10} \end{array}$$

(b)  $62_h$  and  $958_h$

$$\begin{array}{r} 2E_h \quad 0010\_1110 \\ +34_h \quad 0011\_0100 \\ \hline 62_h \quad 0110\_0010 = 98_{10} \end{array}$$

$$\begin{array}{r} 2E_h \\ \times 34_h \\ \hline B^3 8 \\ 8^2 A \\ \hline 9 \ 5 \ 8_h = 2392_{10} \end{array}$$

1.13 (a) Convert 27.315 to binary:

	Integer	Remainder	Coefficient
	Quotient		
27/2 =	13	+	$\frac{1}{2}$ $a_0 = 1$
13/2	6	+	$\frac{1}{2}$ $a_1 = 1$
6/2	3	+	0 $a_2 = 0$
3/2	1	+	$\frac{1}{2}$ $a_3 = 1$
$\frac{1}{2}$	0	+	$\frac{1}{2}$ $a_4 = 1$



$$27_{10} = 11011_2$$

	Integer		Fraction	Coefficient
.315 x 2 =	0	+	.630	a <sub>1</sub> = 0
.630 x 2 =	1	+	.26	a <sub>2</sub> = 1
.26 x 2 =	0	+	.52	a <sub>3</sub> = 0
.52 x 2 =	1	+	.04	a <sub>4</sub> = 1

$$.315_{10} \approx .0101_2 = .25 + .0625 = .3125$$

$$27.315 \approx 11011.0101_2$$

**(b)**  $2/3 \approx .6666666667$

	Integer		Fraction		Coefficient
.6666_6666_67 x 2 =	1	+	.3333_3333_34		a <sub>1</sub> = 1
.3333333334 x 2 =	0	+	.6666666668		a <sub>2</sub> = 0
.6666666668 x 2 =	1	+	.3333333336		a <sub>3</sub> = 1
.3333333336 x 2 =	0	+	.6666666672		a <sub>4</sub> = 0
.6666666672 x 2 =	1	+	.3333333344		a <sub>5</sub> = 1
.3333333344 x 2 =	0	+	.6666666688		a <sub>6</sub> = 0
.6666666688 x 2 =	1	+	.3333333376		a <sub>7</sub> = 1
.3333333376 x 2 =	0	+	.6666666752		a <sub>8</sub> = 0

$$.6666666667_{10} \approx .10101010_2 = .5 + .125 + .0313 + .0078 = .6641_{10}$$

$$.10101010_2 = .1010_2 = .AA_{16} = 10/16 + 10/256 = .6641_{10} \text{ (Same as (b)).}$$

<b>1.14</b>	<b>(a)</b>	1001_0000	<b>(b)</b>	0000_0000	<b>(c)</b>	1101_1010
	1s comp:	0110_1111	1s comp:	1111_1111	1s comp:	0010_0101
	2s comp:	0111_0001	2s comp:	0000_0000	2s comp:	0010_0110
	<b>(d)</b>	1010_1011	<b>(e)</b>	1010_0101	<b>(f)</b>	1111_1111
	1s comp:	0101_0100	1s comp:	0101_1010	1s comp:	0000_0000
	2s comp:	0101_0111	2s comp:	0101_1011	2s comp:	0000_0001

<b>1.15</b>	<b>(a)</b>	25,875,036	<b>(b)</b>	76,325,800
	9s comp:	74,124,963	9s comp:	26,674,199
	10s comp:	74,124,964	10s comp:	26,674,200

<b>(c)</b>	25,101,236	<b>(d)</b>	00000000
9s comp:	74,898,763	9s comp:	99999999
10s comp:	74,898,764	10s comp:	100000000

<b>1.16</b>	C3AF	C3AF:	1100_0011_1010_1111
15s comp:	3C50	1s comp:	0011_1100_0101_0000
16s comp:	3C51	2s comp:	0011_1100_0101_0001 = 3C51

**1.17 (a)**  $6,473 - 5297 = 1176$

$5297 \rightarrow 05297 \rightarrow 94702$  (9s comp)  $\rightarrow 94703$  (10s comp)

$6473 - 5297 = 6473 + 94703 = 101,176$  (positive)

Magnitude: 1,176

Result:  $6,473 - 5297 = 1176$

**(b)**  $1,076 - 3,217 = -2,141$

$3,217 \rightarrow 96,782$  (9s comp)  $\rightarrow 96,783$  (10s comp)

$1,076 - 3,217 = 1,076 + 96,783 = 97,858$  (negative)

Magnitude: 2,141

Result:  $1,076 - 3,217 = -2,141$

**(c)**  $4,361 \rightarrow 04361 \rightarrow 95638$  (9s comp)  $\rightarrow 95639$  (10s comp)

$2043 - 4361 = 02043 + 95639 = 97682$  (Negative)

Magnitude: 2318

Result:  $2043 - 6152 = -2318$

**(d)**  $745 \rightarrow 00745 \rightarrow 99254$  (9s comp)  $\rightarrow 99255$  (10s comp)

$1631 - 745 = 01631 + 99255 = 0886$  (Positive)

Result:  $1631 - 745 = 886$

**1.18 (a)**  $0\_10110(22)$

1s comp:  $1\_01001$

2s comp:  $1\_01010$

$0\_10111(23)$

Diff:  $0\_00001$  (Positive)

result is negative

Result: +1

Check:  $23 - 22 = +1$

**(b)**  $0\_100110$

1s comp:  $1\_011001$  with sign extension

2s comp:  $1\_011010$

$0\_100010$

$1\_111100$  sign bit indicates that the

$0\_000011$  1s complement

$0\_000100$  2s complement

$0\_000100$  magnitude

Result: -4

Check:  $34 - 38 = -4$

**(c)**  $0\_110101$

1s comp:  $1\_001010$

2s comp:  $1\_001011$

$0\_001001$

Diff:  $1\_010100$  (negative)

result is positive

$0\_101011$  (1s comp)

$0\_101100$  (2s complement)

$101100$  (magnitude)

$-44_{10}$  (result)

**(d)**  $0\_010101$

1s comp:  $1\_101010$  with sign extension

2s comp:  $1\_101011$

$0\_101000$

$0\_010011$  sign bit indicates that the

Result:  $19_{10}$

Check:  $40 - 21 = 19_{10}$

**1.19** +9286 → 009286; +801 → 000801; -9286 → 990714; -801 → 999199

(a)  $(+9286) + (+801) = 009286 + 000801 = 010087$

(b)  $(+9286) + (-801) = 009286 + 999199 = 008485$

(c)  $(-9286) + (+801) = 990714 + 000801 = 991515$

(d)  $(-9286) + (-801) = 990714 + 999199 = 989913$

**1.20** +49 → 0\_110001 (Needs leading zero extension to indicate + value);

+29 → 0\_011101 (Leading 0 indicates + value)

-49 → 1\_001110 + 0\_000001 → 1\_001111

-29 → 1\_100011 (sign extension indicates negative value)

(a)  $(+29) + (-49) = 0_011101 + 1_001111 = 1_101100$  (1 indicates negative value.)  
Magnitude =  $0_010011 + 0_000001 = 0_010100 = 20$ ; Result  $(+29) + (-49) = -20$

(b)  $(-29) + (+49) = 1_100011 + 0_110001 = 0_010100$  (0 indicates positive value)  
 $(-29) + (+49) = +20$

(c) Must increase word size by 1 (sign extension) to accomodate overflow of values:  
 $(-29) + (-49) = 11_100011 + 11_001111 = 10_110010$  (1 indicates negative result)  
Magnitude:  $01_001110 = 78_{10}$   
Result:  $(-29) + (-49) = -78_{10}$

**1.21** +9742 → 009742 → 990257 (9's comp) → 990258 (10s comp)

+641 → 000641 → 999358 (9's comp) → 999359 (10s comp)

(a)  $(+9742) + (+641) → 010383$

(b)  $(+9742) + (-641) → 009742 + 999359 = 009101$   
Result:  $(+9742) + (-641) = 9101$

(c)  $(-9742) + (+641) = 990258 + 000641 = 990899$  (negative)  
Magnitude: 009101  
Result:  $(-9742) + (641) = -9101$

(d)  $(-9742) + (-641) = 990258 + 999359 = 989617$  (Negative)  
Magnitude: 10383  
Result:  $(-9742) + (-641) = -10383$

**1.22** 6,514

BCD: 0110\_0101\_0001\_0100

ASCII: 0\_011\_0110\_0\_011\_0101\_1\_011\_0001\_1\_011\_0100  
 ASCII: 0011\_0110\_0011\_0101\_1011\_0001\_1011\_0100

3,274

BCD: 0011\_0010\_0111\_0100

ASCII: 0011\_0011\_1011\_0010\_1011\_0111\_1011\_0100

### 1.23

0111 1001 0001 ( 791)  
 0110 0101 1000 (+658)  
 1101 1110 1001  
 0110 0110  
 0001 0011 0100  
 0001 0001  
 0001 0100 0100 1001 (1,449)

### 1.24 (a) See text

(b) 6 4 2 1 **Decimal**

0 0 0 0 0  
 0 0 0 1 1  
 0 0 1 0 2  
 0 0 1 1 3  
 0 1 0 0 4  
 0 1 0 1 5  
 0 1 1 0 6  
 0 1 1 1 7  
 1 0 1 0 8  
 1 0 1 1 9

### 1.25

(a) 6,428<sub>10</sub> BCD: 0110\_0100\_0010\_1000

(b) Excess-3: 1001\_0111\_0101\_1011

(c) 2421: 1100\_0100\_0010\_1110

2421: 0110\_0100\_1000\_1110

(d) 6311: 1000\_0110\_0010\_1011

### 1.26

6,428<sub>9s</sub> Comp: 3,571

6 4 2 8

2421 code: 0011\_1011\_0111\_0001

1.25(c): 1100\_0100\_0010\_1110 (2421 code – alternative #1)

1s comp: 0011\_1011\_1101\_0001 (2421 code - alternative #2)

6 4 2 8

6,428<sub>2421</sub>    0110\_0100\_1000\_1110(2421 code alternative #2)  
1s comp        1001\_1011\_0111\_0001 Match

5,736 9s Comp:    4,263  
2421 code: 0100\_0010\_1100\_0011  
1s comp:        1011\_1101\_0011\_1100

**1.27** For a deck with 52 cards, we need 6 bits ( $2^5 = 32 < 52 < 64 = 2^6$ ). Let the msb's select the suit (e.g., diamonds, hearts, clubs, spades are encoded respectively as 00, 01, 10, and 11. The remaining four bits select the "number" of the card. Example: 0001 (ace) through 1011 (9), plus 101 through 1100 (jack, queen, king). This a jack of spades might be coded as 11\_1010. (Note: only 52 out of 64 patterns are used.)

**1.28**        G        (dot)        (space)        B        o        o        l        e  
  
11000111\_11101111\_01101000\_01101110\_00100000\_11000100\_11101111\_11100101

**1.29** Steve Jobs

**1.30** 73 F4 E5 76 E5 4A EF 62 73

73:    0\_111\_0011    s  
F4:    1\_111\_0100    t  
E5:    1\_110\_0101    e  
76:    0\_111\_0110    v  
E5:    1\_110\_0101    e  
4A: 0\_100\_1010    j  
EF: 1\_110\_1111    o  
62:    0\_110\_0010    b  
73:    0\_111\_0011    s

Even parity

**1.31** 62 + 32 = 94 printing characters; 34 special characters

**1.32** Complement bit 6 (from the right)

**1.33** (a) 897        (b) 564        (c) 871        (d) 2,199

**1.34** ASCII for decimal digits with even parity:

(0): 00110000 (1): 10110001 (2): 10110010 (3): 00110011  
(4): 10110100 (5): 00110101 (6): 00110110 (7): 10110111  
(8): 10111000 (9): 00111001

## CHAPTER 2

### 2.1 (a)

$x y z$	$x + y + z$	$(x + y + z)'$	$x'$	$y'$	$z'$	$x' y' z'$	$x y z$	$(xyz)$	$(xyz)'$	$x'$	$y'$	$z'$	$x' + y' + z'$
0 0 0	0	1	1	1	1	1	0 0 0	0	1	1	1	1	1
0 0 1	1	0	1	1	0	0	0 0 1	0	1	1	1	0	1
0 1 0	1	0	1	0	1	0	0 1 0	0	1	1	0	1	1
0 1 1	1	0	1	0	0	0	0 1 1	0	1	1	0	0	1
1 0 0	1	0	0	1	1	0	1 0 0	0	1	0	1	1	1
1 0 1	1	0	0	1	0	0	1 0 1	0	1	0	1	0	1
1 1 0	1	0	0	0	1	0	1 1 0	0	1	0	0	1	1
1 1 1	1	0	0	0	0	0	1 1 1	1	0	0	0	0	0

### (b)

$x y z$	$x + y z$	$(x + y)$	$(x + z)$	$(x + y)(x + z)$
0 0 0	0	0	0	0
0 0 1	0	0	1	0
0 1 0	0	1	0	0
0 1 1	1	1	1	1
1 0 0	1	1	1	1
1 0 1	1	1	1	1
1 1 0	1	1	1	1
1 1 1	1	1	1	1

### (c)

$x y z$	$x(y + z)$	$xy$	$xz$	$xy + xz$
0 0 0	0	0	0	0
0 0 1	0	0	0	0
0 1 0	0	0	0	0
0 1 1	0	0	0	0
1 0 0	0	0	0	0
1 0 1	1	0	1	1
1 1 0	1	1	0	1
1 1 1	1	1	1	1

### (c)

$x y z$	$x$	$y + z$	$x + (y + z)$	$(x + y)$	$(x + y) + z$
0 0 0	0	0	0	0	0
0 0 1	0	1	1	0	1
0 1 0	0	1	1	1	1
0 1 1	0	1	1	1	1
1 0 0	1	0	1	1	1
1 0 1	1	1	1	1	1
1 1 0	1	1	1	1	1
1 1 1	1	1	1	1	1

### (d)

$x y z$	$yz$	$x(yz)$	$xy$	$(xy)z$
0 0 0	0	0	0	0
0 0 1	0	0	0	0
0 1 0	0	0	0	0
0 1 1	1	0	0	0
1 0 0	0	0	0	0
1 0 1	0	0	0	0
1 1 0	0	0	1	0
1 1 1	1	1	1	1

**2.2 (a)**  $xy + xy' = x(y + y') = x$

**(b)**  $(x + y)(x + y') = x + yy' = x(x + y') + y(x + y') = xx + xy' + xy + yy' = x$

**(c)**  $xyz + x'y + xyz' = xy(z + z') + x'y = xy + x'y = y$

**(d)**  $(x + y)'(x' + y')' = (x'y')(xy) = (x'y')(yx) = x'(y'y)x = 0$

**(e)**  $(x + y + z')(x'y' + z) = xx'y' + xz + ya'y' + yz + z'x'y' + z'z = xz + yz + x'y'z'$

$$(f) x'yz + xyz' + xyz + x'yz' = x'y(z + z') + xy(z + z') = x'y + xy = (x' + x)y = y$$

**2.3 (a)**  $xyz + x'y + xyz' = xy + x'y = y$

**(b)**  $x'yz + xz = (x'y + x)z = z(x + x')(x + y) = z(x + y)$

**(c)**  $(x + y)'(x' + y') = x'y'(x' + y') = x'y'$

**(d)**  $xy + x(wz + wz') = x(y + wz + wz') = x(w + y)$

**(e)**  $(yz' + x'w)(xy' + zw') = yz'xy' + yz'zw' + x'wxy' + x'wzw' = 0$

**(f)**  $(x' + z')(x + y' + z') = x'x + x'y' + x'z' + z'x + z'y' + z'z' = x'y' + x'z' + xz' + y'z' = z' + y'(x' + z')$   
 $= z' + y'z' + x'y' = z' + x'y'$

**2.4 (a)**  $A'C' + ABC + AC' = C' + ABC = (C + C')(C' + AB) = AB + C'$

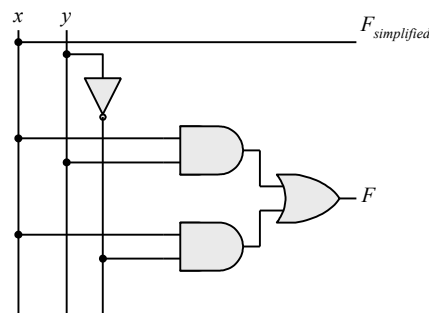
**(b)**  $(B'C' + D)' + D + BC + AD = (B'C')'D' + D + BC + AD = [(B + C)D' + D] + BC + AD =$   
 $= (D + D')(D + B + C) + BC + AD = D + AD + B + BC + C = D(1 + A) + B(1 + C) + C$   
 $= B + C + D$

**(c)**  $A'B(D' + C'D) + B(A + A'CD) = B(A'D' + A'C'D + A + A'CD)$   
 $= B(A'D' + A + A'D(C + C')) = B(A + A'(D' + D)) = B(A + A') = B$

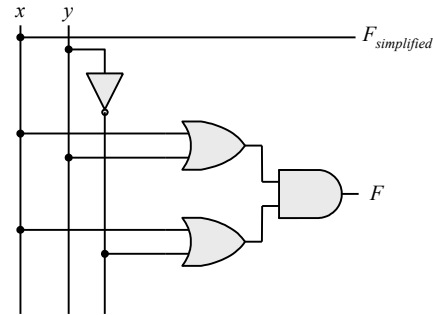
**(d)**  $(A' + C)(A' + C')(A + B + C'D) = (A' + CC')(A + B + C'D) = A'(A + B + C'D)$   
 $= AA' + A'B + A'C'D = A'(B + C'D)$

**(e)**  $ABC'D + A'BD + ABCD = AB(C + C')D + A'BD = ABD + A'BD = BD$

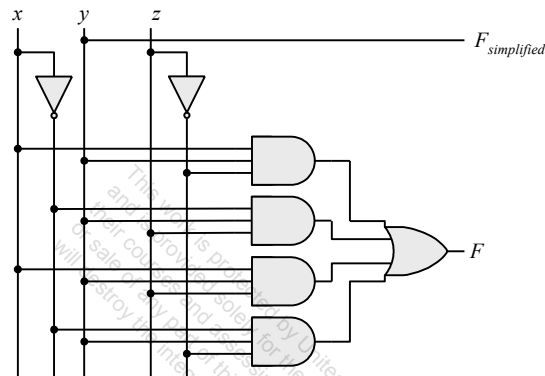
**2.5 (a)**



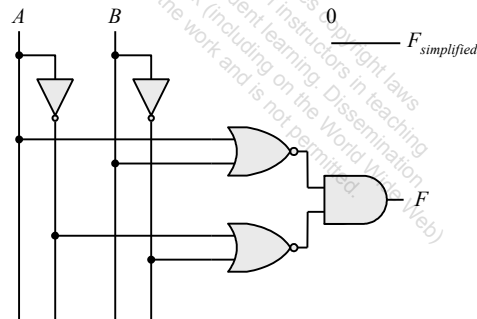
**(b)**



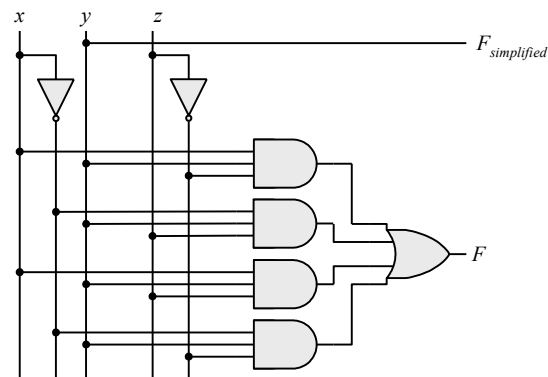
(c)



(d)

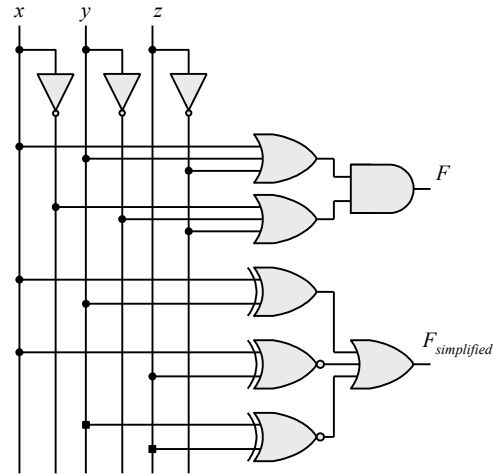


(e)

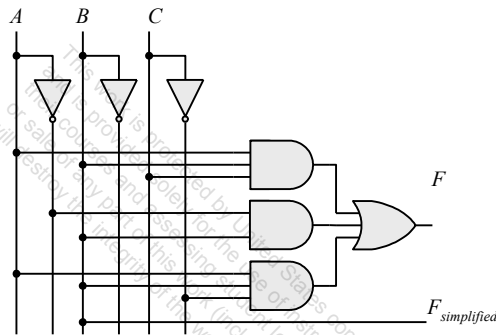


(f)

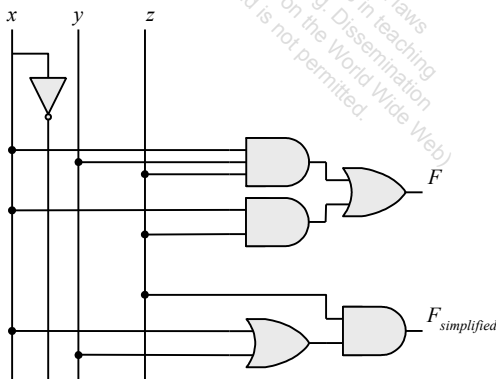




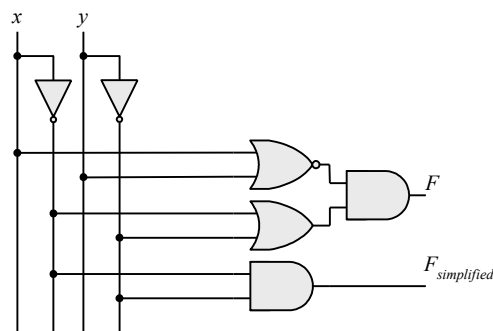
2.6 (a)



(b)

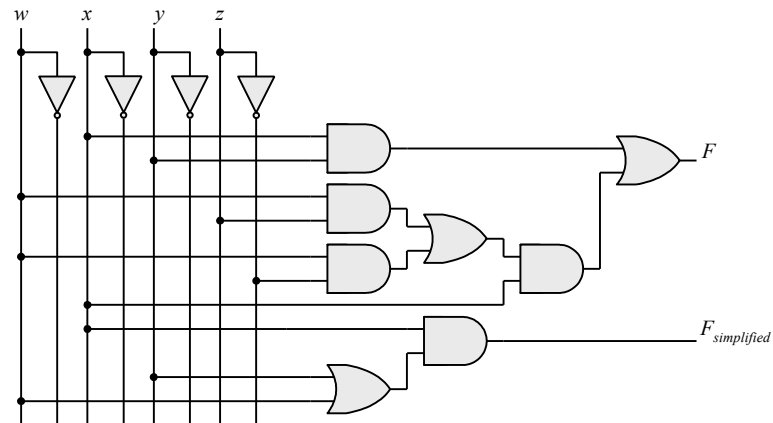


(c)

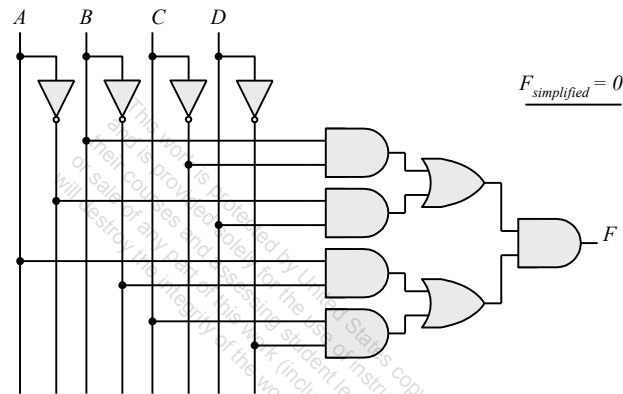


This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

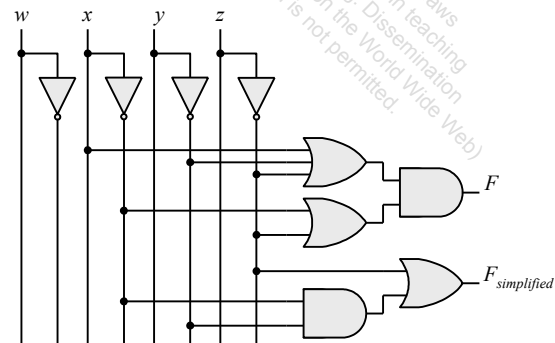
(d)



(e)

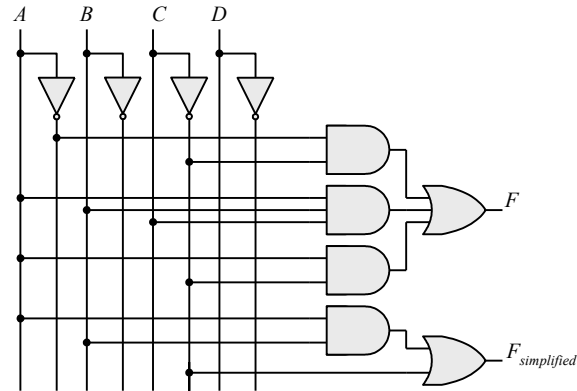


(f)

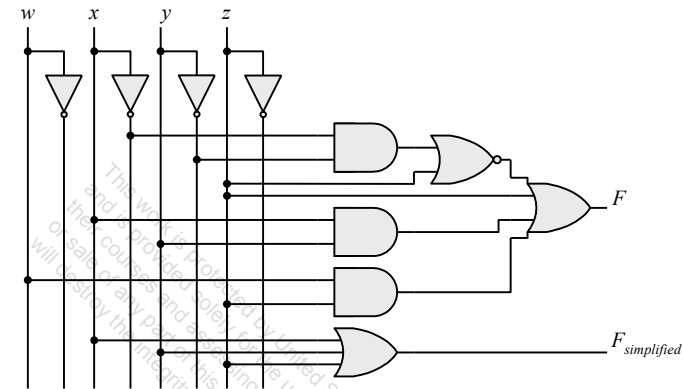


2.7

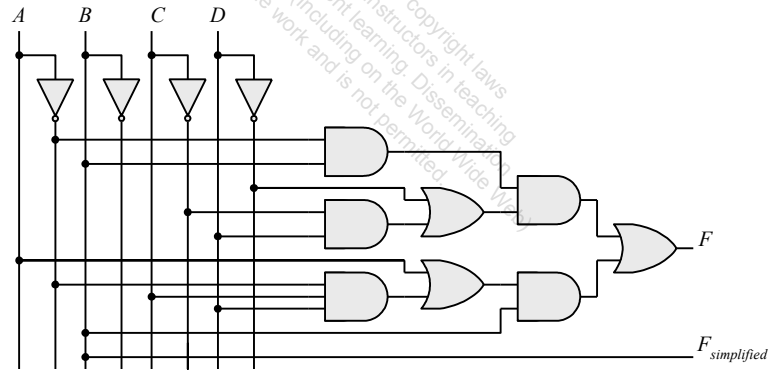
(a)



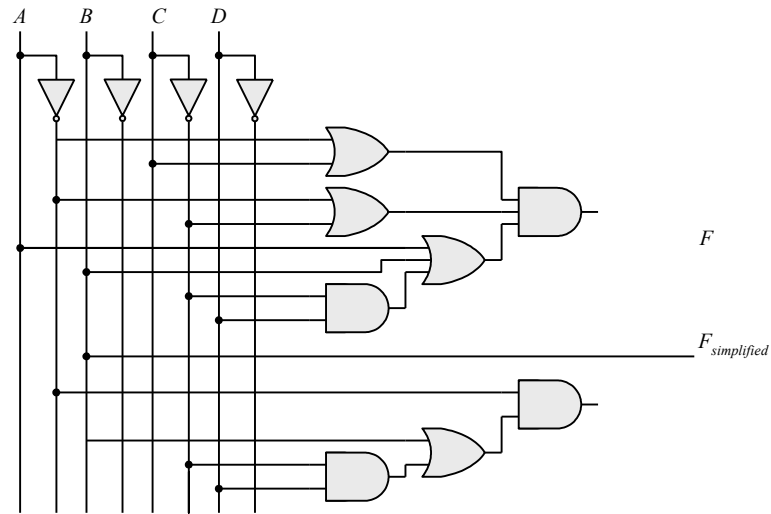
(b)



(c)

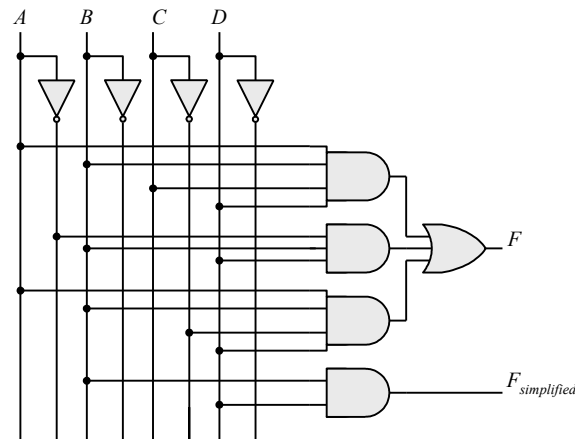


(d)



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

(e)



**2.8**  $F' = (wx + yz)' = (wx)'(yz)' = (w' + x')(y' + z')$

$$FF' = wx(w' + x')(y' + z') + yz(w' + x')(y' + z') = 0$$

$$F + F' = wx + yz + (wx + yz)' = A + A' = 1 \text{ with } A = wx + yz$$

**2.9 (a)**  $F' = (xy' + x'y)' = (xy')'(x'y)' = (x' + y)(x + y') = xy + x'y'$

**(b)**  $F' = [(a + c)(a + b')(a' + b + c')]' = (a + c)' + (a + b')' + (a' + b + c')'$   
 $= a'c' + a'b + ab'c$

**(c)**  $F' = [z + z'(v'w + xy)]' = z'[z'(v'w + xy)]' = z'[z'v'w + xyz']'$   
 $= z'[(z'v'w)'(xyz')'] = z'[(z + v + w)' + (x' + y' + z)]$   
 $= z'z + z'v + z'w' + z'x' + z'y' + z'z = z'(v + w' + x' + y')$

**2.10 (a)**  $F_1 + F_2 = \Sigma m_{1i} + \Sigma m_{2i} = \Sigma (m_{1i} + m_{2i})$

**(b)**  $F_1 F_2 = \Sigma m_i \Sigma m_j$  where  $m_i m_j = 0$  if  $i \neq j$  and  $m_i m_j = 1$  if  $i = j$

**2.11 (a)**  $F(x, y, z) = \Sigma(1, 4, 5, 6, 7)$

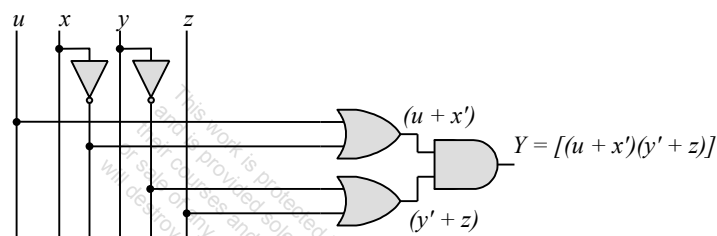
**(b)**  $F(a, b, c) = \Sigma(0, 2, 3, 4, 5, 7)$

$F = xy + xy' + y'z$		$F = ac + b'c'$	
		$= \Sigma(0, 2, 3, 4, 5, 7)$	
x y z	F	a b c	F
0 0 0	0	0 0 0	1
0 0 1	1	0 0 1	0
0 1 0	0	0 1 0	1
0 1 1	0	0 1 1	1
1 0 0	1	1 0 0	1
1 0 1	1	1 0 1	1
1 1 0	1	1 1 0	0
1 1 1	1	1 1 1	1

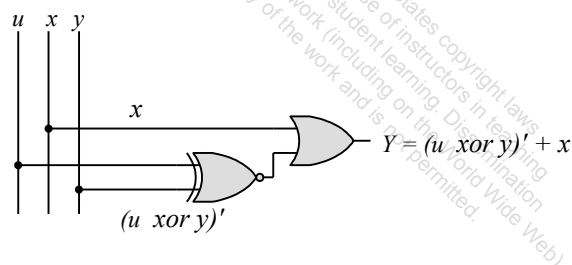
**2.12**       $A = 1011\_0001$   
                $B = 1010\_1100$

- (a)  $A \text{ AND } B = 1010\_0000$   
 (b)  $A \text{ OR } B = 1011\_1101$   
 (c)  $A \text{ XOR } B = 0001\_1101$   
 (d)  $\text{NOT } A = 0100\_1110$   
 (e)  $\text{NOT } B = 0101\_0011$

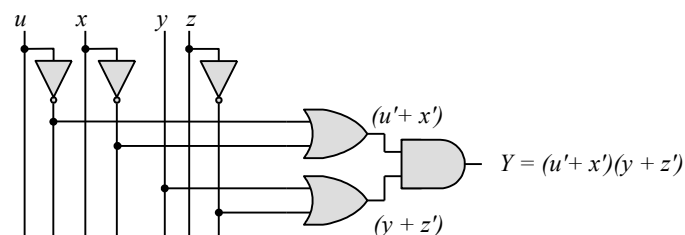
**2.13**    (a)



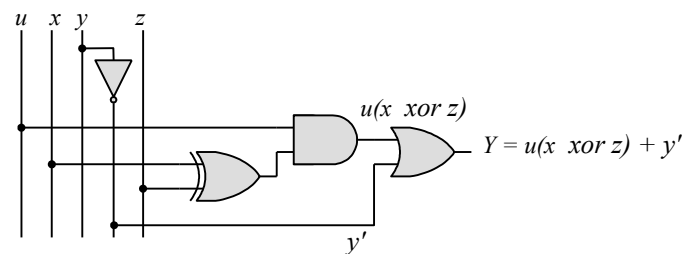
(b)



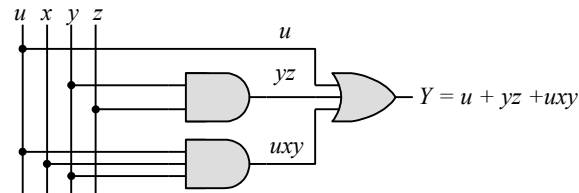
(c)



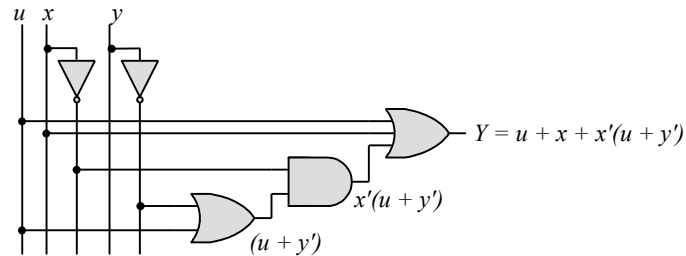
(d)



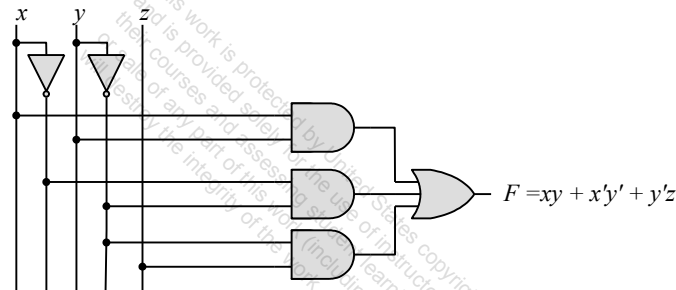
(e)



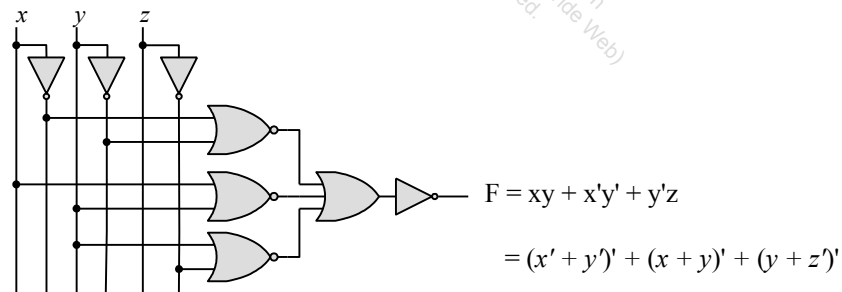
(f)



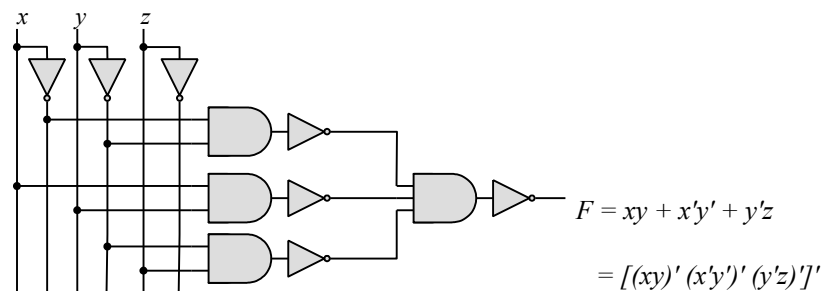
2.14 (a)



(b)

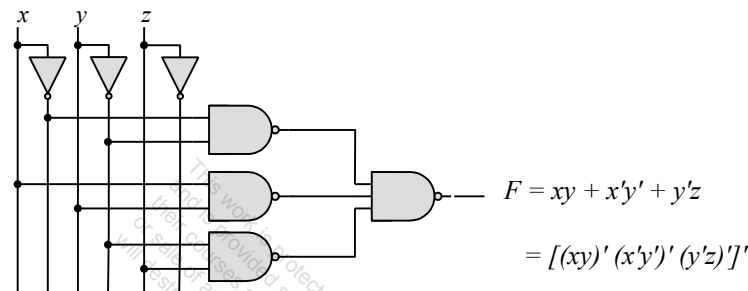


(c)

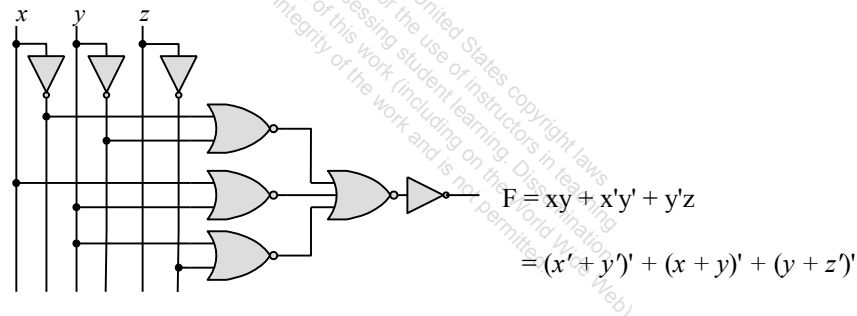




(d)



(e)



**2.15 (a)**  $T_1 = A'B'C' + A'B'C + A'BC' = A'B'(C' + C) + A'C'(B' + B) = A'B' + A'C' = A'(B' + C')$

**(b)**  $T_2 = T_1' = A'BC + AB'C' + AB'C + ABC' + ABC$   
 $= BC(A' + A) + AB'(C' + C) + AB(C' + C)$   
 $= BC + AB' + AB = BC + A(B' + B) = A + BC$

$\Sigma(3, 5, 6, 7) = \Pi(0, 1, 2, 4)$

$T_1 = A'B'C' + A'B'C + A'BC'$   
  
 $T_1 = A'B' + A'C' = A'(B' + C')$

$T_2 = A'BC + AB'C' + AB'C + ABC' + ABC$   
  
 $T_2 = AC' + BC + AC = A + BC$

$$\begin{aligned}
 \mathbf{2.16} \quad (\mathbf{a}) \quad F(A, B, C) &= A'B'C' + A'B'C + A'BC' + A'BC + AB'C' + AB'C + ABC' + ABC \\
 &= A'(B'C' + B'C + BC' + BC) + A((B'C' + B'C + BC' + BC)) \\
 &= (A' + A)(B'C' + B'C + BC' + BC) = B'C' + B'C + BC' + BC \\
 &= B'(C' + C) + B(C' + C) = B' + B = 1
 \end{aligned}$$

(b)  $F(x_1, x_2, x_3, \dots, x_n) = \sum m_i$  has  $2^{n-1}/2$  minterms with  $x_1$  and  $2^{n-1}/2$  minterms with  $x'_1$ , which can be factored and removed as in (a). The remaining  $2^{n-1}$  product terms will have  $2^{n-1}/2$  minterms with  $x_2$  and  $2^{n-1}/2$  minterms with  $x'_2$ , which can be factored to remove  $x_2$  and  $x'_2$ . continue this process until the last term is left and  $x_n + x'_n = 1$ . Alternatively, by induction,  $F$  can be written as  $F = x_n G + x'_n G$  with  $G = 1$ . So  $F = (x_n + x'_n)G = 1$ .

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

**2.17 (a)**  $F = (b + cd)(c + bd) = bc + bd + cd + bcd$   
 $= \Sigma(3, 5, 6, 7, 11, 13, 14, 15)$   
 $F' = \Sigma(0, 1, 2, 4, 8, 9, 10, 12)$   
 $F = \Pi(0, 1, 2, 4, 8, 9, 10, 12)$

a	b	c	d	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

**(b)**  $(cd + b'c + bd')(b + d) = bcd + bd' + cd + b'cd = cd + bd'$   
 $= \Sigma(3, 4, 7, 11, 12, 14, 15)$   
 $= \Pi(0, 1, 2, 5, 6, 8, 9, 10, 13)$

a	b	c	d	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

**(c)**  $(c' + d)(b + c') = bc' + c' + bd + c'd = (c' + bd)$   
 $= \Sigma(0, 1, 4, 5, 7, 8, 12, 13, 15)$   
 $= \Pi(2, 3, 6, 9, 10, 11, 14)$

(d)  $bd' + acd' + ab'c + a'c' = \Sigma (0, 1, 4, 5, 10, 11, 14)$

$F' = \Sigma (2, 3, 6, 7, 8, 9, 12, 13, 15)$

$F = \Pi (0, 2, 3, 6, 7, 8, 12, 13, 15)$

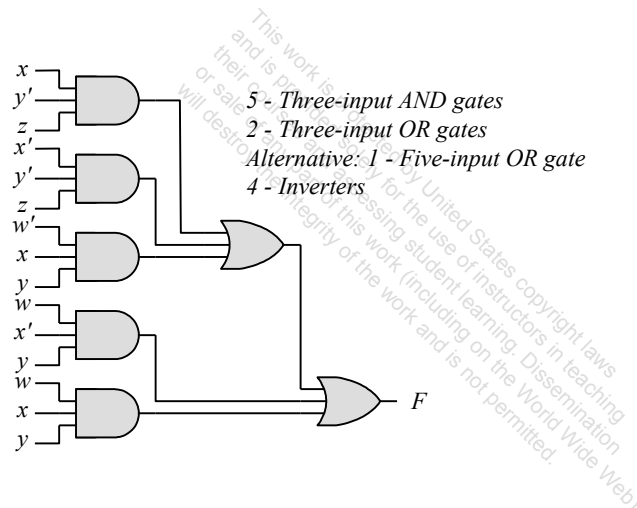
a	b	c	d	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

2.18 (a)

wx y z	F	$F = xy'z + x'y'z + w'xy + wx'y + wxy$ $F = \Sigma(1, 5, 6, 7, 9, 10, 11, 13, 14, 15)$
00 0 0	0	
00 0 1	1	
00 1 0	0	
00 1 1	0	
01 0 0	0	
01 0 1	1	
01 1 0	1	
01 1 1	1	
10 0 0	0	
10 0 1	1	
10 1 0	1	
10 1 1	1	
11 0 0	0	
11 0 1	1	
11 1 0	1	
11 1 1	1	

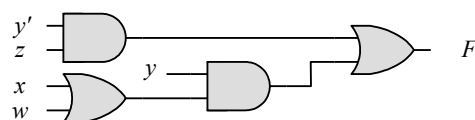
(b)



(c)  $F = xy'z + x'y'z + w'xy + wx'y + wxy = y'z + xy + wy = y'z + y(w + x)$

(d)  $F = y'z + yw + yx = \Sigma(1, 5, 9, 13, 10, 11, 13, 15, 6, 7, 14, 15)$   
 $= \Sigma(1, 5, 6, 7, 9, 10, 11, 13, 14, 15)$

(e)



1 - Inverter, 2 - Two-input AND gates, 2 - Two-input OR gates

**2.19**  $F = B'D + A'D + BD$

$ABCD$	$ABCD$	$ABCD$
$\overline{B'D}$	$A'\overline{D}$	$\overline{B-D}$
0001 = 1	0001 = 1	0101 = 5
0011 = 3	0011 = 3	0111 = 7
1001 = 9	0101 = 5	1101 = 13
1011 = 11	0111 = 7	1111 = 15

$$F = \Sigma(1, 3, 5, 7, 9, 11, 13, 15) = \Pi(0, 2, 4, 6, 8, 10, 12, 14)$$

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

(b)  $F(x, y, z) = \Pi(3, 5, 7)$   
 $F' = \Pi(0, 1, 2, 4, 6)$   
 $F = \Sigma(0, 1, 2, 4, 6)$

**2.21** (a)  $F(x, y, z) = \Sigma(1, 3, 5) = \Pi(0, 2, 4, 6, 7)$

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

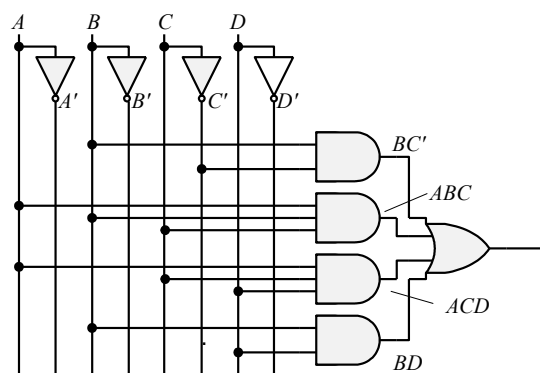
(c)  $F(x, y, z) = \Pi(0, 2, 4, 6) = \Sigma(1, 3, 5, 7)$

(d)  $F(w, x, y, z) = \Sigma(1, 3, 5, 7, 9) = \Pi(0, 2, 4, 6, 8, 10, 11, 12, 13, 14, 15)$

**2.22** (a)  $(u + xw)(x + u'v) = ux + uu'v + xxw + xwu'v = ux + xw + xwu'v$   
 $= ux + xw = x(u + w)$   
 $= ux + xw$  (SOP form)  
 $= x(u + w)$  (POS form)

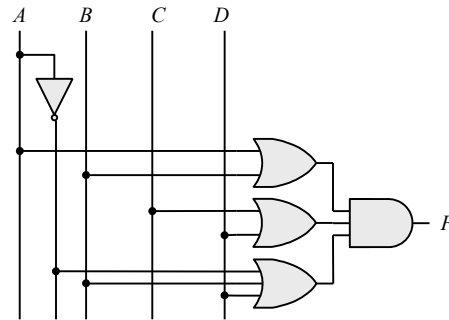
(b)  $x' + x(x + y')(y + z') = x' + x(xy + xz' + y'y + y'z')$   
 $= x' + xy + xz' + xy'z' = x' + xy + xz'$  (SOP form)  
 $= (x' + y + z')$  (POS form)

**2.23** (a)  $BC' + ABC + ACD + BD$

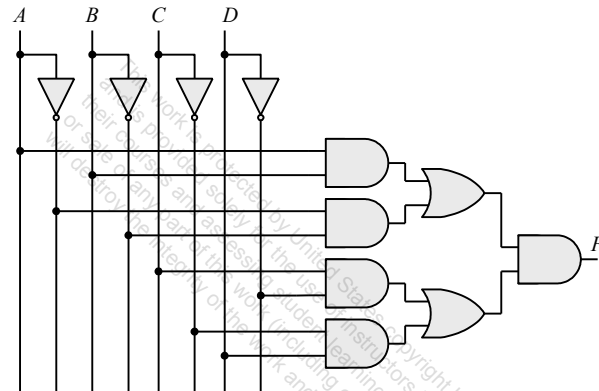


This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

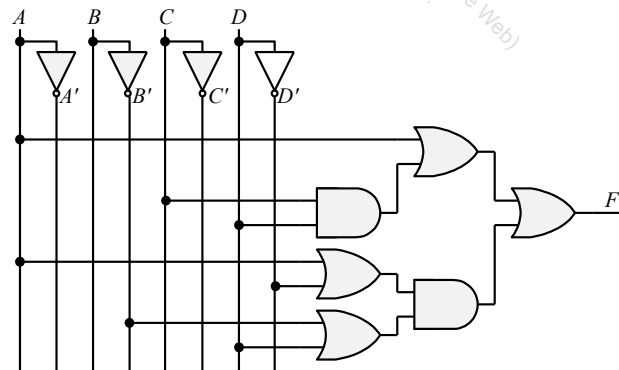
(b)  $(A + B)(C + D)(A' + B + D)$



(c)  $(AB + A'B')(CD' + C'D)$



(d)  $A + CD + (A + D')(B' + D)$



2.24  $x \oplus y = x'y + xy'$  and  $(x \oplus y)' = (x + y')(x' + y)$

Dual of  $x'y + xy' = (x' + y)(x + y') = (x \oplus y)'$



**2.25** (a)  $x|y = xy' \neq y|x = x'y$  Not commutative  
 $(x|y)|z = xy'z' \neq x|(y|z) = x(yz')' = xy' + xz$  Not associative

(b)  $(x \oplus y) = xy' + x'y = y \oplus x = yx' + y'x$  Commutative

$(x \oplus y) \oplus z = \sum(1, 2, 4, 7) = x \oplus (y \oplus z)$  Associative

## 2.26

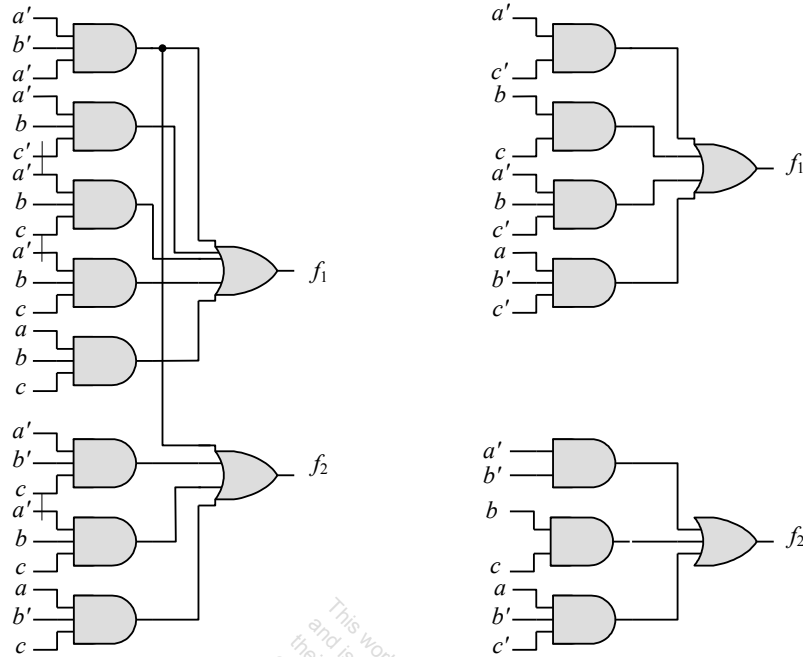
Gate		NAND (Positive logic)		NOR (Negative logic)	
x y	z	x y	z	x y	z
L L	H	0 0	1	1 1	0
L H	H	0 1	1	1 0	0
H L	H	1 0	1	0 1	0
H H	L	1 1	0	0 0	1

Gate		NOR (Positive logic)		NAND (Negative logic)	
x y	z	x y	z	x y	z
L L	H	0 0	1	1 1	0
L H	L	0 1	0	1 0	1
H L	L	1 0	0	0 1	1
H H	L	1 1	0	0 0	1

**2.27**  $f_1 = a'b'c' + a'bc' + a'bc + ab'c' + abc = a'c' + bc + a'bc' + ab'c'$

$f_2 = a'b'c' + a'b'c + a'bc + ab'c' + abc = a'b' + bc + ab'c'$



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

**2.28** (a)  $y = a(bcd)'e = a(b' + c' + d')e$

$$y = a(b' + c' + d')e = ab'e + ac'e + ad'e$$

$$= \Sigma(17, 19, 21, 23, 25, 27, 29)$$

a bcde	y	a bcde	y
0 0000	0	1 0000	0
0 0001	0	<b>1 0001</b>	1
0 0010	0	1 0010	0
0 0011	0	<b>1 0011</b>	1
0 0100	0	1 0100	0
0 0101	0	<b>1 0101</b>	1
0 0110	0	1 0110	0
0 0111	0	<b>1 0111</b>	1
0 1000	0	1 1000	0
0 1001	0	<b>1 1001</b>	1
0 1010	0	1 1010	0
0 1011	0	<b>1 1011</b>	1
0 1100	0	1 1100	0
0 1101	0	<b>1 1101</b>	1
0 1110	0	1 1110	0
0 1111	0	<b>1 1111</b>	1

(b)  $y_1 = a \oplus (c + d + e) = a'(c + d + e) + a(c'd'e') = a'c + a'd + a'e + ac'd'e'$

$$y_2 = b'(c + d + e)f = b'cf + b'df + b'ef$$

$$y_1 = a(c + d + e) = a'(c + d + e) + a(c'd'e') = a'c + a'd + a'e + ac'd'e'$$

$$y_2 = b'(c + d + e)f = b'cf + b'df + b'ef$$

$a'-c---$	$a'--d--$	$a'---e-$	$a-c'd'e'-$		
001000 = 8	000100 = 8	000010 = 2	100000 = 32		
001001 = 9	000101 = 9	000011 = 3	100001 = 33		
001010 = 10	000110 = 10	000110 = 6	110000 = 34		
001011 = 11	000111 = 11	000111 = 7	110001 = 35		
001100 = 12	001100 = 12	001010 = 10			
001101 = 13	001101 = 13	001011 = 11			
001110 = 14	001110 = 14	001110 = 14			
001111 = 15	001111 = 15	001111 = 15			
011000 = 24	010100 = 20	010010 = 18	001001 = 9	001001 = 9	000011 = 3
011001 = 25	010101 = 21	010011 = 19	001011 = 11	001011 = 11	000111 = 7
011010 = 26	010110 = 22	010110 = 22	001101 = 13	001101 = 13	001011 = 11
011011 = 27	010111 = 23	010111 = 23	001111 = 15	001111 = 15	001111 = 15
			101001 = 41	101001 = 41	100011 = 35
011100 = 28	011100 = 28	011010 = 26	101011 = 43	101011 = 43	100111 = 39
011101 = 29	011101 = 29	011001 = 27	101101 = 45	101101 = 45	101011 = 51
011110 = 30	011110 = 30	011110 = 30	101111 = 47	101111 = 47	101111 = 55
011111 = 31	011111 = 31	011111 = 31			

$$y_1 = \Sigma (2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 18, 19, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35)$$

$$y_2 = \Sigma (3, 7, 9, 13, 15, 35, 39, 41, 43, 45, 47, 51, 55)$$

<i>ab cdef</i>	<i>y<sub>1</sub> y<sub>2</sub></i>	<i>ab cdef</i>	<i>y<sub>1</sub> y<sub>2</sub></i>	<i>ab cdef</i>	<i>y<sub>1</sub> y<sub>2</sub></i>	<i>ab cdef</i>	<i>y<sub>1</sub> y<sub>2</sub></i>
00 0000	0 0	01 0000	0 0	10 0000	1 0	11 0000	0 0
00 0001	0 0	01 0001	0 0	10 0001	1 0	11 0001	0 0
00 0010	1 0	01 0010	1 0	10 0010	1 0	11 0010	0 0
00 0011	1 1	01 0011	1 0	10 0011	1 1	11 0011	0 1
00 0100	0 0	01 0100	0 0	10 0100	0 0	11 0100	0 0
00 0101	0 0	01 0101	0 0	10 0101	0 0	11 0101	0 0
00 0110	1 0	01 0110	1 0	10 0110	0 0	11 0110	0 0
00 0111	1 1	01 0111	1 0	10 0111	0 1	11 0111	0 1
00 1000	1 0	01 1000	1 0	10 1000	0 0	11 1000	0 0
00 1001	1 1	01 1001	1 0	10 1001	0 1	11 1001	0 0
00 1010	1 0	01 1010	1 0	10 1010	0 0	11 1010	0 0
00 1011	1 0	01 1011	1 0	10 1011	0 1	11 1011	0 0
00 1100	1 0	01 1100	1 0	10 1100	0 0	11 1100	0 0
00 1101	1 1	01 1101	1 0	10 1101	0 1	11 1101	0 0
00 1110	1 0	01 1110	1 0	10 1110	0 0	11 1110	0 0
00 1111	1 1	01 1111	1 0	10 1111	0 1	11 1111	0 0

**2.29**  $x'y' + x'z + x'z' = x'z' + y'z' + x'z$

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

$$x' = x' + y'z' \text{ FALSE}$$

**2.30**  $(b + d)(a' + b' + c) = a'b + bb' + bc + a'd + b'd + cd = a'b + bc + a'd + b'd + cd$

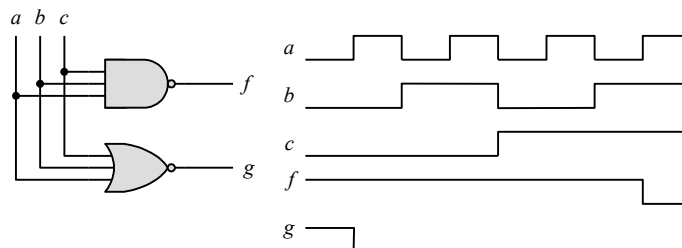
**2.31**  $a'b + a'c' + abc = a'bc + a'bc' + a'bc' + a'b'c' + abc = \Sigma (m_3 + m_2 + m_0 + m_7)$

$$(a'b + a'c' + abc)' = \Sigma (m_1 + m_4 + m_5 + m_6)$$

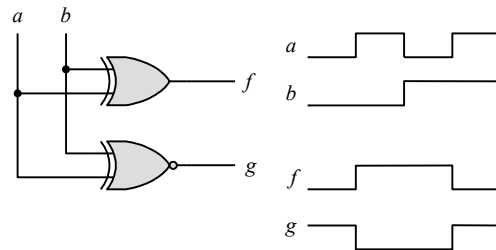
$$(a'b + a'c' + abc) = \Pi (M_1 + M_4 + M_5 + M_6)$$

$$(a'b + a'c' + abc) = (a' + b' + c)(a + b' + c')(a + b' + c)(a + b + c')$$

**2.32**



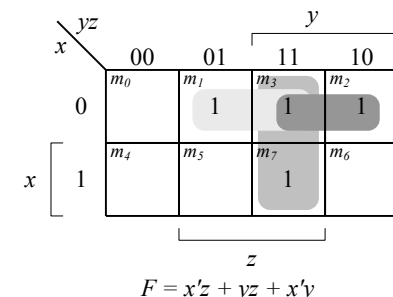
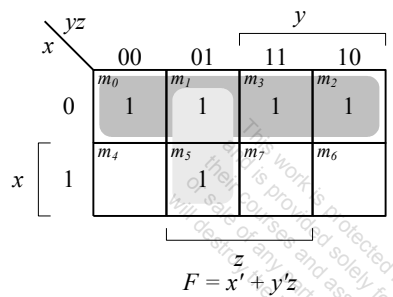
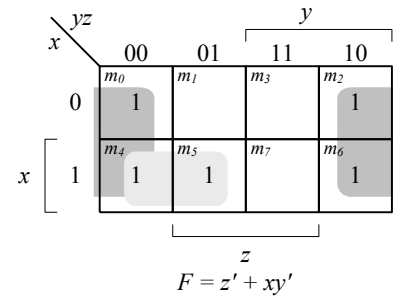
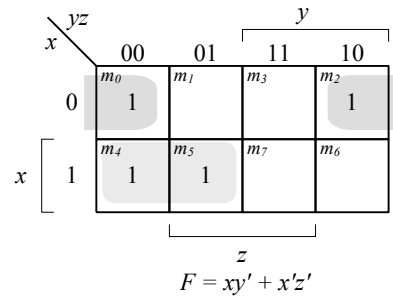
## 2.33



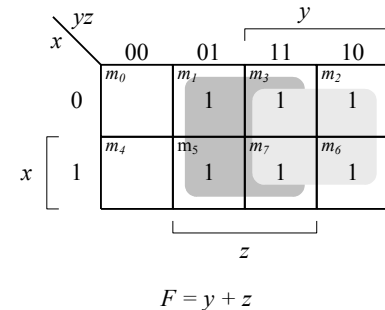
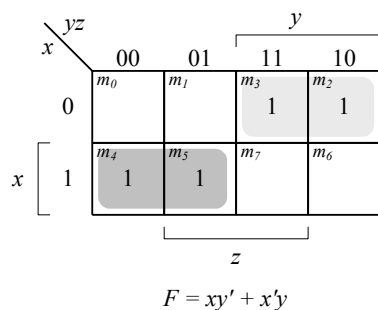
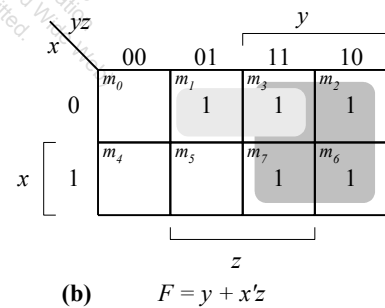
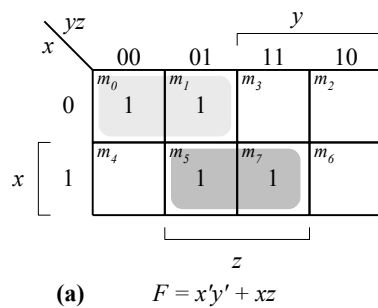
This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

## Chapter 3

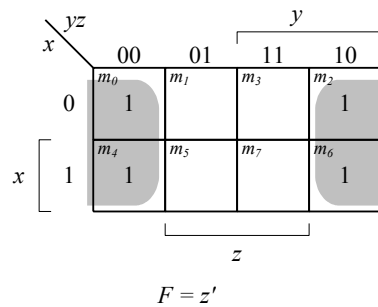
### 3.1



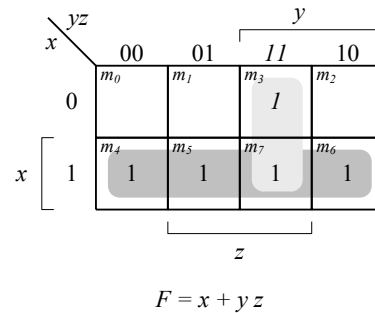
### 3.2



(c)



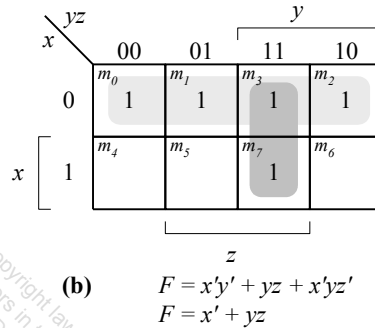
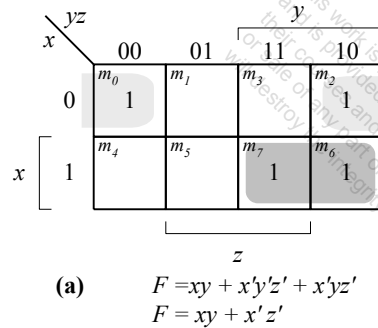
(d)



(e)

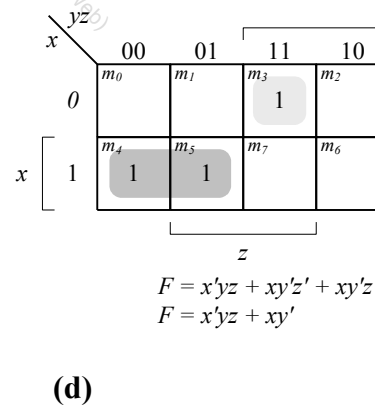
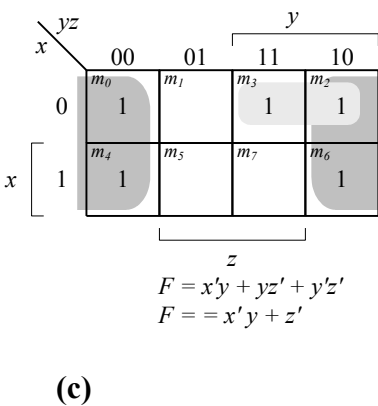
(f)

### 3.3



(a)

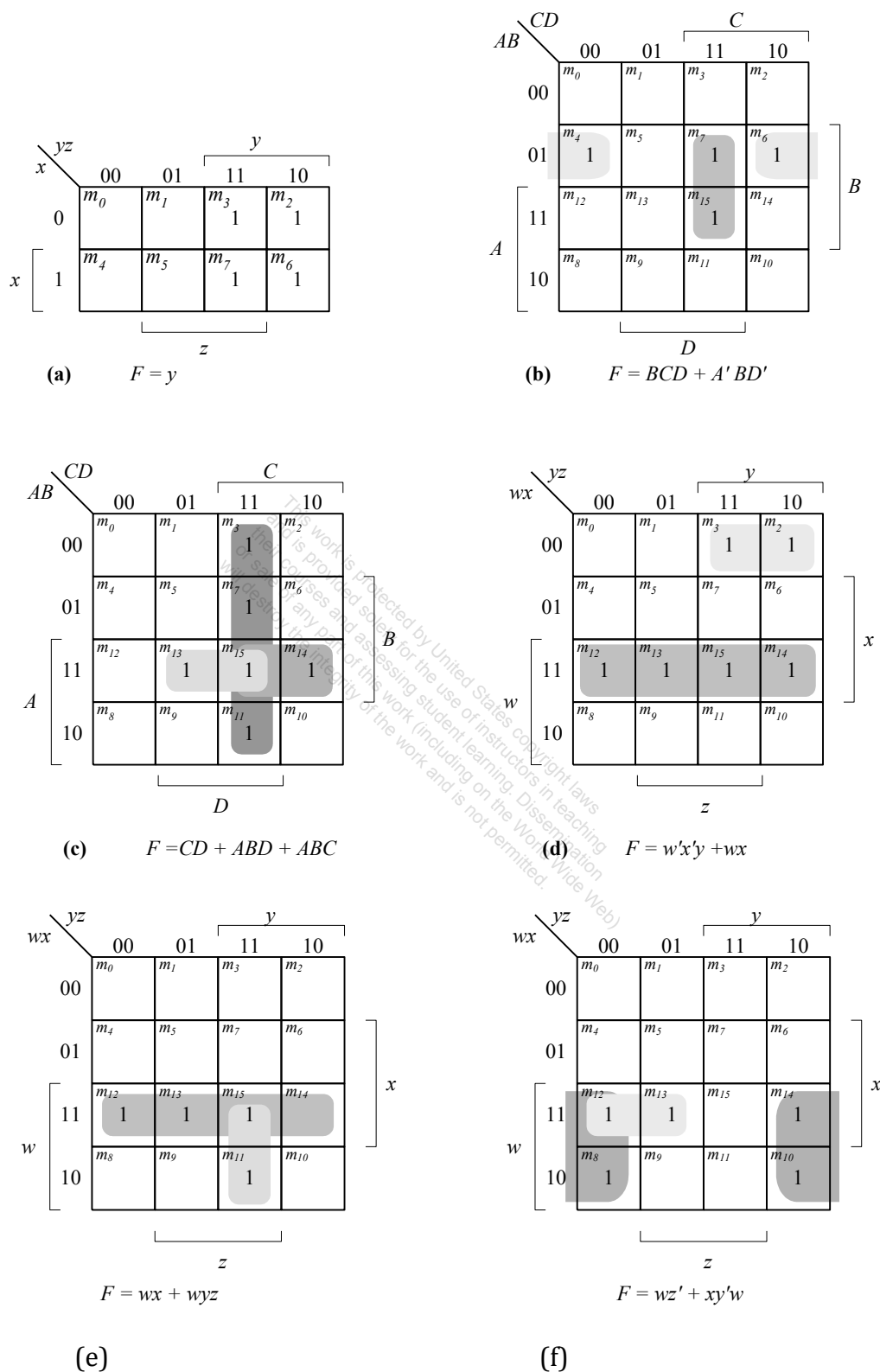
(b)



(c)

(d)

### 3.4





		y			
		yz			
wx		00	01	11	10
		$m_0$	$m_1$	$m_3$	$m_2$
00	00	1	1		
	01	$m_4$	$m_5$	$m_7$	$m_6$
11	11	$m_{12}$	$m_{13}$	1	1
	10	$m_8$	$m_9$	1	1

$F = w'y' + w'y$

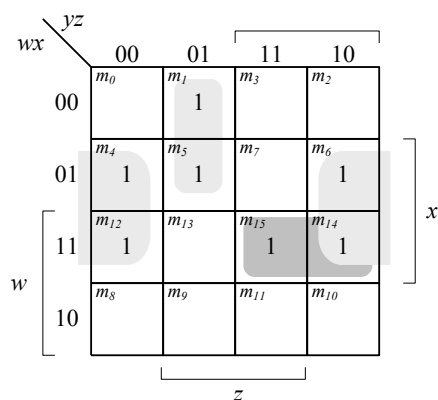
(g)

		y			
		yz			
wx		00	01	11	10
		$m_0$	$m_1$	$m_3$	$m_2$
00	00			1	1
	01	$m_4$	$m_5$	1	1
11	11	$m_{12}$	$m_{13}$	1	
	10	$m_8$	$m_9$	1	

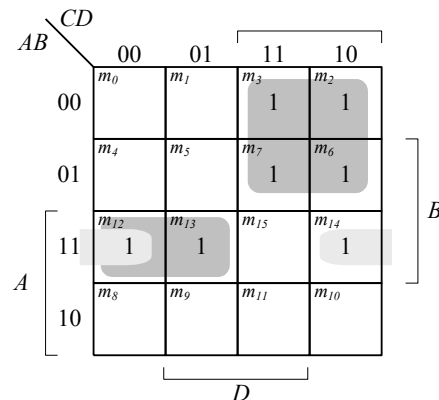
$F = wy' + w'y$

(h)

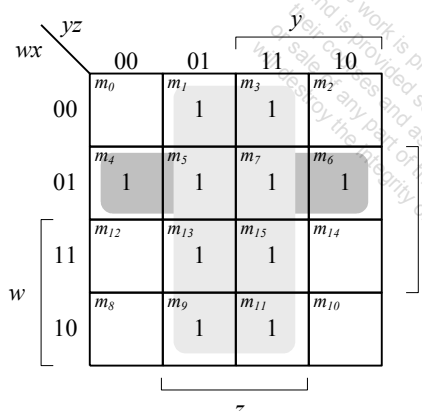
### 3.5



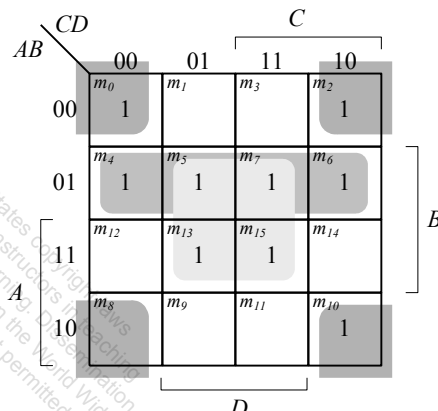
(a)  $F = xz' + w'y'z + wxy$



(b)  $F = A'C + ABC' + ABD'$

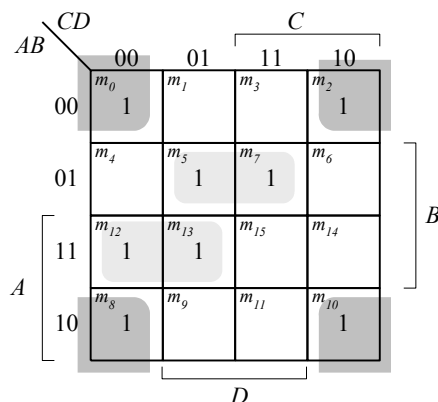


(c)  $F = z + xw'$

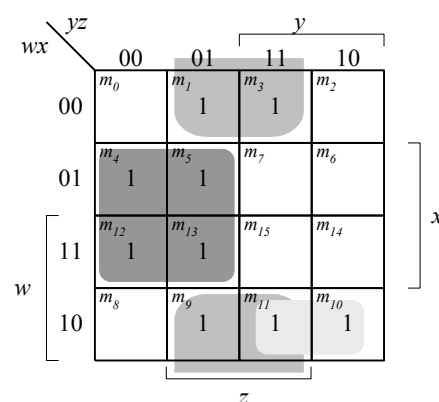


(d)  $F = BD + A'B + B'D'$   
or  $= BD + B'D' + A'D'$

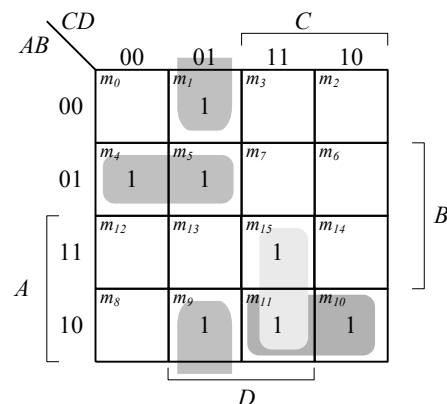
### 3.6



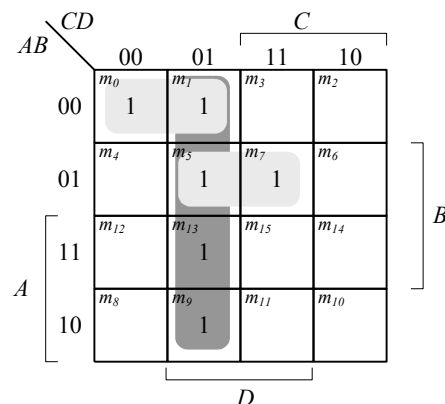
(a)  $F = B'D' + A'BD + ABC'$



(b)  $F = xy' + x'z + wx'y$

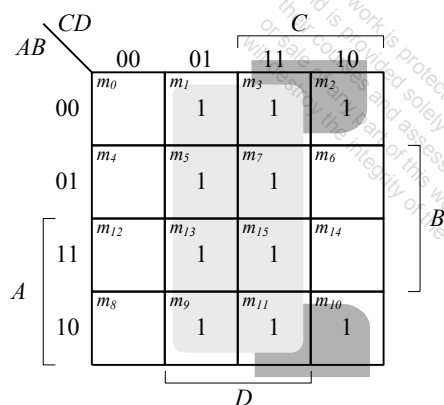


(c)  $F = A'BC' + B'C'D + ACD + AB'C$

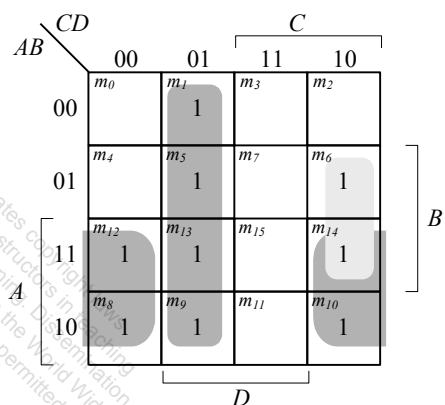


(d)  $F = C'D + A'BD + A'B'C'$

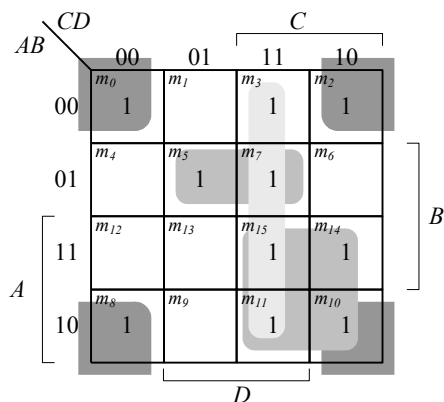
### 3.7



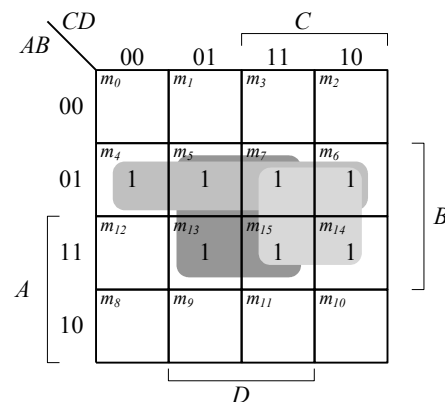
(a)  $F = D + B'C$



(b)  $F = AD' + C'D + BCD'$



(c)  $F = B'D' + AC + A'BD + CD$  (or  $B'C$ )



(d)  $F = A'B + BD + BC$

3.8 (a)  $F(x, y, z) = \Sigma(3, 5, 6, 7)$

		yz			
		00	01	11	10
x	0	$m_0$	$m_1$	$m_3$ 1	$m_2$
	1	$m_4$	$m_5$ 1	$m_7$ 1	$m_6$ 1

$$F = x'yz + xyz + xy'z + xyz'$$

(b)  $F = \Sigma(1, 3, 5, 9, 12, 13, 14)$

		CD			
		00	01	11	10
AB	00	$m_0$	$m_1$ 1	$m_3$ 1	$m_2$
	01	$m_4$	$m_5$ 1	$m_7$	$m_6$
	11	$m_{12}$ 1	$m_{13}$ 1	$m_{15}$	$m_{14}$ 1
	10	$m_8$	$m_9$ 1	$m_{11}$	$m_{10}$

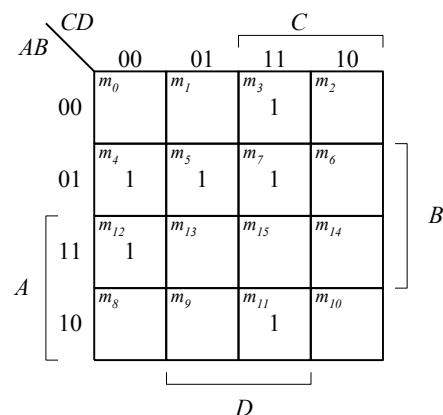
$$F = A'B'C'D + A'BC'D + A'BC'D + ABC'D' + ABC'D + ABCD' + ABC'D$$

(c)  $F = \Sigma(0, 1, 2, 3, 11, 12, 14, 15)$

		yz			
		00	01	11	10
wx	00	$m_0$ 1	$m_1$ 1	$m_3$ 1	$m_2$ 1
	01	$m_4$	$m_5$	$m_7$	$m_6$
	11	$m_{12}$ 1	$m_{13}$	$m_{15}$ 1	$m_{14}$ 1
	10	$m_8$	$m_9$	$m_{11}$ 1	$m_{10}$

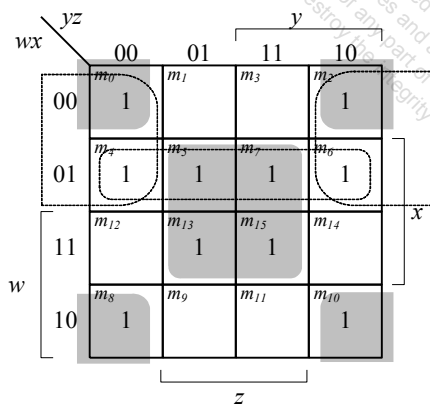
$$F = w'x'y'z' + w'x'y'z + w'x'yz + w'x'yz' + wxy'z' + wxyz + wxyz' + wx'yz$$

(d)  $F = \Sigma(3, 4, 5, 7, 11, 12)$



$$F = A'B'CD + A'BC'D' + A'BC'D + A'BCD + ABC'D' + AB'CD$$

### 3.9



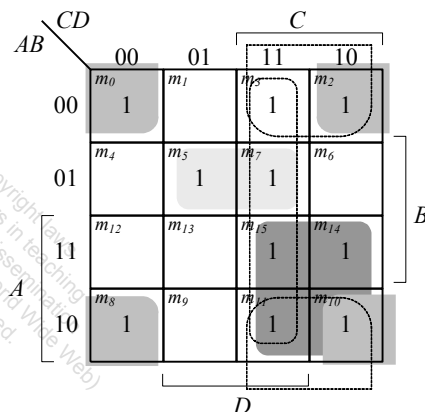
(a)

**Essential:**  $xz, x'z'$

**Non-essential:**  $w'x, w'z'$

$$F = xz + x'z' + (w'x \text{ or } w'z')$$

OR  $B'C$ )

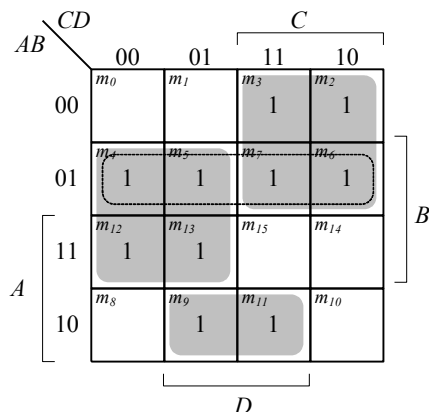


(b)

**Essential:**  $B'D', AC, A'BD$

**Non-essential:**  $CD, B'C$

$$F = B'D' + AC + A'BD + (CD$$

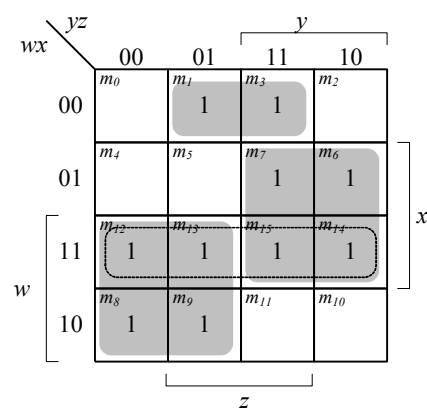


(c)

**Essential:**  $BC'$ ,  $AC$ ,  $A'B'D$

**Non-Essential:**  $A'B$

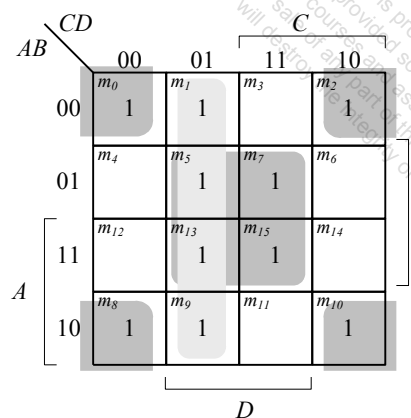
$$F = BC' + AC + A'B'D$$



(d)

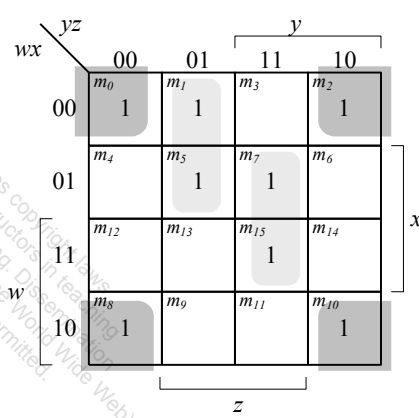
**Essential:**  $wy'$ ,  $xy$ ,  $w'x'z$

$$F = wy' + xy + w'x'z$$



**Essential:**  $BD$ ,  $B'C'$ ,  $C'D$

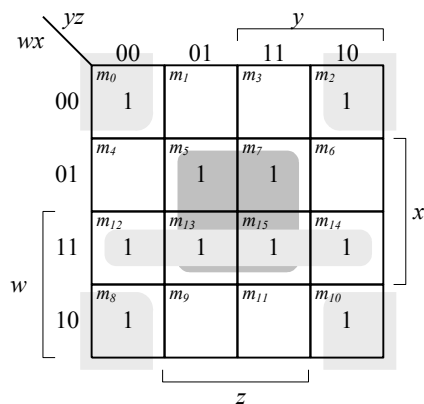
$$F = BD + B'C' + C'D$$



**Essential:**  $x'z'$ ,  $w'y'z$ ,  $xyz$

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

### 3.10

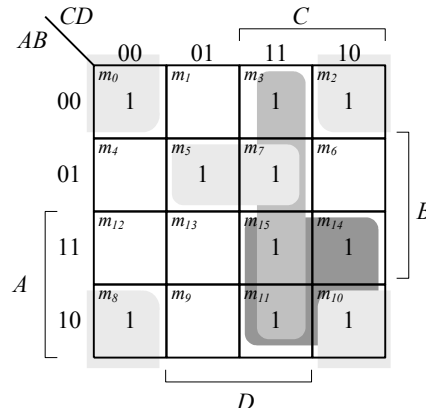


$$F = \Sigma(0, 2, 5, 7, 8, 10, 12, 13, 14, 15)$$

**Essential:**  $xz, wx, x'z'$   
 $A'BD$

$$F = xz + wx + x'z'$$

(a)

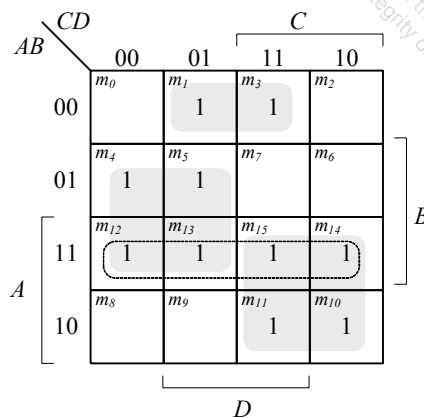


$$F = \Sigma(0, 2, 3, 5, 7, 8, 10, 11, 14, 15)$$

**Essential:**  $AC, B'D', CD, A'BD$

$$F = AC + B'D' + CD + A'BD$$

(b)

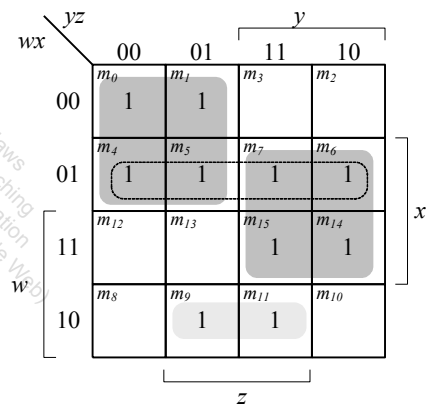


$$F = \Sigma(1, 3, 4, 5, 10, 11, 12, 13, 14, 15)$$

**Essential:**  $AC, BC', A'B'D$   
**Non-essential:**  $AB, A'B'D, B'CD, A'C'D$   
 $w'wz, w'x'z$

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

(c)

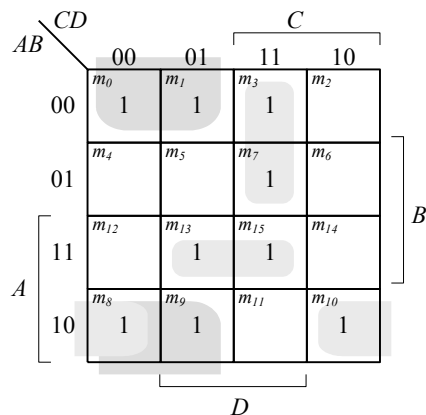


$$F = \Sigma(0, 1, 4, 5, 6, 7, 9, 11, 14, 15)$$

**Essential:**  $w'y', xy, wx'z$   
**Non-essential:**  $wx, x'y'z,$

$$F = w'y' + xy + wx'z$$

(d)



$$F(A, B, C, D) = S(0, 1, 3, 7, 8, 9, 10, 12, 13, 14, 15)$$

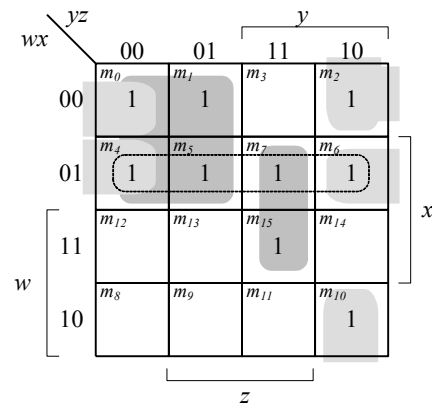
10, 15)

**Essential:**  $B'C'$ ,  $AB'D'$

**Non-essential:**  $ABD$ ,  $A'CD$ ,  $BCD$

$$F = B'C' + AB'D' + A'CD + ABD$$

(e)



$$F = S(0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$$

**Essential:**  $w'y'$ ,  $w'z'$ ,  $xyz$ ,  $x'yz'$

**Non-Essential:**  $w'x$

$$F = w'y' + w'z' + xyz + x'yz'$$

(f)



**3.11 (a)**  $F(w, x, y, z) = \sum (0, 1, 2, 5, 8, 10, 13)$

		yz				
		00	01	11	10	
wx	00	$m_0$ 1	$m_1$ 1	$m_3$	$m_2$ 1	x
	01	$m_4$	$m_5$ 1	$m_7$	$m_6$	
	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$	
	10	$m_8$ 1	$m_9$	$m_{11}$	$m_{10}$ 1	
		z				

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

		yz				
		00	01	11	10	
wx	00	$m_0$	$m_1$	$m_3$ 1	$m_2$	x
	01	$m_4$	$m_5$	$m_7$ 1	$m_6$ 1	
	11	$m_{12}$ 1	$m_{13}$ 1	$m_{15}$ 1	$m_{14}$ 1	
	10	$m_8$	$m_9$ 1	$m_{11}$ 1	$m_{10}$	
		z				

$$F' = wz + xz' + yz$$

$$F = (w' + z')(x' + z)(y' + z')$$

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

**3.12 (a)**

$$F = \Pi(1, 3, 5, 7, 13, 15)$$

$$F = (A' + B' + C' + D)(A' + B' + C + D)(A' + B + C' + D)(A' + B + C + D)(A + B + C' + D)(A + B + C + D)$$

$$F' = ABCD' + ABC'D' + AB'CD' + AB'C'D' + A'B'CD' + A'B'C'D' = \Sigma(0, 2, 8, 10, 12, 14)$$

		CD			
		C		D	
A	AB	00	01	11	10
		$m_0$	$m_1$	$m_3$	$m_2$
	00	0			0
	01	$m_4$	$m_5$	$m_7$	$m_6$
	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
	10	0			0
		$m_8$	$m_9$	$m_{11}$	$m_{10}$
		0			0

$$F' = AD' + B'D' = (A + B')D'$$

$$F = A'B + D$$

**(b)**

$$F = \Pi(1, 3, 6, 9, 11, 12, 14)$$

$$F' = \Sigma(1, 3, 6, 9, 11, 12, 14)$$

$$F = \Sigma(0, 2, 4, 5, 7, 8, 10, 13, 15)$$

$$F = BD + B'D' + A'BC' \text{ (or } A'C'D')$$

		CD			
		C		D	
A	AB	00	01	11	10
		$m_0$	$m_1$	$m_3$	$m_2$
	00		0	0	
	01	$m_4$	$m_5$	$m_7$	$m_6$
					0
	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
	10	0			0
		$m_8$	$m_9$	$m_{11}$	$m_{10}$
			0	0	

**3.13 (a)**  $F = A'C' + B'C' + BC' + AB = A'C' + C' + AB = C' + AB$

		BC			
		00	01	11	10
A	0	m <sub>0</sub> 1	m <sub>1</sub>	m <sub>3</sub>	m <sub>2</sub> 1
	1	m <sub>4</sub> 1	m <sub>5</sub>	m <sub>7</sub> 1	m <sub>6</sub> 1

C

$$F' = A'C + B'C$$

$$F = (A + C')(B + C')$$

**(b)**  $F = ACD' + C'D + AB' + ABCD$

		CD			
		00	01	11	10
AB	00	m <sub>0</sub>	m <sub>1</sub> 1	m <sub>3</sub>	m <sub>2</sub>
	01	m <sub>4</sub>	m <sub>5</sub> 1	m <sub>7</sub>	m <sub>6</sub>
	11	m <sub>12</sub>	m <sub>13</sub> 1	m <sub>15</sub> 1	m <sub>14</sub> 1
	10	m <sub>8</sub> 1	m <sub>9</sub> 1	m <sub>11</sub> 1	m <sub>10</sub> 1

D

$$F = AC + AB' + C'D$$

$$F' = A'C + A'D' + BC'D'$$

$$F = (A + C')(A + D)(B' + C + D)$$

(c)

$$F = (A' + B + D')(A' + B' + C')(A' + B' + C)(B' + C + D')$$

$$F' = AB'D + ABC + ABC' + BC'D$$

CD \ AB		C			
		00	01	11	10
A	00	$m_0$	$m_1$	$m_3$	$m_2$
	01	$m_4$	$m_5$	$m_7$	$m_6$
	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
	10	$m_8$	$m_9$	$m_{11}$	$m_{10}$

D

$$F' = AB + BC'D$$

$$F = (A' + B')(B' + C + D')$$

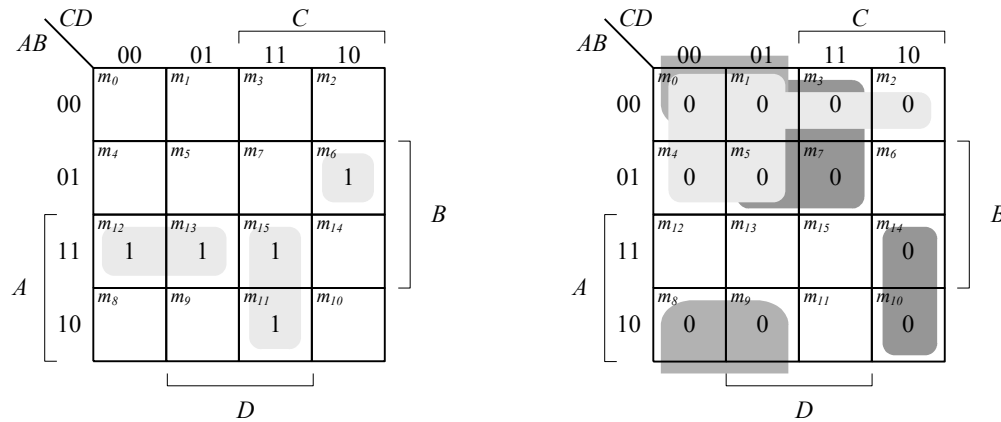
$$F = A'D' + A'BC + AB'$$

CD \ AB		C			
		00	01	11	10
A	00	1	0	0	1
	01	1	0	1	1
	11	0	0	0	0
	10	1	1	1	1

D

(d)

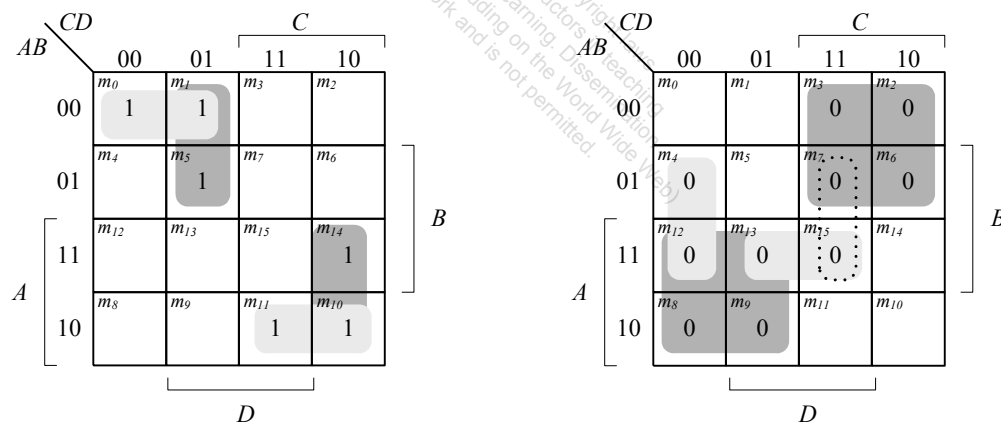
$$F = BCD' + ABC' + ACD$$



$$F' = A'C' + A'D + B'C' + A'B' + ACD'$$

$$F = (A + C)(A + D')(B + C)(A + B)(A' + C' + D)$$

### 3.14



SOP form (using 1s):  $F = A'BC'D + AB'CD + A'B'C' + ACD'$

$$F = A'B'C' + A'C'D + AB'C + ACD'$$

POS form (using 0s):  $F' = AC' + A'C + A'C'D' + ABD$

$$F = (A' + C)(A + C')(A + C + D)(A' + B' + D')$$

Alternative POS:  $F' = AC' + A'C + A'C'D' + BCD$

$$F = (A' + C)(A + C')(A + C + D)(B' + C' + D')$$

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

### 3.15

		$yz$		$y$	
		00	01	11	10
$x$	0	$m_0$	$m_1$	$m_3$	$m_2$
	1	$m_4$	$m_5$	$m_7$	$m_6$
		$z$			
		1	1	x	x
		1	1	1	x
		1	1	x	1

$$F = 1$$

$$F = \Sigma(0, 1, 2, 3, 4, 5, 6, 7)$$

(a)

		$CD$		$C$	
		00	01	11	10
$AB$	00	$m_0$	$m_1$	$m_3$	$m_2$
	01	$m_4$	$m_5$	$m_7$	$m_6$
$A$	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
	10	$m_8$	$m_9$	$m_{11}$	$m_{10}$
		$D$		$B$	
		1	x	1	x
		1	1	1	1
		1	1	1	x

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

$$F = \Sigma(0, 2, 4, 6, 8, 10, 13, 14)$$

(b)

		$CD$		$C$	
		00	01	11	10
$AB$	00	$m_0$	$m_1$	$m_3$	$m_2$
	01	$m_4$	$m_5$	$m_7$	$m_6$
$A$	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
	10	$m_8$	$m_9$	$m_{11}$	$m_{10}$
		$D$		$B$	
		1	1	1	1
		1	1	1	1
		1	1	1	1

$$F = BC + CD + ABD' + A'BD$$

$$F = \Sigma(3, 5, 6, 7, 11, 12, 14, 15)$$

(c)

		$CD$		$C$	
		00	01	11	10
$AB$	00	$m_0$	$m_1$	$m_3$	$m_2$
	01	$m_4$	$m_5$	$m_7$	$m_6$
$A$	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
	10	$m_8$	$m_9$	$m_{11}$	$m_{10}$
		$D$		$B$	
		x	1	1	1
		1	1	1	x
		1	1	1	1

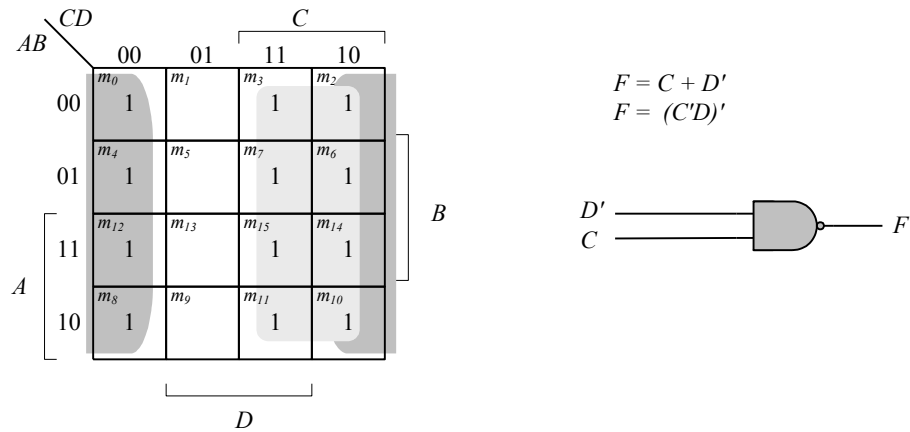
$$F = B'D' + C'D' + A'BC$$

$$F = \Sigma(0, 2, 4, 6, 7, 8, 10,$$

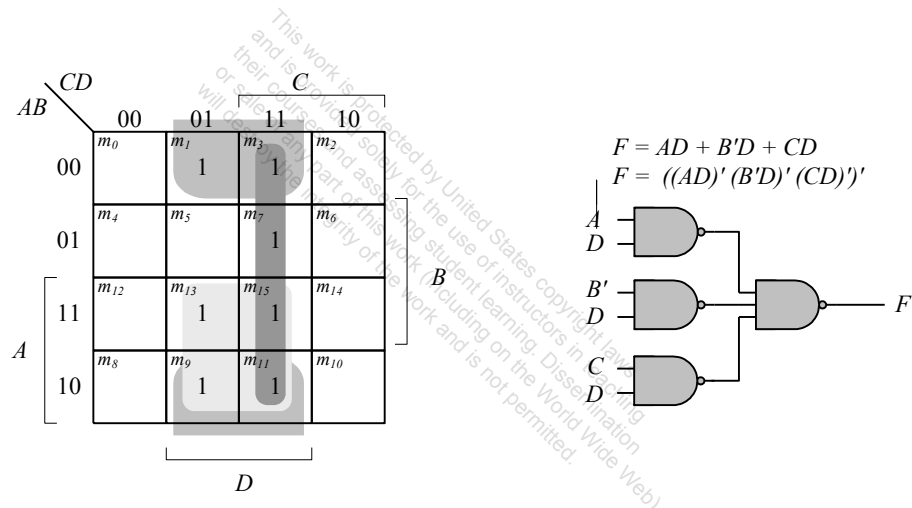
(c)

12)

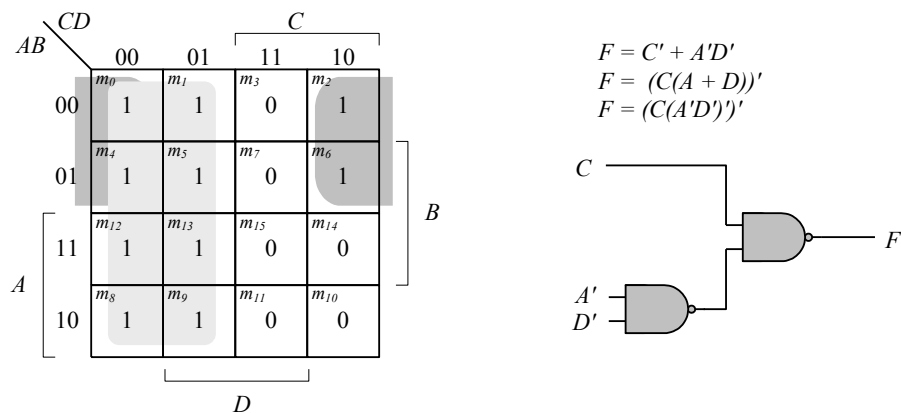
3.16 (a)



(b)



(c)  $F = (A' + C' + D')(A' + C')(C' + D')$   
 $F' = (A' + C' + D')' + (A' + C')' + (C' + D')'$   
 $F' = ACD + AC + CD$





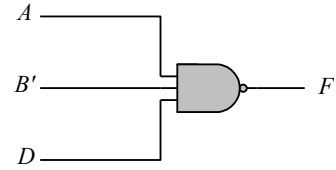
(d)

AB \ CD		C			
		00	01	11	10
A	00	$m_0$ 1	$m_1$ 1	$m_3$ 1	$m_2$ 1
	01	$m_4$ 1	$m_5$ 1	$m_7$ 1	$m_6$ 1
	11	$m_{12}$ 1	$m_{13}$ 1	$m_{15}$ 1	$m_{14}$ 1
	10	$m_8$ 1	$m_9$	$m_{11}$	$m_{10}$ 1

D

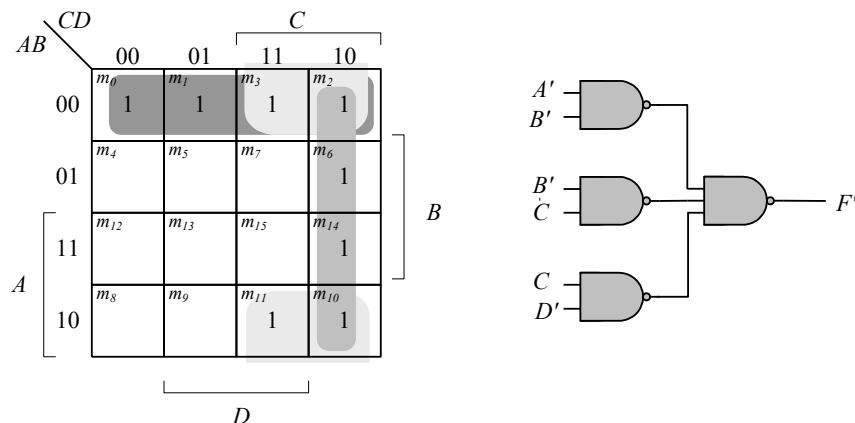
$$F = A' + B + D'$$

$$F = (A(B'D))'$$



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

3.17 (a)



$$F = A'B' + B'C + CD'$$

$$F = ((A + B)(B + C')(C' + D))'$$

$$F = ((A'B')'(B'C)'(CD')')$$

$$F' = (A'B')(B'C)(CD')$$

$$F = A'B' + B'C + CD'$$

$$F = ((A + B)(B + C')(C' + D))'$$

$$F = [(A'B')'(B'C)'(CD')']' \quad (\text{nand gates})$$

(b)

$$F' = m_4 + m_5 + m_7 + m_{12} + m_{13} + m_{15} + m_8 + m_9 + m_{11}$$

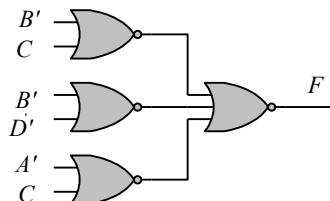
$$F' = BC' + BD + AC'$$

$$F = (BC')'(BD)'(AC')'$$

$$F = (B' + C)(B' + D')(A' + C)$$

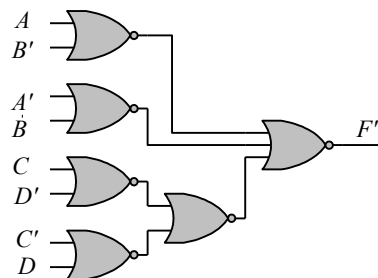
$$F' = (B' + C)' + (B' + D')' + A' + C$$

$$F = [(B' + C)' + (B' + D')' + A' + C]' \quad (\text{nor gates})$$



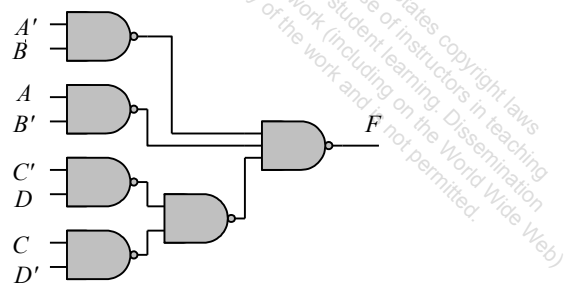
(3.18) (a)

$$\begin{aligned} F(A, B, C, D) &= (A \text{ XOR } B)'(C \text{ XOR } D) \\ &= (A'B + AB')'(C'D + CD') \\ &= ((A'B + AB') + (C'D + CD'))' \\ &= ((A + B')' + (A' + B))' + ((C + D')' + (C' + D))' \quad (\text{nor gates}) \end{aligned}$$

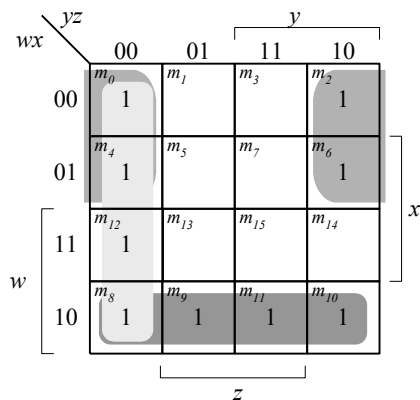


(b)

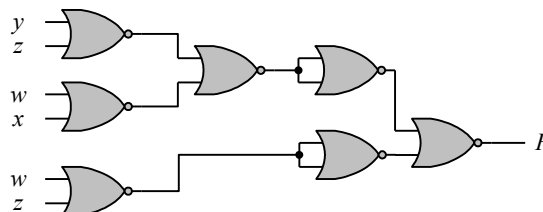
$$\begin{aligned} F(A, B, C, D) &= (A \text{ XOR } B)'(C \text{ XOR } D) \\ &= (A'B + AB')'(C'D + CD') \\ &= (A'B)'(AB')'((C'D)'(CD'))' \quad (\text{nand gates}) \end{aligned}$$



3.19 (a)  $F = (w + z')(x' + z')(w' + x' + y')$

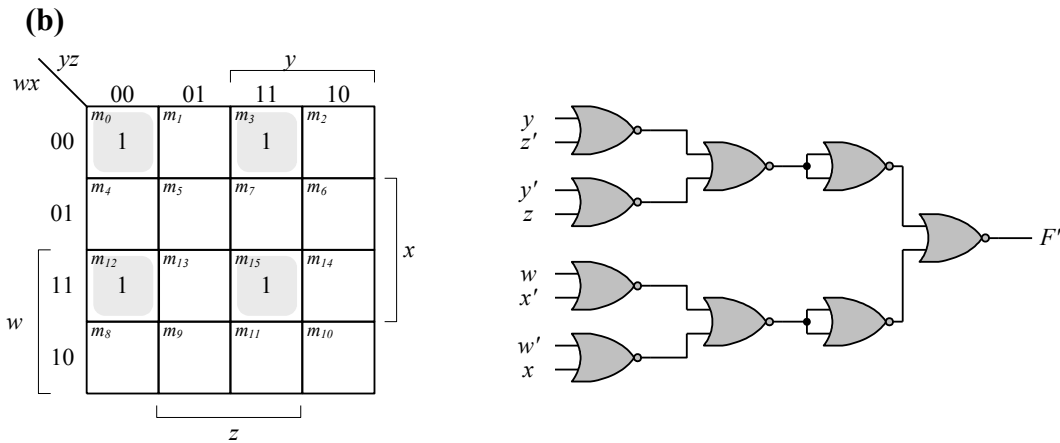


$$F = y'z' + wx' + w'z'$$



$$F = [(y + z)' + (w' + x)' + (w + z)']$$

$$F' = [(y + z)' + (w' + x)' + (w + z)']'$$



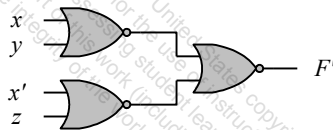
$$F = \Sigma(0, 3, 12, 15)$$

$$F' = y'z + yz' + w'x + wx' = [(y + z')(w + x')(w + x')(w' + x)]'$$

$$F = (y + z)' + (y' + z)' + (w + x)' + (w' + x)'$$

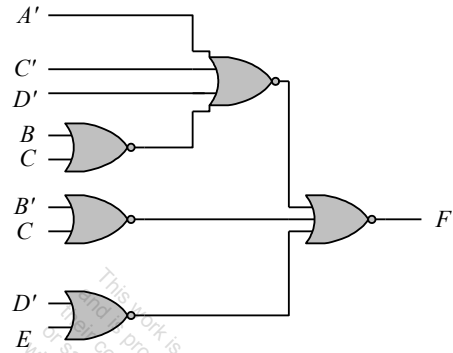
(c)  $F = [(x + y)(x' + z)]' = (x + y)' + (x' + z)'$

$$F' = [(x + y)' + (x' + z)']'$$



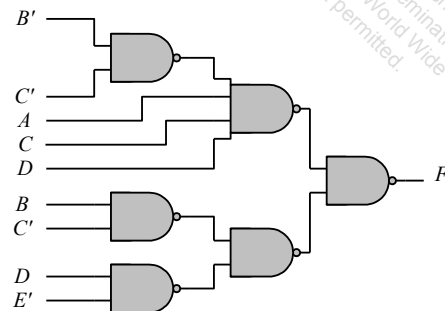
### 3.20 Multi-level NOR:

$$\begin{aligned}
 F &= ACD(B + C) + (BC' + DE') \\
 F' &= [ACD(B + C) + (BC' + DE')]' \\
 F' &= [(A' + C' + D')(B + C) + (B' + C)' + (D' + E)]' \\
 F' &= [((A' + C' + D') + (B + C)')' + (B' + C)' + (D' + E)]' \\
 F' &= [(A' + C' + D' + (B + C)')' + (B' + C)' + (D' + E)]'
 \end{aligned}$$



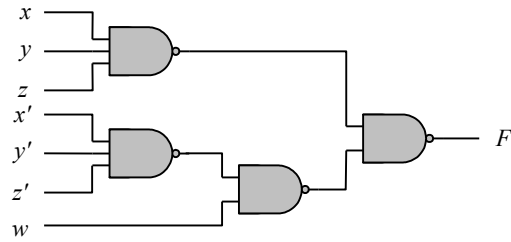
### Multi-level NAND:

$$\begin{aligned}
 F &= CD(B + C)A + (BC' + DE') \\
 F' &= [CD(B + C)A]' [BC' + DE']' \\
 F' &= [CD(B'C)'A]' [BC' + DE']' \\
 F' &= [CD(B'C)'A]' [(BC)' (DE)']'
 \end{aligned}$$



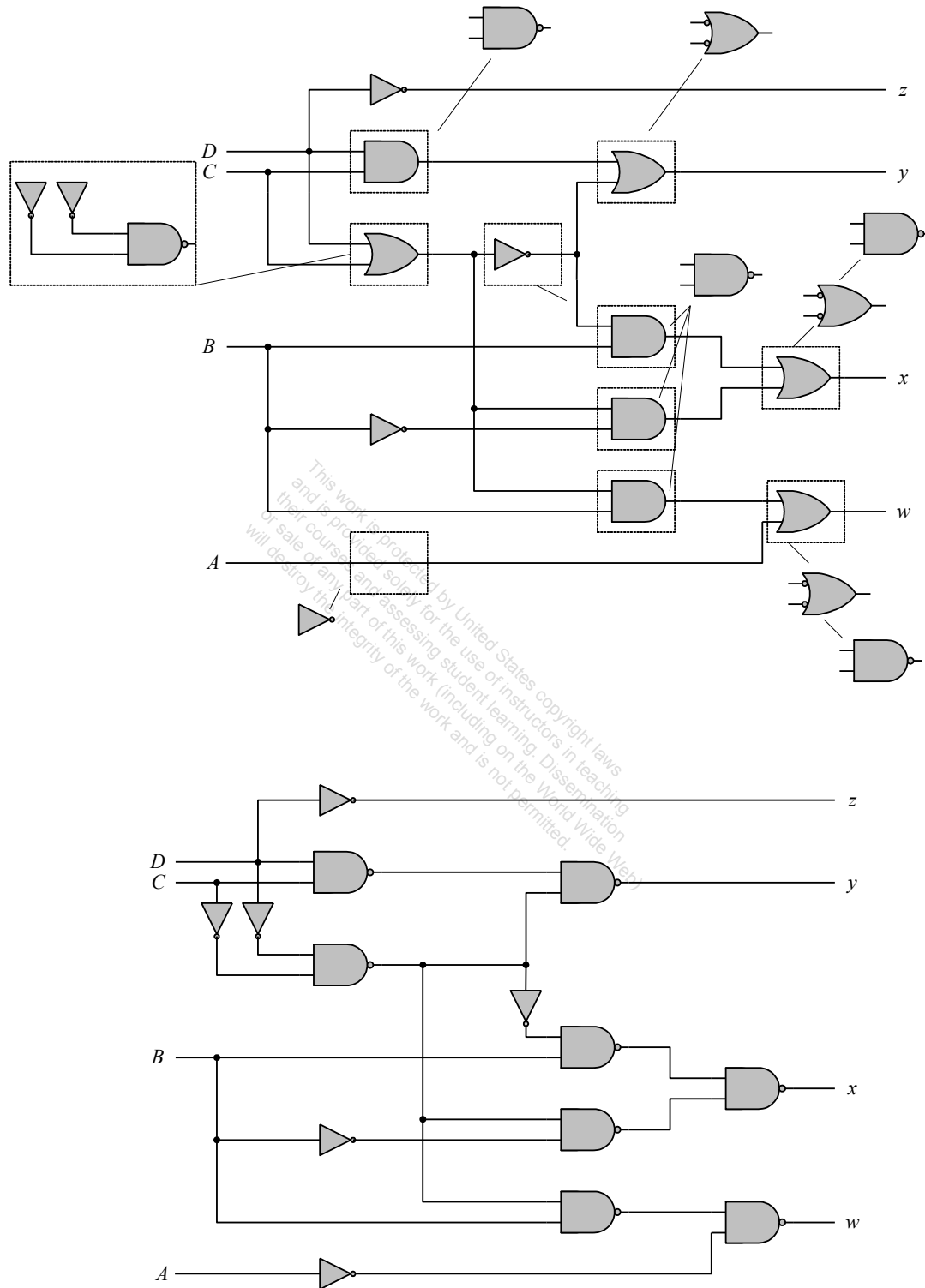
### 3.21

$$\begin{aligned}
 F &= w(x + y + z) + xyz \\
 F' &= [w(x + y + z)][xyz]' = [w(x'y'z')]'(xyz)'
 \end{aligned}$$

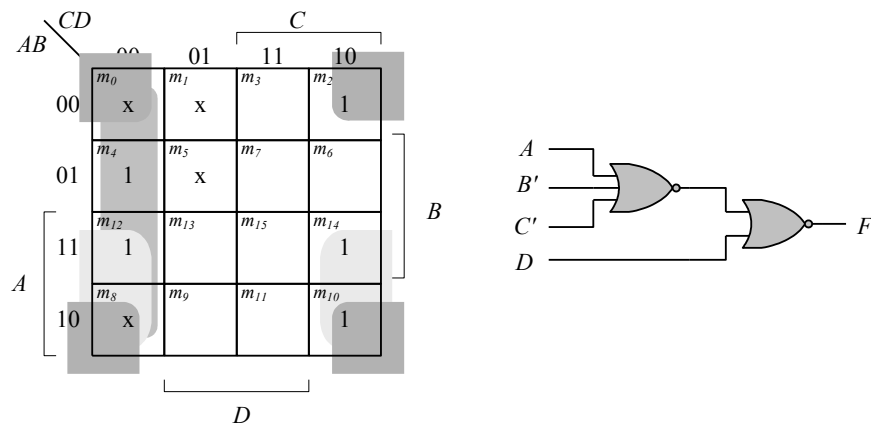


This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

### 3.22



### 3.23

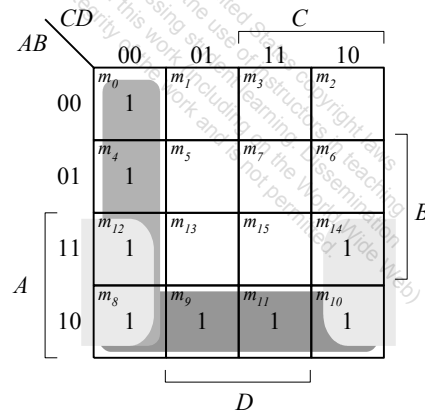


$$F = B'D' + AD' + C'D'$$

$$F' = D + A'BC$$

$$F = [D + A'BC]' = [D + (A + B' + C)']'$$

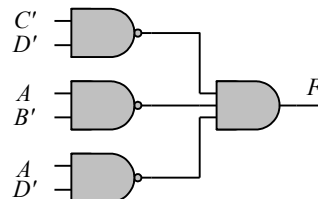
### 3.24 $F(A, B, C, D) = S(0, 4, 8, 9, 10, 11, 12, 14)$



(a)  $F = C'D' + AB' + AD'$

$$F' = (C'D')'(AB')'(AD)'$$

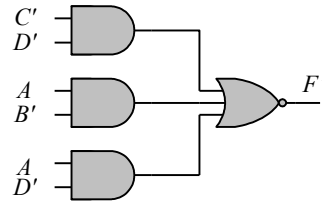
AND-NAND:



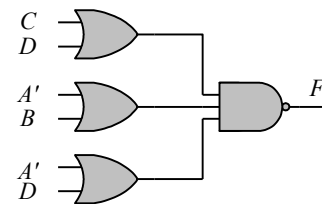
(b)  $F' = [C'D' + AB' + AD']'$

AND-NOR:

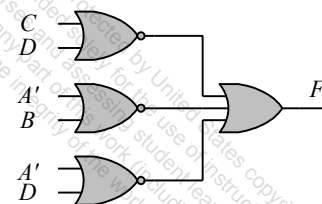




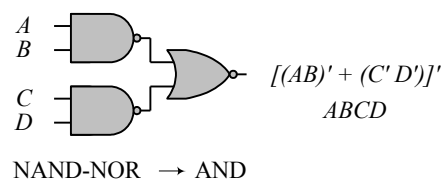
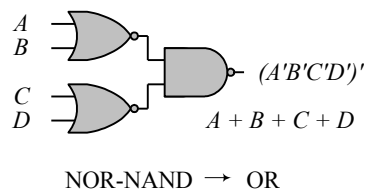
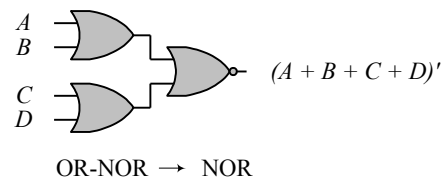
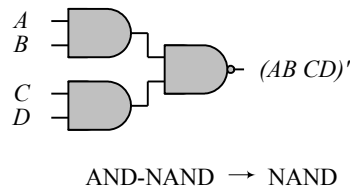
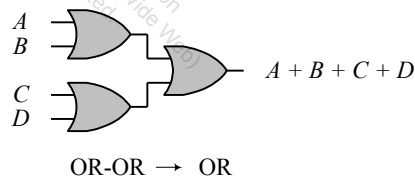
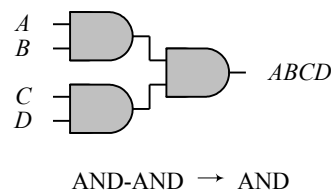
(c)  $F = C'D' + AB' + AD' = (C + D)' + (A' + B)' + (A' + D)'$   
 $F' = (C'D')(AB')(AD')' = (C + D)(A' + B)(A' + D)$   
 $F = [(C + D)(A' + B)(A' + D)]'$   
 OR-NAND:

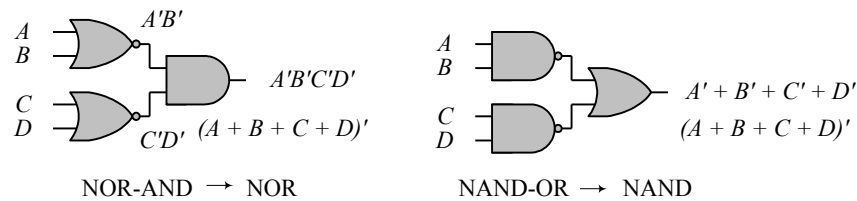


(d)  $F = C'D' + AB' + AD' = (C + D)' + (A' + B)' + (A' + D)'$   
 NOR-OR:



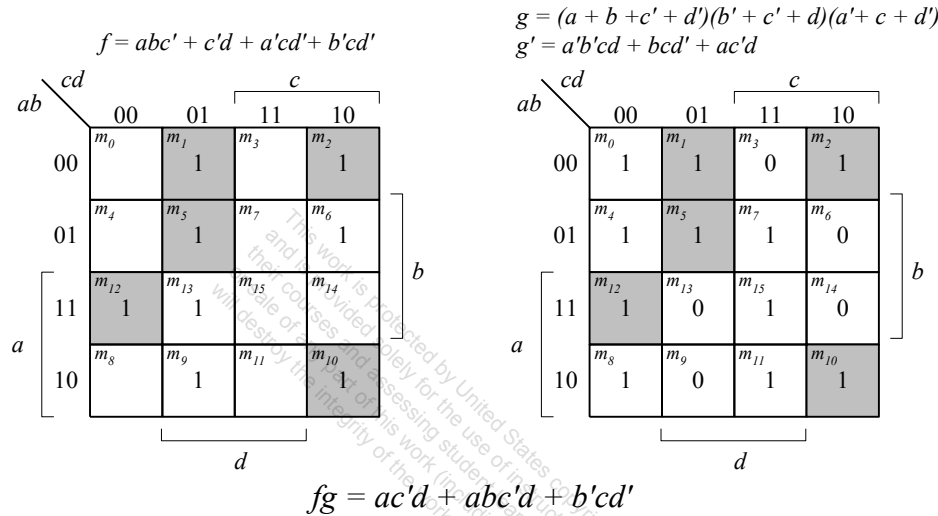
### 3.25





The degenerate forms use 2-input gates to implement the functionality of 4-input gates.

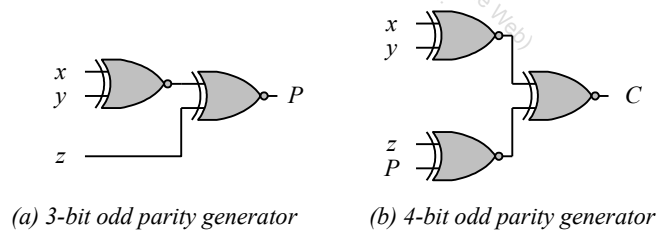
### 3.26



### 3.27

$$x \oplus y = x'y + xy'; \text{ Dual} = (x' + y)(x + y') = (x \oplus y)'$$

### 3.28



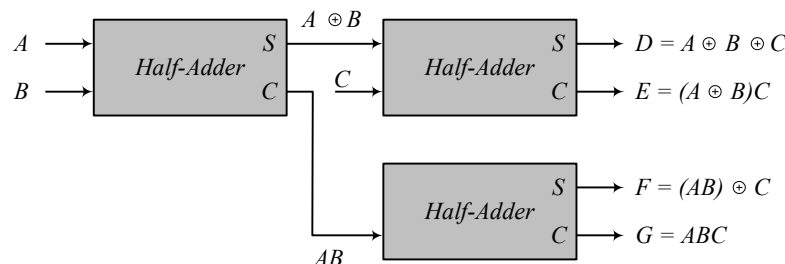
3.29

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

$$E = A'BC + AB'C = (A \oplus B)C$$

$$F = ABC' + (A' + B')C = ABC' + (AB)'C = (AB) \oplus C$$

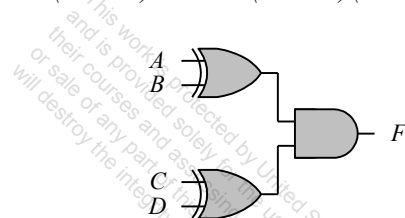
$$G = ABC$$



3.30

$$F = AB'CD' + A'BCD' + AB'C'D + A'BC'D$$

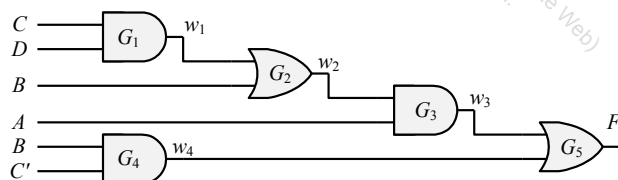
$$F = (A \oplus B)CD' + (A \oplus B)C'D = (A \oplus B)(C \oplus D)$$



3.31

Note: It is assumed that a complemented input is generated by another circuit that is not part of the circuit that is to be described.

From Fig. 3.20a:



(a) Verilog/SystemVerilog

```

module Fig_3_20a_gates (F, A, B, C, C_bar, D);
output F;
input A, B, C, C_bar, D;
wire w1, w2, w3, w4;
and (w1, C, D);
or (w2, w1, B);
and (w3, w2, A);
and (w4, B, C_bar);
or (F, w3, w4);
endmodule
    
```

VHDL

```

entity Fig_3_20a_gates is
  port (F: out Std_Logic; A, B, B_bar, C, C_bar, D: in Std_Logic);
end Fig_3_20a_gates;

architecture Gates of Fig_3_20a_gates is
  component and2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
  component or2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
  signal C: Std_Logic;
begin -- Instantiate components
  G1: and2_gate port map (w1 => y, C => xin1, D => xin2);
  G2: or2_gate port map (w2 => y, w1 => xin1, B => xin2);
  G3: and2_gate port map (w3 => y, w2 => xin1, A => xin2);
  G4: and2_gate port map (w4 => y, B => xin1, C_bar => xin2);
  G5: or2_gate port map (F => y, w3 => xin1, w4 => xin2);
end Gates;

entity and2_gate is
  port (y: out Std_Logic; xin1, xin2: in Std_Logic);
end and2_gate;

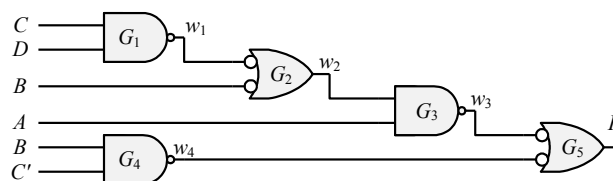
architecture Behavioral of and2_gate is
begin
  y <= xin1 and xin2;
end and2_gate;

entity or2_gate is
  port (y: out Std_Logic; xin1, xin2: in Std_Logic);
end or2_gate;

architecture Behavioral of or2_gate is
begin
  y <= xin1 or xin2;
end or2_gate;

```

From Fig. 3.20b:



## (b) Verilog/SystemVerilog

```

module Fig_3_20b_gates (F, A, B, B_Bar, C, C_bar, D);
  output F;
  input A, B, B_bar, C, C_bar, D;
  wire w1, w2, w3, w4;
  not (w1_bar, w1);
  not (w3_bar, w3);
  not (w4_bar, w4);
  nand (w1, C, D);
  or (w2, w1_bar, B_bar);

```

```
nand    (w3, w2, A);  
nand    (w4, B, C_bar);  
or      (F, w3_bar, w4_bar);  
endmodule
```

This work is protected by United States copyright laws  
and is provided solely for the use of instructors in teaching  
their courses and assessing student learning. Dissemination  
or sale of any part of this work (including on the World Wide Web)  
will destroy the integrity of the work and is not permitted.

## VHDL

```

entity Fig_3_20b_gates is
    port (F: out Std_Logic; A, B, B_bar, C, C_bar, D: in Std_Logic);
end Fig_3_20b_gates;

architecture Gates of Fig_3_20b_gates is
    component nand2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
    component inv_or2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
begin
    G1: nand2_gate port map (w1 => y, C => xin1, D => cin2);
    G2: inv_or2_gate port map (w2 => y, w1 => xin1, B => xin2);
    G3: nand2_gate port map w3 => y, w2 => xin1, A => xin2);
    G4: nand2_gate port map (w4 => y, B => xin1, C_bar => xin2);
    G5: inv_or2_gate port map (F => y, w3 => xin1, w4 => xin2);
end Gates;

entity inv_or2_gate is
    port (y: out Std_logic; xin1, xin2: in Std_Logic);
end nand2_gate;

architecture Behavior of inv_or2_gate is
begin
    y <= not (not(xin1) and not(xin2));
end Behavioral;

entity nand2_gate is
    port (y: out Std_logic; xin1, xin2: in Std_Logic);
end nand2_gate;

architecture Behavior of nand2_gate is
begin
    y <= not (xin1 and xin2);
end Behavioral;

```

(c) **Verilog**

```
module Fig_3_21a_gates (F, A, A_bar, B, B_bar, C, D_bar);
  output F;
  input A, A_bar, B, B_bar, C, D_bar;
  wire w1, w2, w3, w4;
  and (w1, A, B_bar);
  and (w2, A_bar, B);
  or (w3, w1, w2);
  or (w4, C, D_bar);
  and (F, w3, w4);
endmodule
```

**VHDL**

```
entity Fig_3_21a_gates is
  port (F: out Std_Logic; A, S_bar, B, B_bar, C, D_bar: in Std_Logic);
end Fig_3_21a_gates;

architecture Gates_of_Fig_3_24 is
  component and2_gate port (y: out Std_Logic; xin1, xin2: in
Std_Logic);
  component or2_gate port (y: out Std_Logic; xin1, xin2: in
Std_Logic);
  signal w1, w2, w3, w4: Std_Logic;
begin
  G1: and2_gate port map( w1 => y, A => xin1, B_bar => xin2);
  G2: and2_gate port map( w2 => y, A_bar => xin1, B => xin2);
  G3: or2_gate port map( w3 => y, w1 => xin1, w2 => xin2);
  G4: or2_gate port map( w4 => y, C => xin1, D_bar => xin2);
  G5: and2_gate port map( F => y, w3 => xin1, w4 => xin2);
end Gates;
```

(d) **Verilog**

```
module Fig_3_21b_gates (output F, input A, A_bar, B, B_bar, C_bar,
D);
  wire w1, w2, w3, w4, F_bar;
  nand (w1, A, B_bar);
  nand (w2, A_bar, B);
  inv_or2 (w3, w1, w2);
  inv_or2 (w4, C_bar, D);
  nand (F_bar, w3, w4);
  not (F, F_bar);
```

**endmodule**

```
module inv_or2 (y, xin1, xin2);
    assign y = (!xin1) || (!xin2);
endmodule
```

### VHDL

```
entity Fig_3_21b_gates is
    port (F: out Std_Logic; A, A_bar, B, B_bar, C_bar, D: in Std_Logic);
end Fig_3_21b;
```

**architecture** Gates of Fig\_3\_21b **is**

```
    component nand2_gate port (y: out: xin1, xin2: in Std_Logic);
    component inv_or2 port (y: out Std_Logic; xin1: xin2: in
```

Std\_Logic);

```
    component inv_gate port (y: out Std_Logic; x: in Std_Logic);
    signal w1, w2, w3, w4, F_bar: Std_Logic;
```

**begin**

```
    G1: nand2_gate port map (w1 => y, A => xin1, B_bar => xin2);
    G2: nand2_gate port map (w2 => y, A_bar => xin1, B => xin2);
    G3: inv_or2 port map (w3 => y, w1 => xin1, w2 => xin2);
    G4: inv_or2 port map (w4 => y, C_bar => x_in1, D => xin2);
    G5: nand2_gate port map (F_bar => y; => w3 xin1, w4 => xin2);
    G6: inv_gate port map (F => y; F_bar => x);
```

**end** Gates;

(e) **Verilog**

```
module Fig_3_24_gates (F, A, A_bar, B, B_bar, C, D, E_bar);
    output F;
    input A, B, C, D, E_bar;
    wire w1, w2;
    nor (w1, A, B);
    nor (w2, C, D);
    inv3_and M0 (F, w1, w2, w3);
```

**endmodule**

```
module inv3_and (output y, input xin1, xin2, xin3);
    assign y = (!xin1) && (!xin2) && (!xin3);
endmodule
```

### VHDL

```
entity Fig_3_24_gates is
    port (F: out Std_Logic; A, A_bar, B, B_bar, C, D, E_bar: in
Std_Logic);
end Fig_3_24_gates;
```



**architecture Gates of Fig\_3\_24 is**

```

component nor2_gate part (y: out Std_Logic; xin1, xin2: in
Std_Logic);
component inv3_and_gate (y: out Std_Logic; xin1, xin2, xin3: in
Std_Logic);
signal w1, w2: Std_Logic;
begin
    G1: nor2_gate port map (w1 => y; A => xin1, B => xin2);
    G2: nor2_gate port map (w2=> y, C => xin1, D => xin2);
    G3: inv3_and_gate port map (F => y, w1 => xin1, w2 => xin2, E_bar
=> xin3););
end Gates;

```

**entity** inv3\_and\_gate **is**

```

    port (xin1, xin2, xin3: in Std_Logic; y: out Std_Logic);
end inv3_and_gate;

```

**architecture** Operators **of** inv3\_and\_gate **is**

```

begin
    y <= (not xin1) and (not(xin2)) and (not(xin3));
end Operators;

```

(f) **Verilog**

**module** Fig\_3\_25\_gates (F, A, A\_bar, B, B\_bar, C, D\_bar);

**output** F;

**input** A, A\_bar, B, B\_bar, C, D\_bar;

**wire** w1, w2, w3, w4;

**and** (w1, !A\_bar, !B); // alternative would use inverters

**and** (w2, A, !B\_bar);

**nor** (w4, C, D\_bar);

**nor** (w3, w1, w2);

**and** (F, !w3, !w4);

**endmodule**

**VHDL**

**entity** Fig\_3\_25 **is**

```

    port (F: out Std_Logic; A, A_bar, B, B_bar, C, D_bar: in Std_logic);
end Fig_3_25;

```

**architecture** Gates **of** Fig\_3\_25 **is**

```

    component inv2_and_gate port (y: out Std_Logic, xin1, xin2: in
Std_Logic);

```

```

component nor2_gate port (y: out Std_Logic; xin1, xin2: in
Std_Logic);
  signal w1, w2, w3, w4: Std_Logic;
begin
  G1: inv_or2 port map (w1 => y, A_bar => xin1, B => xin2);
  G2: inv_or2 port map (w2 => y, A => xin1, B_bar => xin2);
  G3: nor2_gate port map (w4 => y, C => xin1, D_bar => xin2);
  G4: nor2_gate (w3 => y, w1 => xin1, w2 => xin2);
  G5: inv_and2 port map (F => y, w3 => xin1, w4 => xin2);
end Gates;

entity inv2_and_gate is
  port (y: out Std_Logic; xin1, xin2: in Std_Logic);
end inv3_and_gate;

architecture Operators of inv3_and_gate is
begin
  y <= (not xin1) and (not xin2);
end Operators;

```

**3.32** Note: It is assumed that a complemented input is generated by another circuit that is not part of the circuit that is to be described.

Note: Because the signals here are all scalar-valued, the logical operators (!, &&, and ||) are equivalent to the bitwise operators (~, &, |). If the operands are vectors the bitwise operators produce a vector result; the logical operators would produce a scalar result (i.e., true or false).

(a) **Verilog/SystemVerilog**

```

module Fig_3_20a_CA (F, A, B, C, C_bar, D);
  output F;
  input A, B, C, C_bar, D;
  wire w1, w2, w3, w4;
  assign w1 = C && D;
  assign w2 = w1 || B;
  assign w3 = (w2 && A);
  assign w4 = (B && C_bar);
  assign F = w3 || w4;
endmodule

```

### VHDL

```

entity Fig_3_20a_SA is
  port (A, B_bar, C, D: in Std_Logic);
end

```

**architecture Sig\_Assign of Fig\_3\_20a\_SA is**

    signal w1, w2, w3, w4: Std\_Logic;

**begin**

    w1 <= C and D;

    w2 <= B or w1;

    w3 <= B and A;

    w4 <= B and C\_bar;

    F <= w3 or w4;

**end Sig\_Assign;**

**(b) Verilog**

**module** Fig\_3\_20b\_CA (F, A, B, C, C\_bar, D);

**output** F;

**input** A, B, B\_bar, C, C\_bar, D;

**wire** w2 = !w1;

**wire** w3 = !B\_bar;

**wire** w4, w5, w5\_bar, w6, w6\_bar;

**assign** w1 = !(C && D);

**assign** w4 = w2 || w3;

**assign** w5 = !(w4 && A);

**assign** w5\_bar = !w5;

**assign** w6 = !(B && C\_bar);

**assign** w6\_bar = !w6;

**assign** F = w5\_bar || w6\_bar;

**endmodule**

**VHDL**

**entity** Fig\_3\_20b\_SA is

**port** (F: **out** Std\_Logic; A, B, B\_bar, C, C\_bar, D: **in** Std\_Logic);

**end**

**architecture** Sig\_Assign of Fig\_3\_20b\_SA is

    signal w1, w2, w3, w4, w5, w5\_bar, w6, w6\_bar: Std\_Logic;

**begin**

    w1 <= **not** (C and D);

    w2 <= **not** w1;

    w3 <= **not** B\_bar;

    w4 <= w2 **or** w3;

    w5 <= **not** (w4 and A);

    w5\_bar <= **not** w5;

    w6 <= **not** (B and C\_bar);

    w6\_bar <= **not** w6;

    F <= w5\_bar **or** w6\_bar;

**end Sig\_Assign;**

(c) **Verilog**

```
module Fig_3_21a_CA (F, A, A_bar, B, B_bar, C, D_bar);
  output F;
  input A, A_bar, B, B_bar, C, D_bar;
  wire w1, w2, w3, w4;
  assign w1 = A && B_bar;
  assign w2 = A_bar && B;
  assign w3 = w1 || w2;
  assign w4 = C || D_bar;
  assign F = w3 || w4;
endmodule
```

VHDL

```
entity Fig_3_21a_SA is
  port (F: out Std_Logic; A, A_bar, B, B_bar, C, D_bar: in Std_Logic);
end
architecture Sig_Assign of Fig_3_21a_SA is
  signal w1, w2, w3, w4: Std_Logic;
begin
  w1 <= A and B_bar;
  w2 <= A_bar and B;
  w3 <= w1 or w2;
  w4 <= C or D_bar;
  F <= w3 or w4;
end Sig_Assign;
```

(d) **module** Fig\_3\_21b\_CA (F, A, A\_bar, B, B\_bar, C\_bar, D);

```
  output F;
  input A, A_bar, B, B_bar, C_bar, D;
  wire w1, w2, w1_bar, w2_bar, w3, w4, w5, w6, F_bar;
  assign w1 = !(A && B_bar);
  assign w2 = !(A_bar && B);
  assign w1_bar = !w1;
  assign w2_bar = !w2;
  assign w3 = w1_bar || w2_bar;
  assign w4 = !C_bar;
  assign w5 = !D;
  assign w6 = w4 || w5;
  assign F_bar = !(w3 && w6);
  assign F = !F_bar;
endmodule
```

VHDL

```

entity Fig_3_21b_SA is
  port (F: out Std_Logic; A, A_bar, B, B_bar, C, D: in Std_Logic);
end
architecture Sig_Assign of Fig_3_21b_SA is
  signal w1, w1_bar, w2, w2_bar, w3, w4, w5, w6, F_bar: Std_Logic);
begin
  w1 <= not (A and B_bar);
  w2 <= not (A_bar and B);
  w2_bar <= !w2;
  w1_bar <= not w1;
  w3 <= w1_bar or w2_bar;
  w4 <= not C_bar;
  w5 <= not D;
  w6 <= w4 or w5;
  F_bar <= not (w3 and w6);
  F <= not F_bar;
end Sig_Assign;

```

(e) **Verilog**

```

module Fig_3_24_CA (F, A, B, C, D, E_bar);
  output F;
  input A, B, C, D, E_bar;
  wire w1, w2, w1_bar, w2_bar, w3_bar;
  assign w1 = !(A || B);
  assign w1_bar = !w1;
  assign w2 = !(C || D);
  assign w2_bar = !w2;
  assign w3 = !E_bar;
  assign F = w1_bar && w2_bar && w3;
endmodule

```

### **VHDL**

```

entity Fig_3_24_SA is
  port (F: out Std_Logic; A, B, C, D, E_bar: in Std_Logic);
end Fig_3_24_SA;

architecture Sig_Assign of Fig_3_24_SA is
  signal w1, w2, w1_bar, w2_bar, w3_bar: Std_Logic;

```

```

begin
    w1 <= not (A or B);
    w1_bar <= not w1;
    w2 <= not (C or D);
    w2_bar <= not w2;
    w3 <= not E_bar;
    F <= w1_bar and w2_bar and w3;
endmodule

```

(f) **Verilog**

```

module Fig_3_25_CA (F, A, A_bar, B, B_bar, C, D_bar);
    output F;
    input A, A_bar, B, B_bar, C, D_bar;
    wire w1, w2, w3, w4, w5, w6, w7, w8, w9, w10;
    assign w1 = !A_bar;
    assign w2 = !B;
    assign w3 = w1 && w2;
    assign w4 = !A;
    assign w5 = !B_bar;
    assign w6 = w4 && w5;
    assign w7 = !(C || D_bar);
    assign w8 = !(w3 || w6);
    assign w9 = !w8;
    assign w10 = !w7;
    assign F = w9 && w10;
endmodule

```

**VHDL**

```

entity Fig_3_25_CA is
    port (F: out Std_Logic; A, A_bar, B, B_bar, C, D_bar: in Std_Logic);
end Fig_3_25_CA;

```

**architecture Sig\_Assign of Fig\_3\_25\_SA is**

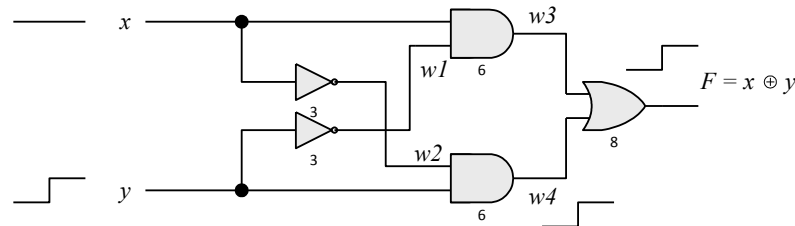
```

    signal w1, w2, w3, w4, w5, w6, w7, w8, w9, w10: Std_Logic;
    w1 <= not A_bar;
    w2 <= not B;
    w3 <= w1 and w2;
    w4 <= not A;
    w5 <= not B_bar;
    w6 <= w4 and w5;
    w7 <= not (C or D_bar);
    w8 <= not (w3 or w6);
    w9 <= not w8;
    w10 <= not w7;
    F <= w9 and w10;
endmodule

```

### 3.33 (a)

Initially, with  $xy = 00$ ,  $w1 = w2 = 1$ ,  $w3 = w4 = 0$  and  $F = 0$ .  $w1$  should change to 0 3ns after  $xy$  changes to 01.  $w4$  should change to 1 6ns after  $xy$  changes to 01.  $F$  should change from 0 to 1 8ns after  $w4$  changes from 0 to 1, i.e., 14 ns after  $xy$  changes from 00 to 01.



### (b) Verilog

```
`timescale 1ns/1ps
```

```
module Prob_3_33 (F, x, y);
  wire w1, w2, w3, w4;
```

```
  and #6 (w3, x, w1);
  not #3 (w1, y);
  and #6 (w4, y, w2);
  not #3 (w2, x);
  or #8 (F, w3, w4);
```

```
endmodule
```

```
module t_Prob_3_33 ();
  reg x, y;
  wire F;
```

```
  Prob_3_33 M0 (F, x, y);
```

```
  initial #200 $finish;
```

```
  initial fork
```

```
    x = 0;
```

```
    y = 0;
```

```
    #10 y = 1;
```

```
  join
```

```
endmodule
```

### VHDL

```
entity Prob_3_33 is
```

```
  port (F: out Std_Logic; x, y: in Std_Logic);
```

```
end Prob_3_33;
```

```
architecture Gates of Fig_3_33 is
```

```
  component and2_gate port (y: out Std_Logic; x, y: in Std_Logic);
```

```
  component inv_gate port (y: out Std_Logic; x: in Std_Logic);
```

```
  signal w1, w2, w3, w4: Std_Logic;
```

```
begin
```

```
  G1: inv_gate port map (w1 => y; y => x);
```

```

G2: inv_gate port map (w2=> y; x => x);
G3: and2_gate port map (w3 => y; x => xin1; w1 => xin2);
G4: and2_gate port map (w4 => x; y => xin1; w2 => xin2);
G5: or2_gate port map (F => y; w3 => xin1; w4 => xin2);
end Gates;

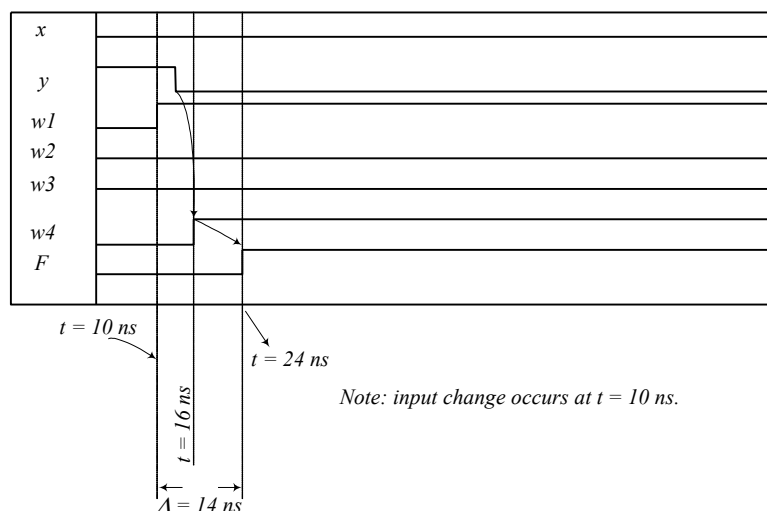
entity t_prob_3_33 is
end t_Prob_3_33;

architecture Behavioral of t_Prob_3_33 is
    component Prob_3_33 port (x, y: in Std_Logic; F: out Std_Logic);
    signal t_x, t_y, t_F: Std_Logic;
begin
    G0: Prob_3_33 port map (t_F => F; t_x => x; t_y => y);
    process ()

end process;
begin
    t_x <= '0';
    t_y <= '0';
    t_y <= '1' after 10 ns;
end Behavioral;

```

(c) To simulate the circuit, it is assumed that the inputs  $xy = 00$  have been applied sufficiently long for the circuit to be stable before  $xy = 01$  is applied. The testbench sets  $xy = 00$  at  $t = 0$  ns, and  $xy = 01$  at  $t = 10$  ns. The simulator assumes that  $xy = 00$  has been applied long enough for the circuit to be in a stable state at  $t = 0$  ns, and shows  $F = 0$  as the value of the output at  $t = 0$ . For illustration, the waveforms show the response to  $xy = 01$  applied at  $t = 10$  ns.



### 3.34 module Prob\_3\_34 (Out\_1, Out\_2, Out\_3, A, B, C, D);

.Digital Design With An Introduction to the Verilog HDL, VHDL, and SystemVerilog, Sixth Edition – Solution Manual.  
M. Mano. M.D. Ciletti, Copyright 2017 .



```

output Out_1, Out_2, Out_3;
input A, B, C, D;
wire A_bar, B_bar, C_bar, D_bar;
assign A_bar = !A;
assign B_bar = !B;
assign C_bar = !C;
assign D_bar = !D;
assign Out_1 = (A + B_bar) && C_bar && ( C || D);
assign Out_2 = ( (C_bar && D) || (B && C && D) || (C && D_bar) ) && (A_bar || B);
assign Out_3 = (((A && B) || C) && D) || (B_bar && C);
endmodule

```

### 3.35

```

module Exmpl-3(A, B, C, D, F)    // Line 1
  inputs A, B, C, Output D, F,  // Line 2
  output B                      // Line 3
  and g1(A, B, B);              // Line 4
  not (D, B, A),                // Line 5
  OR (F, B; C);                // Line 6
endofmodule;                  // Line 7

```

Line 1: Dash not allowed character in identifier; use underscore: Exmpl\_3. Terminate line with semicolon (;).

Line 2: **inputs** should be **input** (no s at the end). Change last comma (,) to semicolon (;). *Output* is declared but does not appear in the port list, and should be followed by a comma if it is intended to be in the list of inputs. If *Output* is a misspelling of **output** and is to declare output ports, *C* should be followed by a semicolon (;) and *F* should be followed by a semicolon (;).

Line 3: *B* cannot be declared both as an input (Line 2) and output (Line 3). Terminate the line with a semicolon (;).

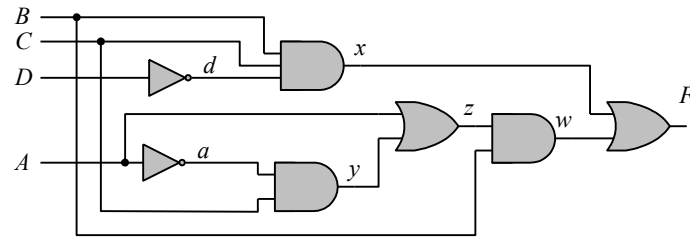
Line 4: *A* cannot be an output of the primitive if it is an input to the module

Line 5: Too many entries for the not gate (may have only a single input, and a single output). Terminate the line with a semicolon, not a comma.

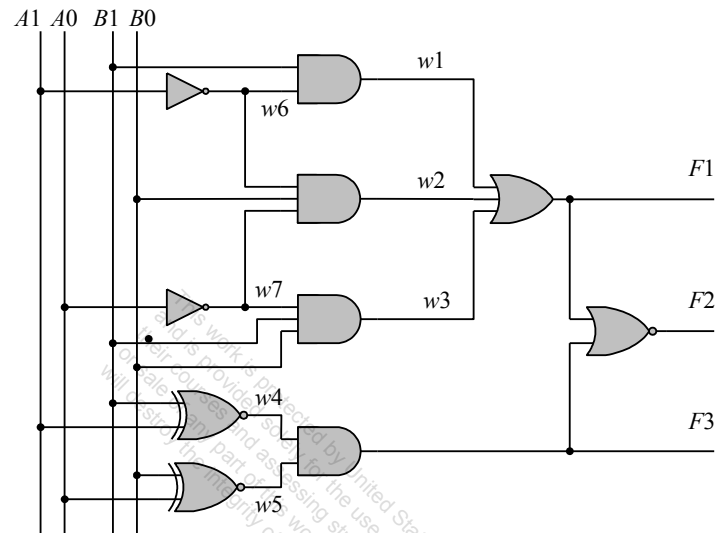
Line 6: OR must be in lowercase: change to “or”. Replace semicolon by a comma (B,) in the list of ports.

Line 7: Remove semicolon (no semicolon after endmodule).

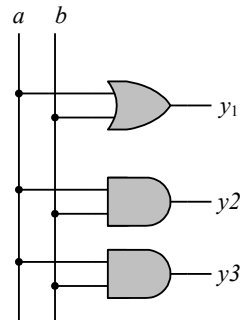
3.36 (a)



(b)



(c)



### 3.37

```

UDP_Majority_4 (y, a, b, c, d);
output      y;
input      a, b, c, d;
table
// a  b  c  d : y
0  0  0  0 : 0;
0  0  0  1 : 0;
0  0  1  0 : 0;
0  0  1  1 : 0;
0  1  0  0 : 0;
0  1  0  1 : 0;
0  1  1  0 : 0;
0  1  1  1 : 1;

1  0  0  0 : 0;
1  0  0  1 : 0;
1  0  1  0 : 0;
1  0  1  1 : 0;
1  1  0  0 : 0;
1  1  0  1 : 0;
1  1  1  0 : 1;
1  1  1  1 : 1;
endtable
endprimitive

```

### 3.38

```

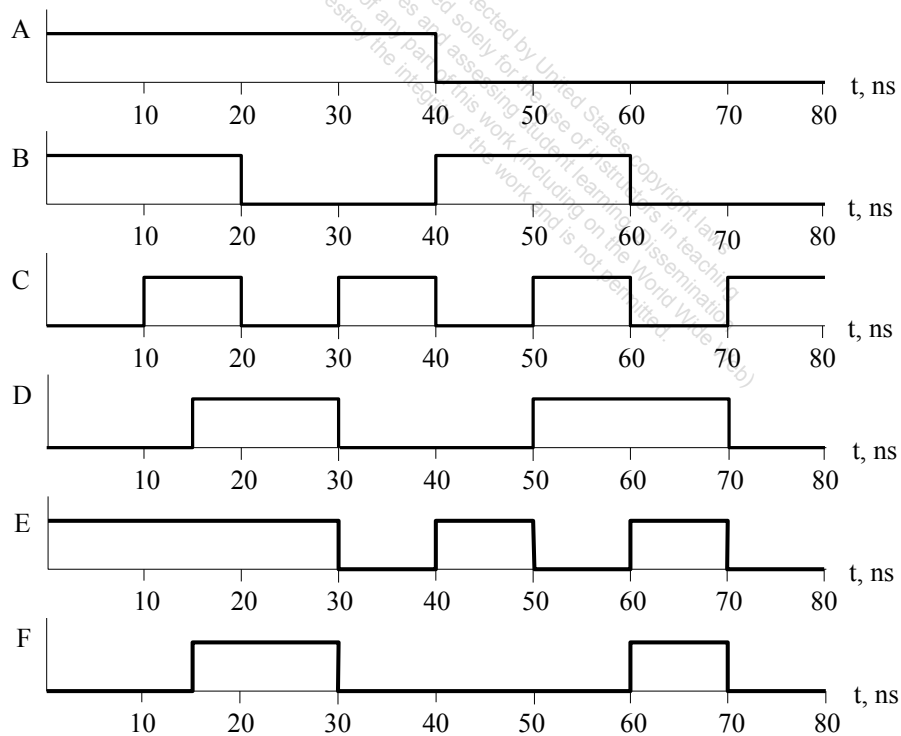
module t_Circuit_with_UDP_02467;
wire E, F;
reg  A, B, C, D;
Circuit_with_UDP_02467 m0 (E, F, A, B, C, D);

initial #100 $finish;
initial fork
    A = 0; B = 0; C = 0; D = 0;
    #40 A = 1;
    #20 B = 1;
    #40 B = 0;
    #60 B = 1;
    #10 C = 1; #20 C = 0; #30 C = 1; #40 C = 0; #50 C = 1; #60 C = 0; #70 C = 1;
    #20 D = 1;
join
endmodule

```

```
// Verilog model: User-defined Primitive
primitive UDP_02467 (D, A, B, C);
  output D;
  input A, B, C;
  // Truth table for D = f (A, B, C) = S (0, 2, 4, 6, 7);
  table
  //  A B C : D // Column header comment
    0 0 0 : 1;
    0 0 1 : 0;
    0 1 0 : 1;
    0 1 1 : 0;
    1 0 0 : 1;
    1 0 1 : 0;
    1 1 0 : 1;
    1 1 1 : 1;
  endtable
endprimitive
// Verilog model: Circuit instantiation of Circuit_UDP_02467
module Circuit_with_UDP_02467 (e, f, a, b, c, d);
  output e, f;
  input a, b, c, d;

  UDP_02467 M0 (e, a, b, c);
  and (f, e, d); //Option gate instance name omitted
endmodule
```



### 3.39

```
a b s c
0 0 0 0
0 1 1 0
1 0 1 0
1 1 0 1
```

```
s = a'b + ab' = a ^ b
c = ab = a && b
```

```
module Prob_3_39 (s, c, a, b);
  input a, b;
  output s, c;

  xor (s, a, b);
  and (c, a, b);
endmodule
```

### 3.40

```
entity Prob_3_39 is
  port ( s, c: out Std_Logic; a, b: in Std_Logic);
end Prob_3_39;
```

```
architecture Gates of Prob_3_40 is
  component xor2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
  component and2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
begin
  G1: xor2_gate port map (s => y, a => xin1, b => xin2);
  G2: and2_gate port map (c => y, a => xin1, b => xin2);
end Gates;
```

```
entity xor2_gate is
  port (y: out Std_logic; xin1, xin2: in Std_Logic);
end xor2_gate;
```

```
architecture Behavior of xor2_gate is
begin
  y <= xin1 xor xin2;
end Behavioral;
```

```
entity and2_gate is
  port (y: out Std_logic; xin1, xin2: in Std_Logic);
end and2_gate;
```

```
architecture Behavior of and2_gate is
begin
  y <= xin1 and xin2;
end Behavioral;
```

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

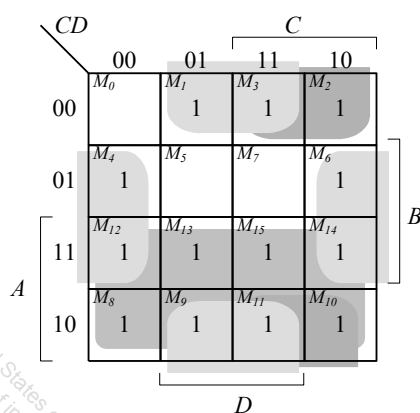
## CHAPTER 4

### 4.1

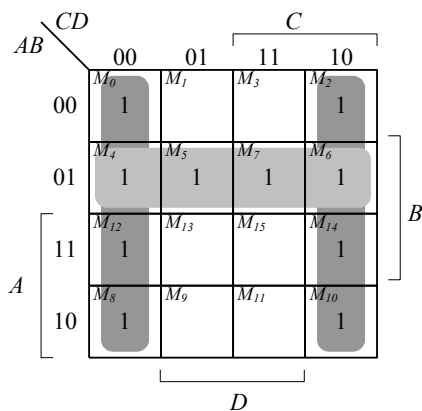
- (a)  $T_1 = B'C$ ,  $T_2 = A'B$ ,  $T_3 = A + T_1 = A + B'C$ ,  
 $T_4 = D \oplus T_2 = D \oplus (A'B) = A'BD' + D(A + B') = A'BD' + AD + B'D$   
 $F_1 = T_3 + T_4 = A + B'C + A'BD' + AD + B'D$   
 With  $A + AD = A$  and  $A + A'BD' = A + BD'$ :  
 $F_1 = A + B'C + BD' + B'D$   
 Alternative cover: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

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

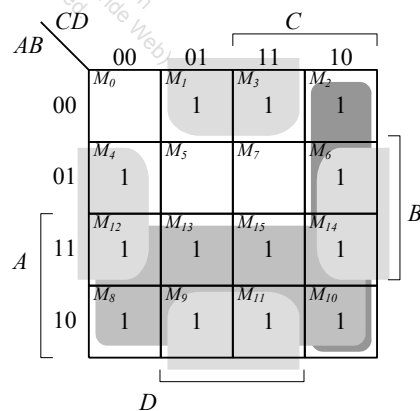
ABCD	$T_1$	$T_2$	$T_3$	$T_4$	$F_1$	$F_2$
0000	0	0	0	0	0	1
0001	0	0	0	1	1	0
0010	1	0	1	0	1	1
0011	1	0	1	1	1	0
0100	0	1	0	1	1	1
0101	0	1	0	0	0	1
0110	0	1	0	1	1	1
0111	0	1	0	0	0	1
1000	0	0	1	0	1	1
1001	0	0	1	1	1	0
1010	1	0	1	0	1	1
1011	1	0	1	1	1	0
1100	0	0	1	0	1	1
1101	0	0	1	1	1	0
1110	0	0	1	0	1	1
1111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



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



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

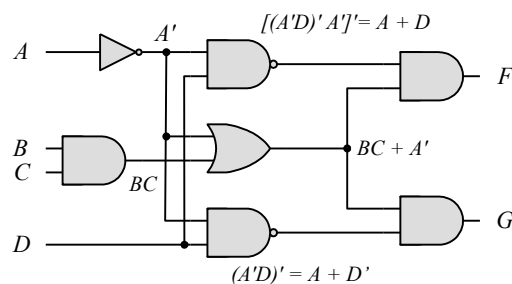


$$F_1 = A + CD' + B'D + BD'$$

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

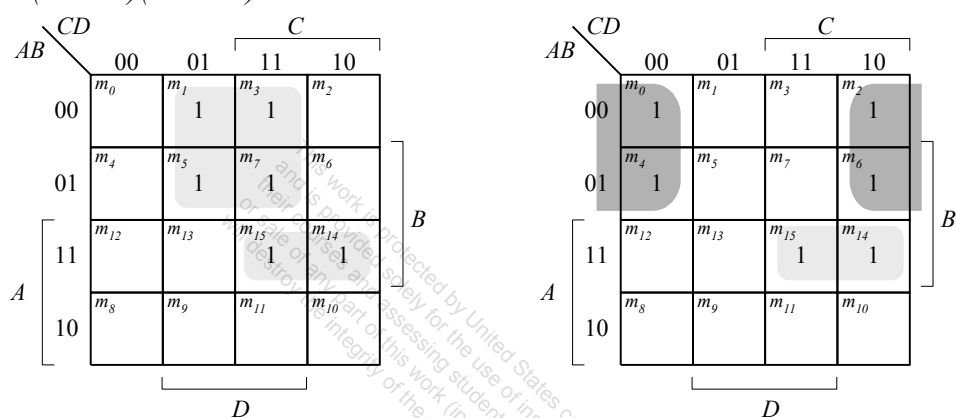


## 4.2



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

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



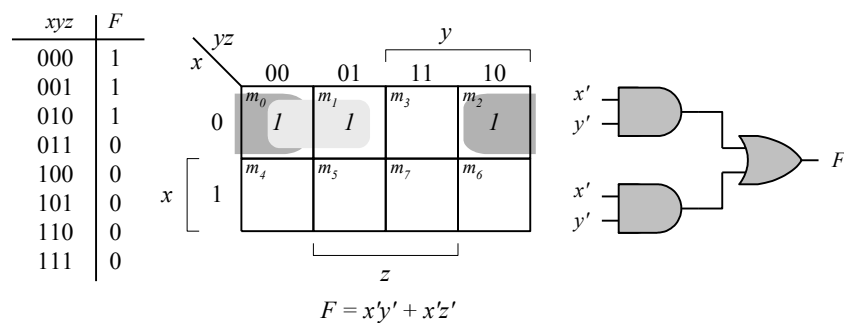
$$F = A'D + ABC + BCD = A'D + ABC$$

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

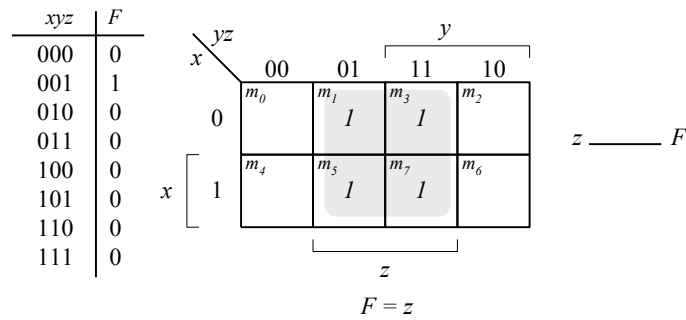
4.3 (a)  $Y_i = (A_i S' + B_i S)E'$  for  $i = 0, 1, 2, 3$

(b) 1024 rows and 14 columns

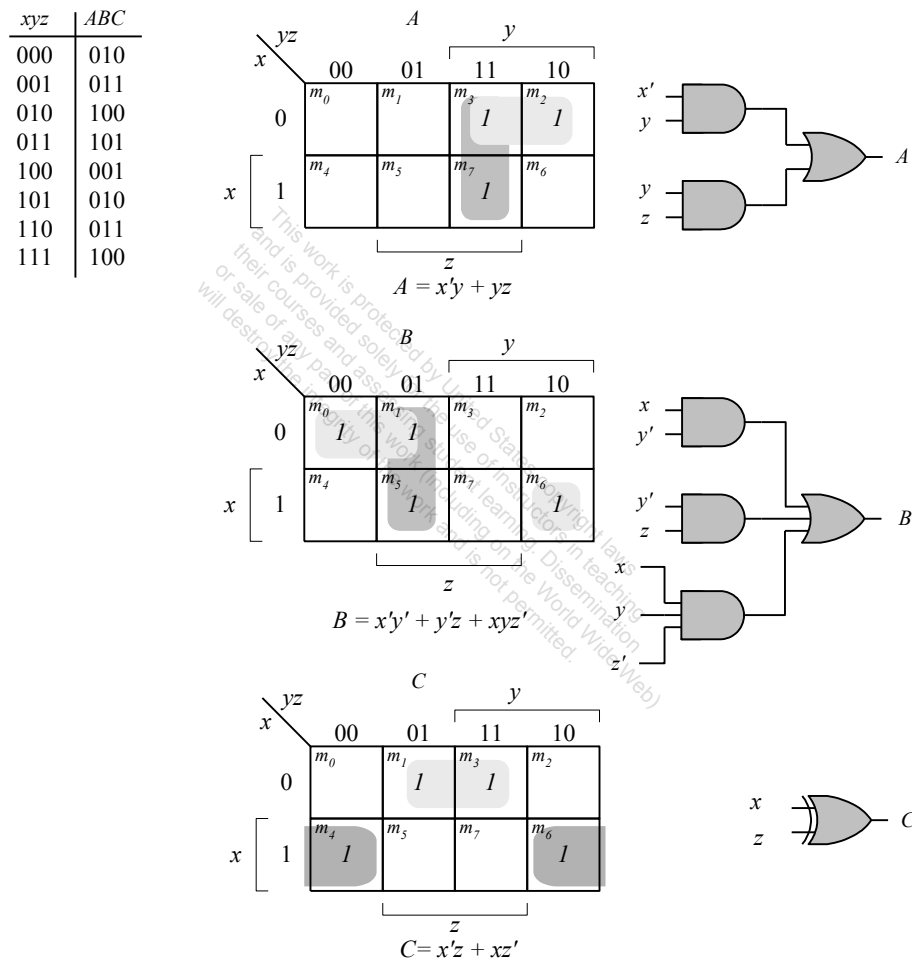
4.4 (a)



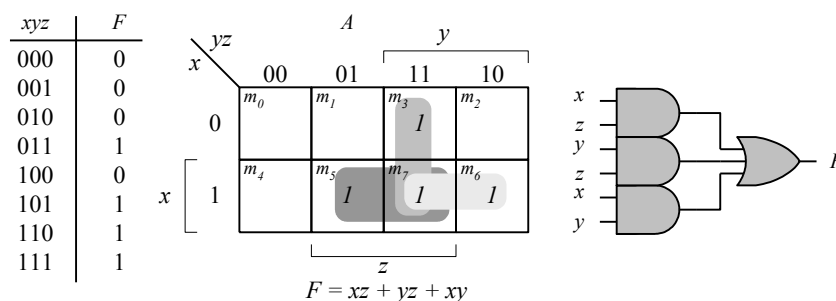
(b)



4.5



## 4.6



### Verilog

```

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

```

### VHDL

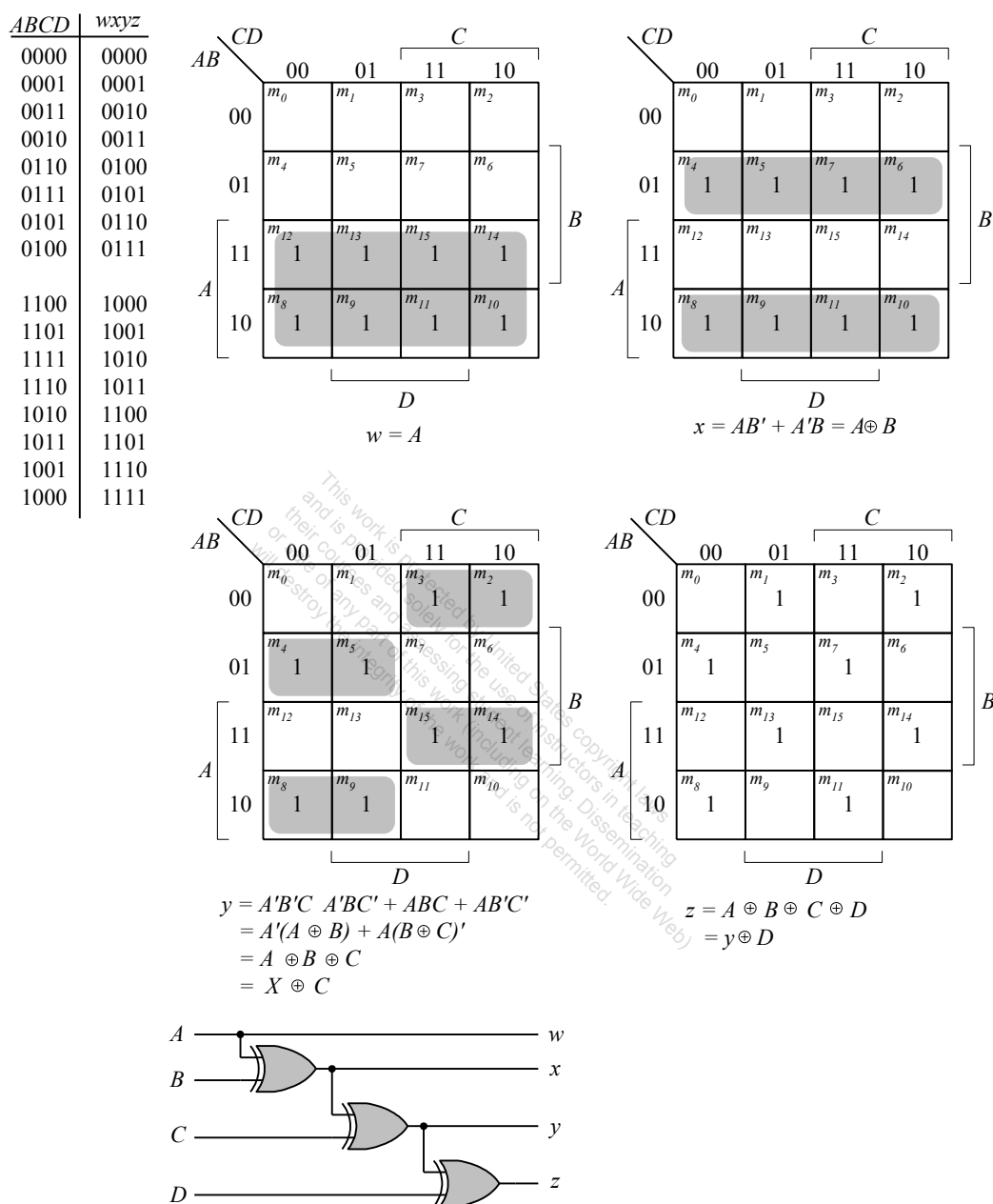
```

entity Prob_4_6 is
    port ( F: out Std_Logic; x, y, z: in Std_Logic);
end Prob_4_6;

architecture Sig_Assign of Prob_4_6 is
    begin
        F<= (x and y) and (y and z) and (x and y)
    end name;

```

# 4.7 (a)



(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

```

## VHDL

```

entity Prob_4_7 is
    port ( w, x, y, z: out Std_Logic; A,B, C, D: in Std_Logic);
end Prob_4_7;

```

```

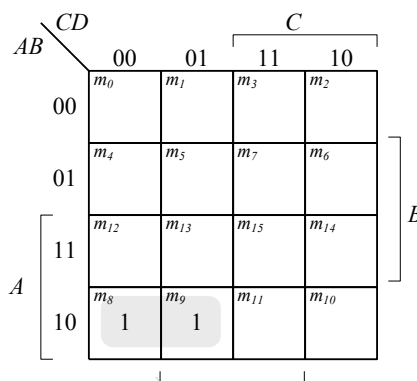
architecture Sig_Assign of Prob_4_7 is
begin
    w <= A;
    x <= A xor B;
    y <= x xor C;
    z <= y xor D;
end Sig_Assign;

```

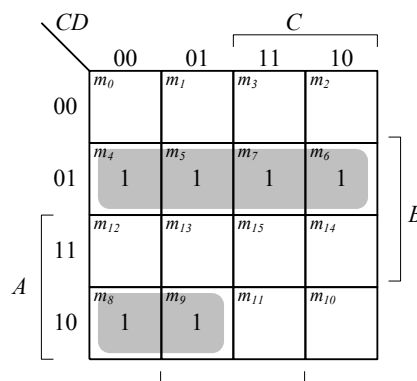
4.8 (a) The 8-4-2-1 code (Table 1.5) and the BCD code (Table 1.4) are identical for digits 0 – 9.

(b)

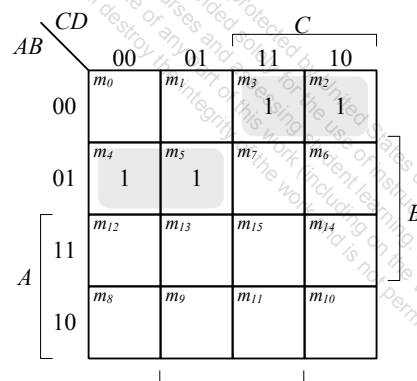
8421 ABCD	Gray wxyz
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101



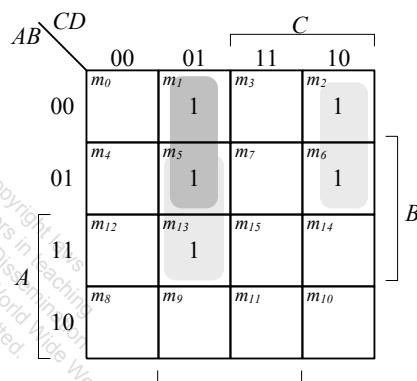
$$w = AB'C'$$



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



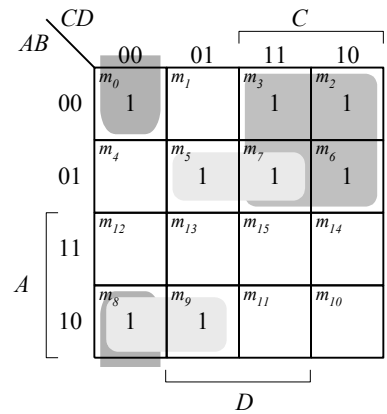
$$y = A'BD' + A'B'D$$



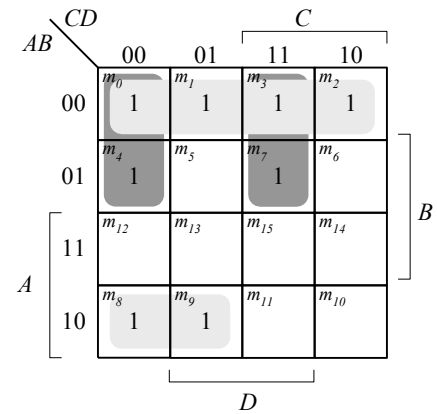
$$z = A'C'D + BC'D + A'CD'$$

## 4.9

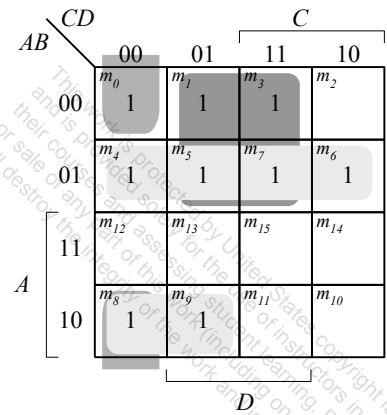
ABCD	a	b	c	d	e	f	g
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1	1	1	0	0	1
0100	0	1	1	0	0	1	1
0101	1	0	1	1	0	1	1
0110	1	0	1	1	1	1	1
0111	1	1	1	0	0	0	0
1000	1	1	1	1	1	1	1
1001	1	1	1	1	0	1	1



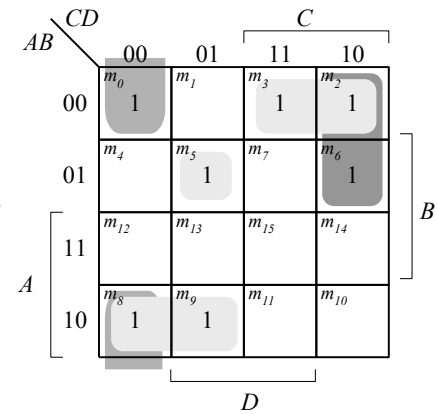
$$a = A'C + A'BD + B'C'D' + AB'C'$$



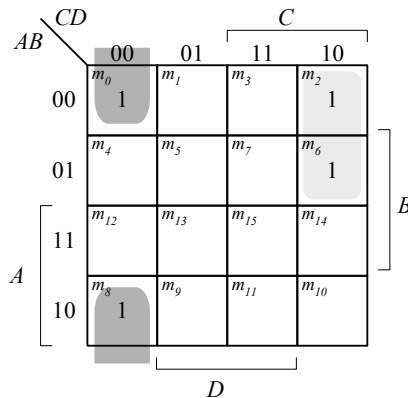
$$b = A'B' + A'C'D' + A'CD + AB'C'$$



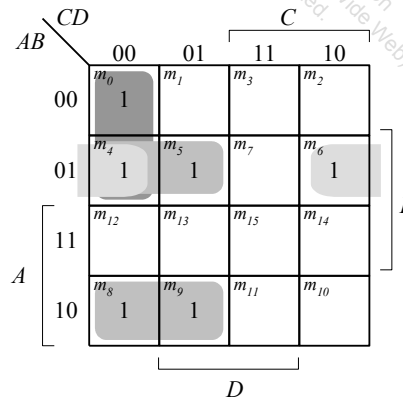
$$c = A'B + A'D + B'C'D' + AB'C'$$



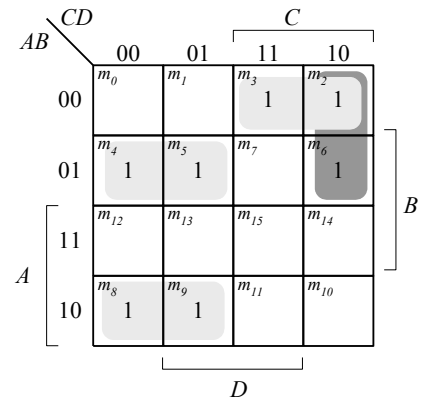
$$d = A'CD' + A'B'C + B'C'D' + AB'C'D$$



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



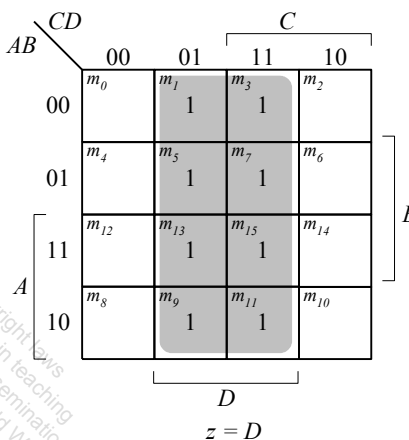
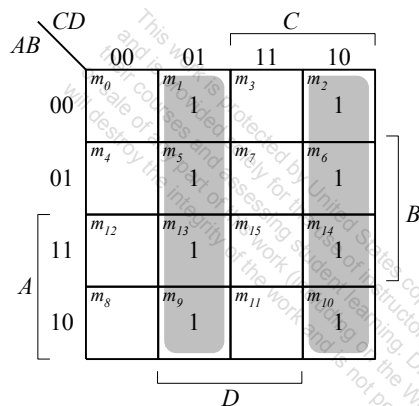
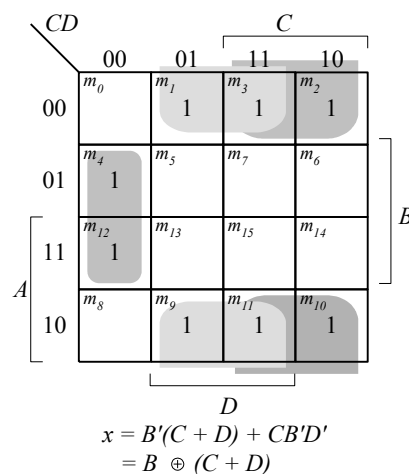
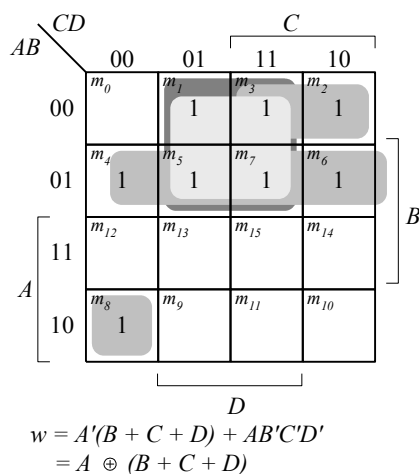
$$f = A'BC' + A'C'D' + A'BD + AB'C'$$



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

## 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



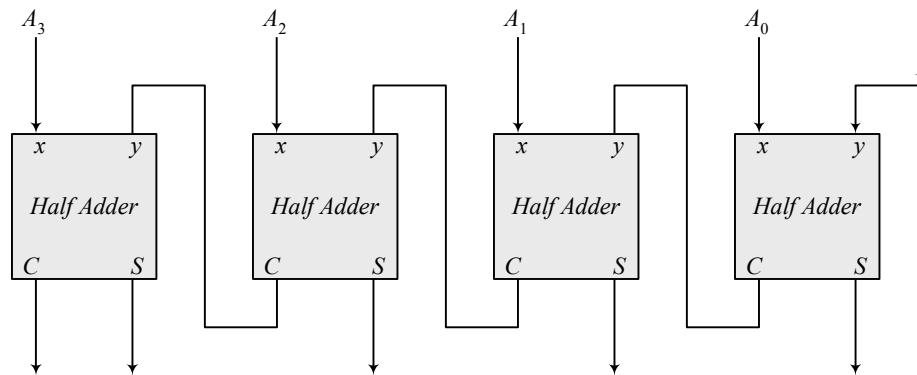
For a 5-bit 2's complementer with input E and output v:

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



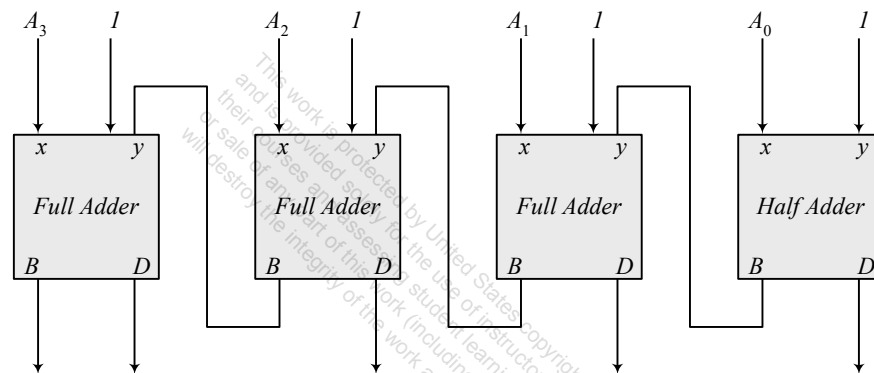
# 4.11

(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

$$10 + 5 + 5 + 10 = 30 \text{ ns}$$

**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)

$$\begin{aligned} (C_i'G_i' + p_i)' &= (C_i + G_i)P_i = G_iP_i + P_iC_i \\ &= A_iB_i(A_i + B_i) + P_iC_i \\ &= A_iB_i + P_iC_i = G_i + P_iC_i \\ &= A_iB_i + (A_i + B_i)C_i = A_iB_i + A_iC_i + B_iC_i = C_{i+1} \\ (P_iG_i') \oplus C_i &= (A_i + B_i)(A_iB_i)' \oplus C_i = (A_i + B_i)(A_i' + B_i') \oplus C_i \\ &= (A_i'B_i + A_iB_i') \oplus C_i = A_i \oplus B_i \oplus C_i = S_i \end{aligned}$$

(b)

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

**4.17**    (a)

$$\begin{aligned} (C_i'G_i' + P_i)' &= (C_i + G_i)P_i = G_iP_i + P_iC_i = A_iB_i(A_i + B_i) + P_iC_i \\ &= A_iB_i + P_iC_i = G_i + P_iC_i \\ &= A_iB_i + (A_i + B_i)C_i = A_iB_i + A_iC_i + B_iC_i = C_{i+1} \end{aligned}$$

$$\begin{aligned} (P_iG_i') \oplus C_i &= (A_i + B_i)(A_iB_i)' \oplus C_i = (A_i + B_i)(A_i' + B_i') \oplus C_i \\ &= (A_i'B_i + A_iB_i') \oplus C_i = A_i \oplus B_i \oplus C_i = S_i \end{aligned}$$

(b)

$$\text{Output of NOR gate} = (A_0 + B_0)' = P'_0$$

Output of NAND gate =  $(A_0B_0)' = G'_0$

$$S_0 = (P_0G'_0) \oplus C_0$$

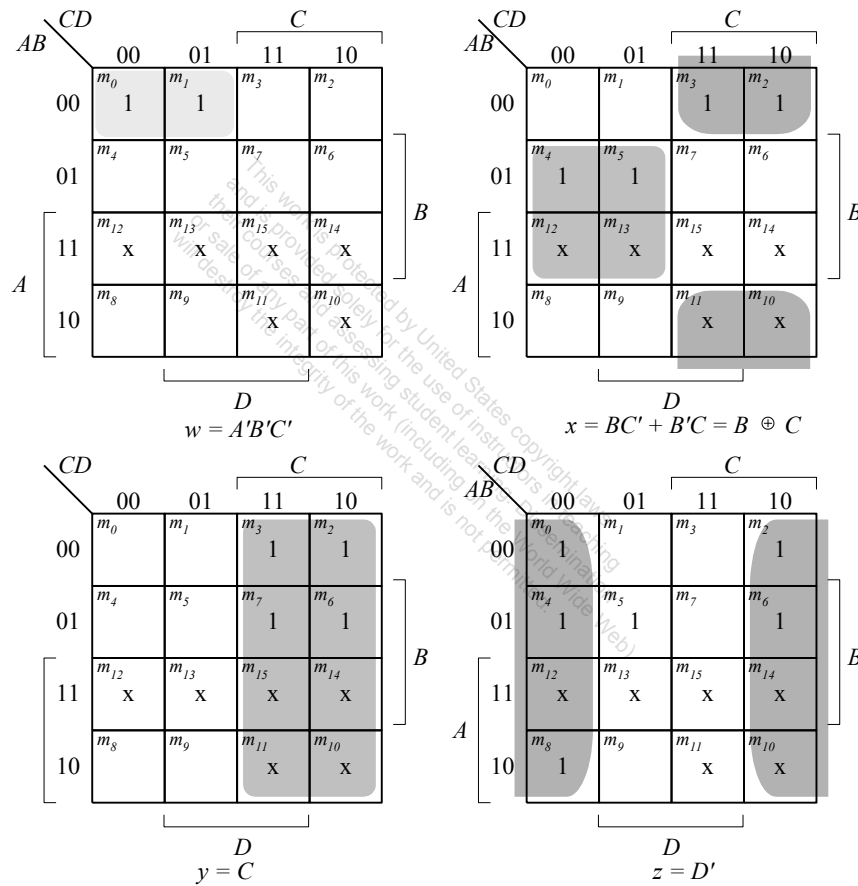
$$C_1 = (C'_0G'_0 + P'_0)' \quad \text{as defined in part (a)}$$

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

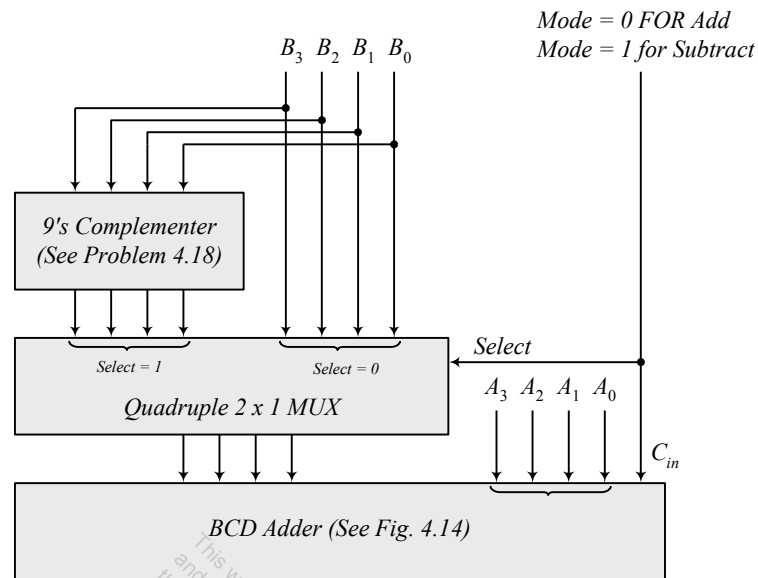
# 4.18

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

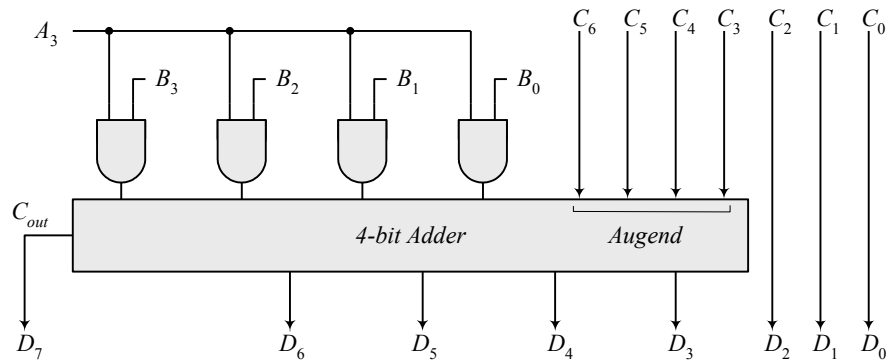
$$d(A, b, c, d) = \Sigma(10, 11, 12, 13, 14, 15)$$



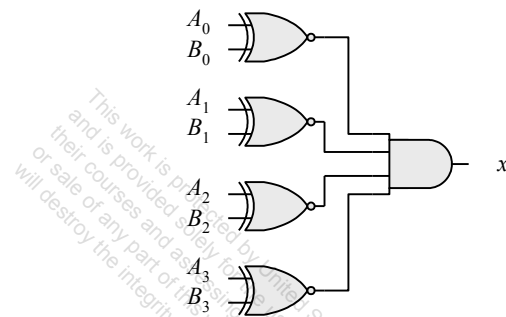
# 4.19



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



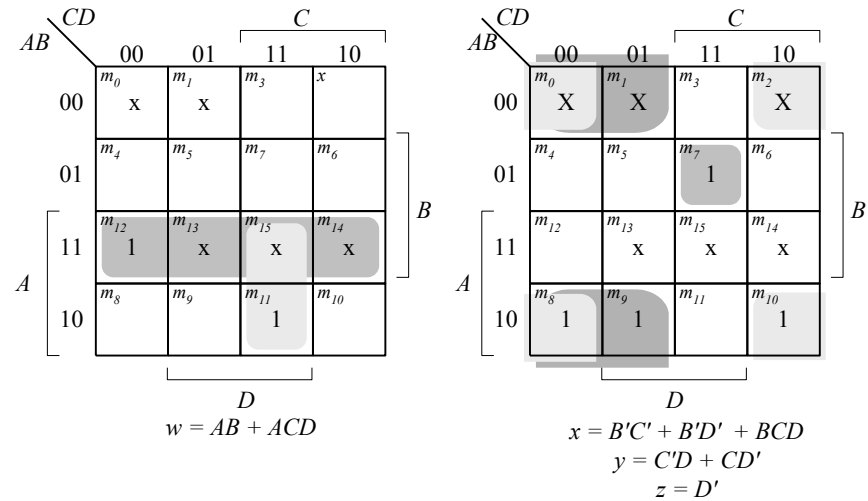
**4.21**



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

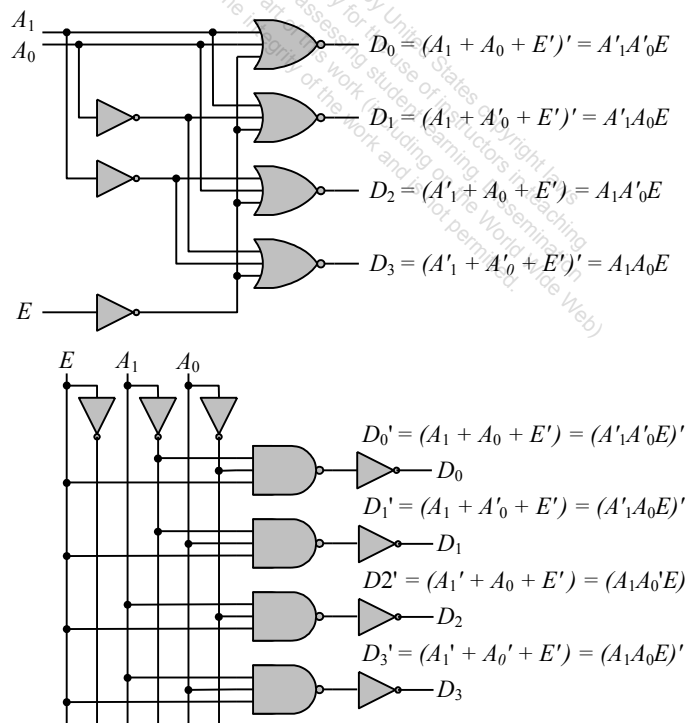
**4.22**

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

$$\begin{aligned} D_0 &= A_1'A_0' = (A_1 + A_0)' \text{ (NOR)} & D_0' &= (A_1'A_0')' \text{ (NAND)} \\ D_1 &= A_1'A_0 = (A_1 + A_0')' \text{ (NOR)} & D_1' &= (A_1'A_0)' \text{ (NAND)} \\ D_2 &= A_1A_0' = (A_1' + A_0)' \text{ (NOR)} & D_2' &= (A_1A_0')' \text{ (NAND)} \\ D_3 &= A_1A_0 = (A_1' + A_0')' \text{ (NOR)} & D_3' &= (A_1A_0)' \text{ (NAND)} \end{aligned}$$



## 4.24

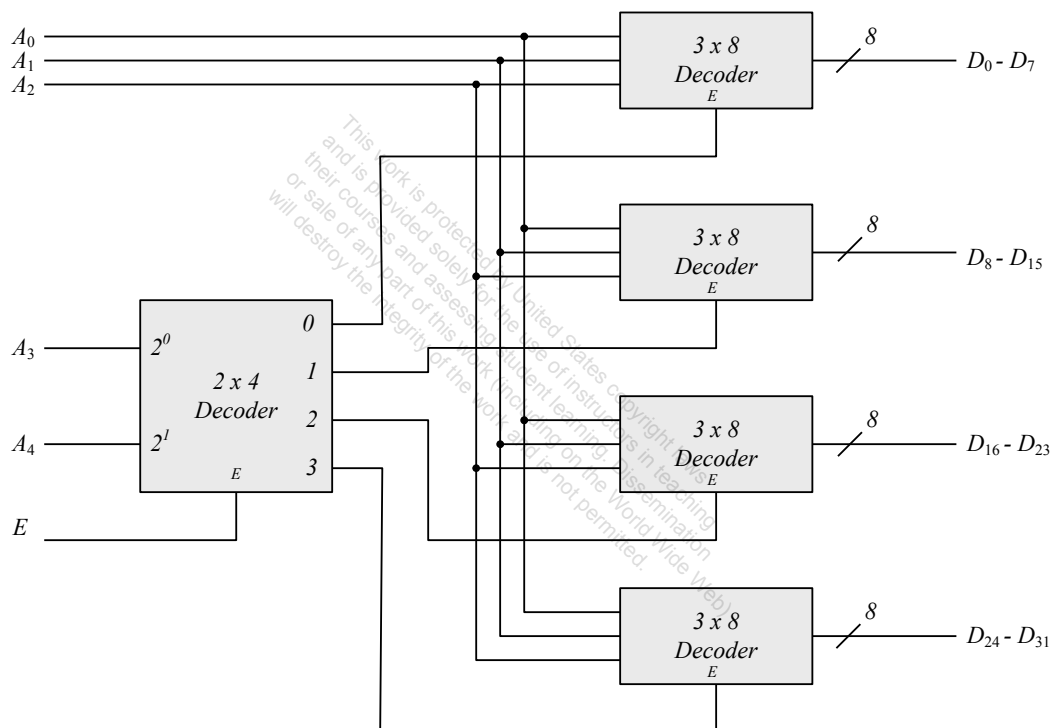
Inputs:  $A, B, C, D$

$$\begin{aligned} D_0 &= A'B'C'D' \\ D_1 &= A'B'C'D \\ D_2 &= B'CD' \\ D_3 &= B'CD \\ D_4 &= BC'D' \end{aligned}$$

$$\begin{aligned} \text{Outputs: } D_0, D_1, \dots, D_9 \\ D_5 &= BC'D \\ D_6 &= BCD' \\ D_7 &= BCD \\ D_8 &= AD' \\ D_9 &= AD \end{aligned}$$

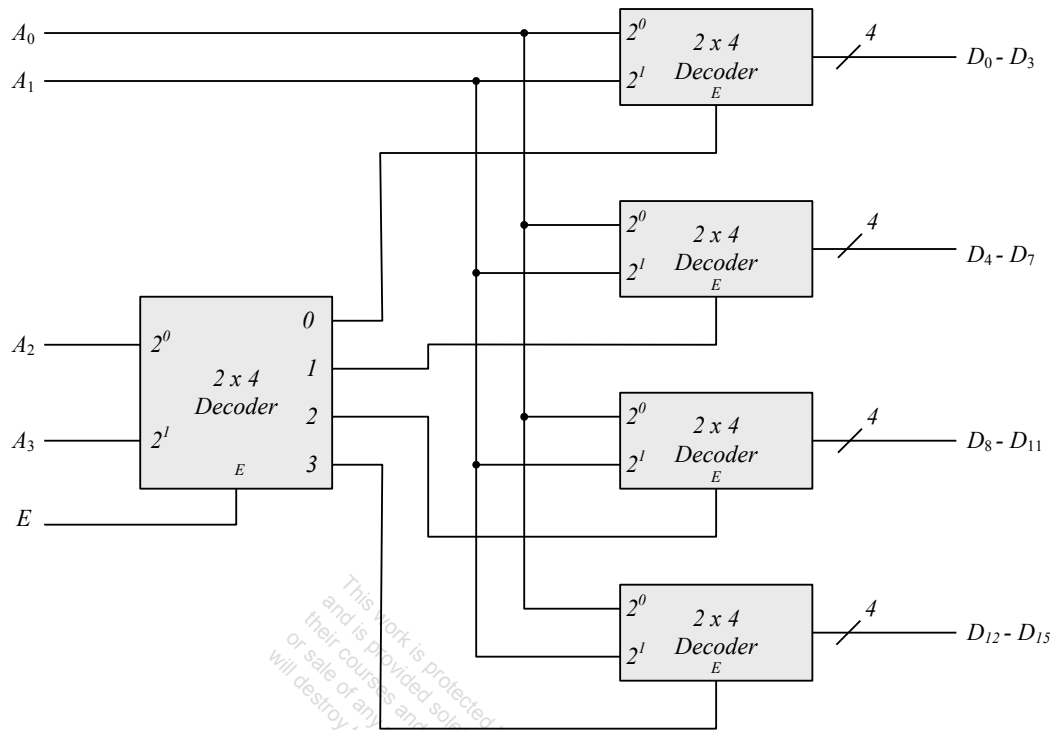
		$CD$			
		00	01	11	10
$AB$	00	$m_0$ $D_0$	$m_1$ $D_1$	$m_3$ $D_3$	$m_2$ $D_2$
	01	$m_4$ $D_4$	$m_5$ $D_5$	$m_7$ $D_7$	$m_6$ $D_6$
	11	$m_{12}$ x	$m_{13}$ x	$m_{15}$ x	$m_{14}$ x
	10	$m_8$ $D_8$	$m_9$ $D_9$	$m_{11}$ x	$m_{10}$ x

## 4.25



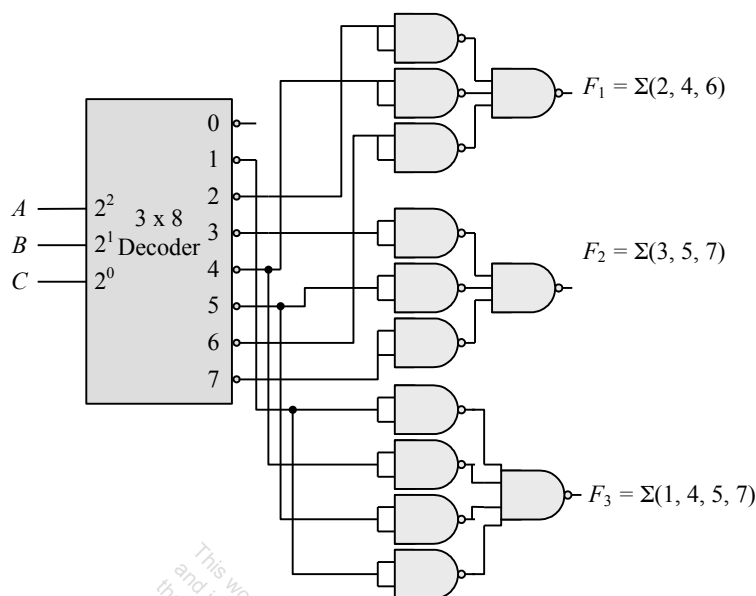


#### 4.26



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

4.27

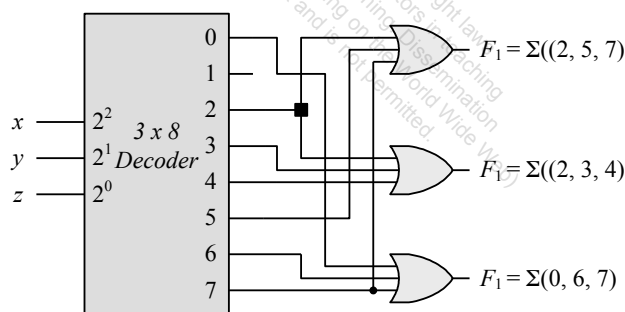


4.28 (a)

$$F_1 = x(y + y')z + x'yz' = xyx + xy'z + x'yz' = \Sigma(2, 5, 7)$$

$$F_2 = xy'z' + x'y = xy'z' + x'yz + x'yz' = \Sigma(2, 3, 4)$$

$$F_3 = x'y'z' + xy(z + z') = x'y'z' + xyz + xyz' = \Sigma(0, 6, 7)$$

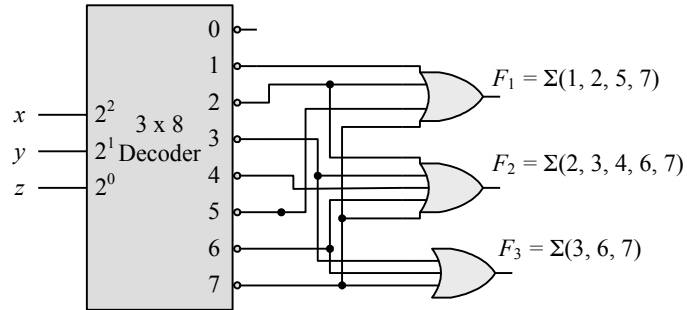


(b)

$$F_1 = (y' + x)z = xy' + xz = xy'z + x'y'z + xyz + xy'z = \Sigma(1, 2, 5, 7)$$

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

$$F_3 = (x + y)z = xy'z + xyz + x'yz + xyz = \Sigma(3, 6, 7)$$



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

# 4.29

Inputs					Outputs		
$D_3$	$D_2$	$D_1$	$D_0$		$x$	$y$	$V$
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

$D_3D_2$		$D_1D_0$			
		00	01	11	10
$D_3$	00	$m_0$	$m_1$	$m_3$	$m_2$
	01	$m_4$	$m_5$	$m_7$	$m_6$
	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
	10	$m_8$	$m_9$	$m_{11}$	$m_{10}$

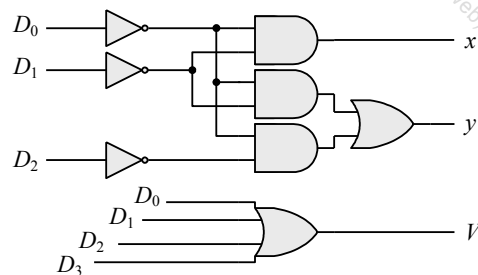
$$V = D_0 + D_1 + D_2 + D_3$$

$D_3D_2$		$D_1D_0$			
		00	01	11	10
$D_3$	00	$m_0$	$m_1$	$m_3$	$m_2$
	01	$m_4$	$m_5$	$m_7$	$m_6$
	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
	10	$m_8$	$m_9$	$m_{11}$	$m_{10}$

$$x = D_1'D_0'$$

$D_3D_2$		$D_1D_0$			
		00	01	11	10
$D_3$	00	$m_0$	$m_1$	$m_3$	$m_2$
	01	$m_4$	$m_5$	$m_7$	$m_6$
	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
	10	$m_8$	$m_9$	$m_{11}$	$m_{10}$

$$y = D_0'D_2' + D_1D_0'$$

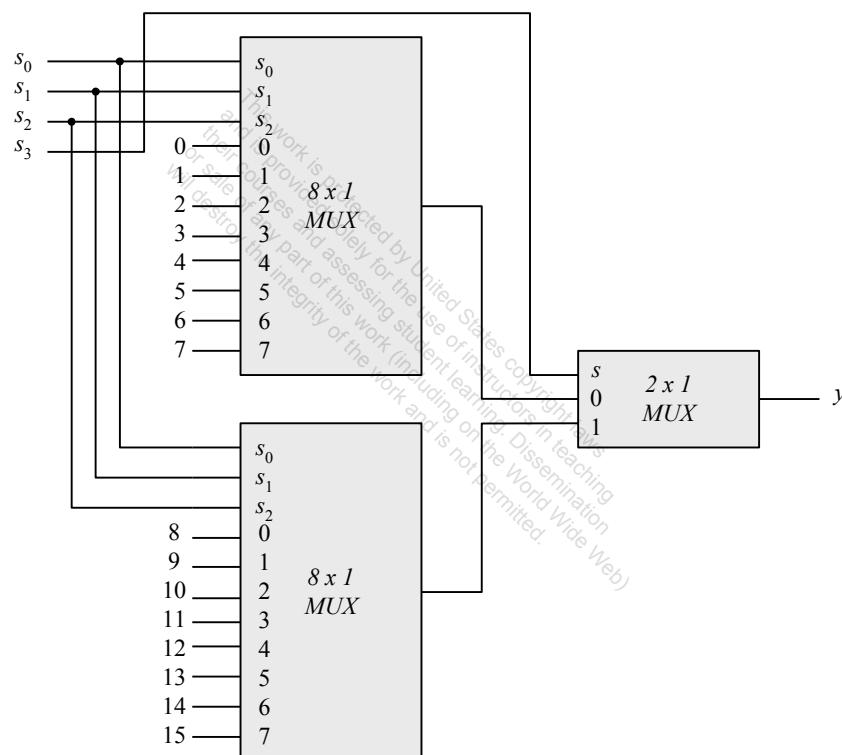


### 4.30

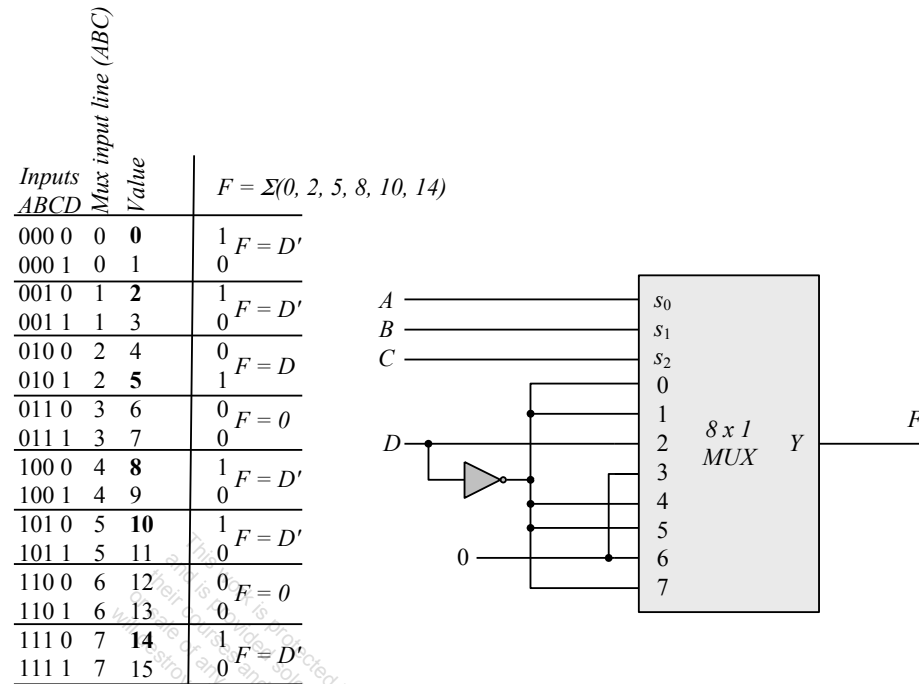
Inputs									Outputs			
$D_0$	$D_1$	$D_2$	$D_3$	$D_4$	$D_5$	$D_6$	$D_7$		$x$	$y$	$z$	$V$
0	0	0	0	0	0	0	0		x	x	x	0
1	0	0	0	0	0	0	0		0	0	0	1
x	1	0	0	0	0	0	0		0	0	1	1
x	x	1	0	0	0	0	0		0	1	0	1
x	x	x	1	0	0	0	0		0	1	1	1
x	x	x	x	1	0	0	0		1	0	0	1
x	x	x	x	x	1	0	0		1	0	1	1
x	x	x	x	x	x	1	0		1	0	0	1
x	x	x	x	x	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)  $F = \Sigma(0, 2, 5, 8, 10, 14)$



(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

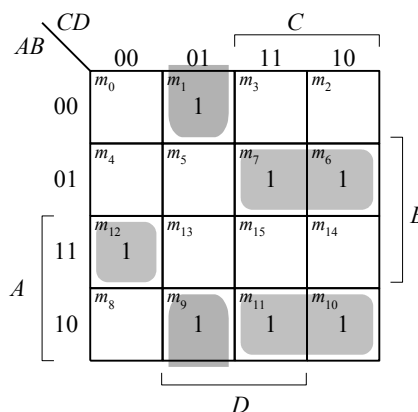
### 4.33

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

S	$I_0$	$I_1$	$I_2$	$I_3$	C	$I_0$	$I_1$	$I_2$	$I_3$
$x'$	0	1	2	3	$x'$	0	1	2	3
$x$	4	5	6	7	$x$	4	5	6	7
	$x$	$x'$	$x'$	$x$		$x$	$x'$	$x'$	$x$

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

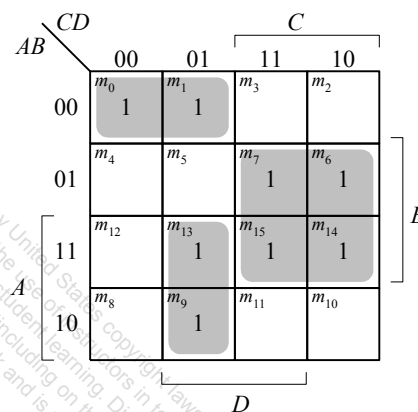


Other minterms = 0  
since  $I_1 = I_2 = I_7 = 0$

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

(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

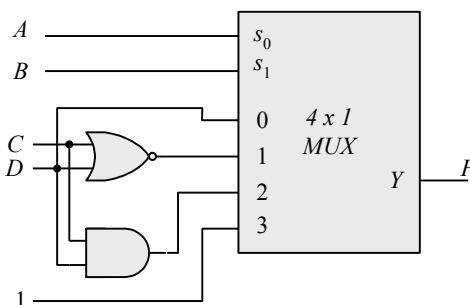


Other minterms = 0  
since  $I_1 = I_2 = 0$

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

4.35 (a)

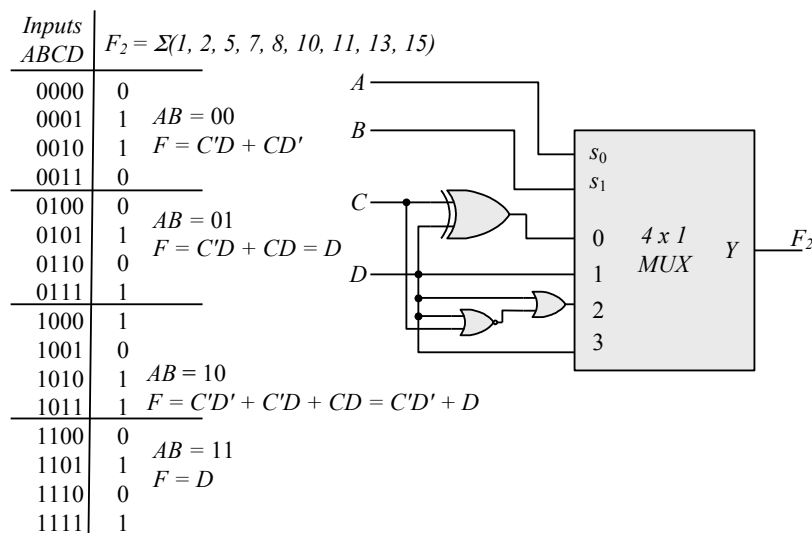
Inputs ABCD	F
0000	0
0001	1 $AB = 00$
0010	0 $F = D$
0011	1
0100	1 $AB = 01$
0101	0 $F = C'D'$
0110	0 $= (C + D)'$
0111	0
1000	0
1001	0 $AB = 10$
1010	0 $F = CD$
1011	1
1100	1
1101	1 $AB = 11$
1110	1 $F = 1$
1111	1





This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

(b)  $F = \Sigma(1, 2, 5, 7, 8, 10, 11, 13, 15)$



#### 4.36 (a) (Verilog/SystemVerilog)

**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)

#### VHDL

```
entity priority_encoder_gates is
  port ( x, y, V: out Std_Logic; D0, D1, D2, D3 : in Std_Logic);
end priority_encoder_gates;

architecture Gates of Priority_Encoder_Gates is
  component and2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
  component or2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
  component not_gate port (y: out Std_Logic; xin: in Std_Logic);
  signal w1, D2_not: Std_Logic;
begin
  G1: not_gate port map (D2_not => y, D2 => xin);
  G2: or2_gate port map (x => y, D2 => xin1, D3 => xin2);
  G3: or3_gate port map (V => y, D0 => xin1, D1 => xin2, x => xin3);
  G4: and2_gate port map (w1 => y, D2_not => xin1, D1 => xin2);
  G5: or2_gate port map (y => y, D3=> xin1, w1 => xin2);
end Gates;
```

```
entity and2_gate is
  port (y: out Std_Logic; xin1, xin2: in Std_Logic);
```

```
end and2_gate;
```

```
architecture Behavioral of and2_gate is
begin
    y <= xin1 and xin2;
end Behavioral;
```

```
entity or2_gate is
    port (y: out Std_Logic; xin1, xin2: in Std_Logic);
end or2_gate;
```

```
architecture Behavioral of or2_gate is
begin
    y <= xin1 or xin2;
end Behavioral;
```

```
entity or3_gate is
    port (y: out Std_Logic; xin1, xin2, xin3: in Std_Logic);
end or3_gate;
```

```
architecture Behavioral of or3_gate is
begin
    y <= xin1 or xin2 or xin3;
end Behavioral;
```

#### 4.37 (a) (Verilog/SystemVerilog)

```
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;

```

```

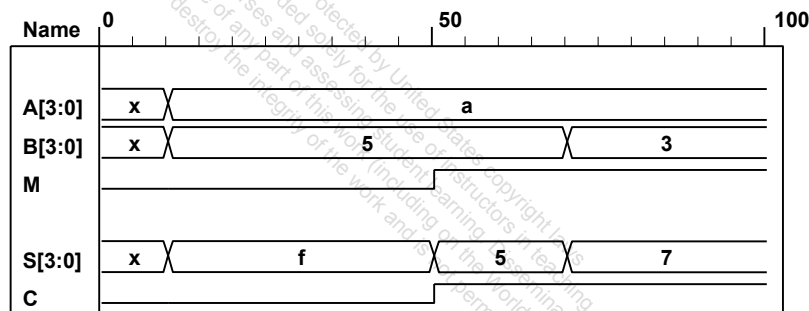
    Add_Sub_4_bit M0 (S, C, A, B, M);

```

```

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

```



(b) (VHDL)

```

entity Add_Sub_4_bit is
    port ( S: out Std_Logic_Vector (3 downto 0); C: out Std_Logic;
          A, B: in Std_Logic_Vector (3 downto 0); M: in Std_Logic);
end Add_Sub_4_bit;

architecture Structural of Add_Sub_4_bit is
    signal B_xor_M: Std_Logic_Vector (3 downto 0);
    signal C1, C2, C3, C4: Std_Logic;
    component xor2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
    component full_adder port (sum, cout: out Std_Logic; a, b, cin: in Std_Logic);
begin
    C <= C4
    G1: xor2_gate port map (B_xor_M(0) => y, B(0) => xin1, M => xin2);
    G2: xor2_gate port map (B_xor_M(1) => y, B(1) => xin1, M => xin2);
    G3: xor2_gate port map (B_xor_M(2) => y, B(2) => xin1, M => xin2);
    G4: xor2_gate port map (B_xor_M(3) => y, B(3) => xin1, M => xin2);
    FA0: full_adder port map (S(0) => sum, C1 => cout, A(0) => a, B_xor_M(0) => b, M => cin);
    FA1: full_adder port map (S(1) => sum, C2 => cout, A(1) => a, B_xor_M(1) => b, C1 => cin);
    FA2: full_adder port map (S(2) => sum, C3 => cout, A(2) => a, B_xor_M(2) => b, C2 => cin);

```

```
FA3: full_adder port map (S(3) => sum, C4 => cout, A(3) => a, B_xor_M(3) => b, C3 => cin);
end Structural;
```

```
entity full_adder is
  port (s, cout: out Std_Logic; a, b, cin: in Std_Logic);
end full_adder;
```

```
architecture Structural of full_adder is
  signal s1, c1, c2;
  component half_adder port (s, c: out Std_logic; a, b: in Std_Logic);
  component or2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
begin
  HA1: half_adder port map (s1 => s, c1 => c, a => a, b => b);
  HA2: half_adder port map (s1 => s, c2 => c, s1=> a, cin => b);
  G1: or2_gate port map (c => y, c2 => xin1, c1 => xin2);
end Structural
```

```
entity half_adder is
  port (s, c: out Std_Logic; a, b: in Std_Logic);
end half_adder;
```

```
architecture Operators of half_adder is
begin
```

```
  s <= x xor y;
  c <= x and y;
end Operators;
```

**Test Bench:**

```
entity t_Add_Sub_4_bit is
  port ();
end Add_Sub_4_bit;
```

```
architecture Behavioral of t_Add_Sub_4_bit is
```

```
  signal t_S: Std_Logic_Vector (3 downto 0), t_C: Std_Logic;
  signal t_A, t_B: Std_Logic_Vector (3 downto 0), t_M: Std_Logic;
  component Add_Sub_4_bit is
    port ( S: out Std_Logic_Vector (3 downto 0); C: out Std_Logic;
          A, B: in Std_Logic_Vector (3 downto 0); M: in Std_Logic);
```

```
begin
```

```
  G1: Add_Sub_4_bit port map(t_S => S, t_C => C, t_A => A, t_B => B, t_M => M);
```

```
  t_M <= '0'      after 10 ns;
  t_A <= '1010'   after 20 ns
  t_B <= '0101'   after 10 ns;
  t_M <= '1'      after 50 ns;
  t_B <= '0011'   after 70 ns.
end Behavioral;
```

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

#### 4.38 (a) (Verilog)

```
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'h0;
```

```
    #90 B = 4'hA;
```

```
    #100 B = 4'hF;
```

```
    #110 enable_bar = 1;
```

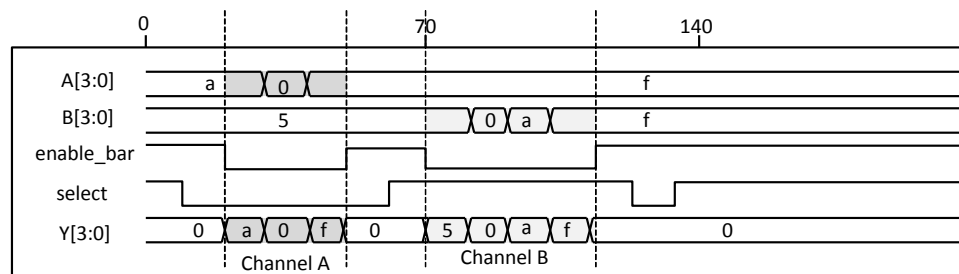
```
    #120 select = 0;
```

```
    #130 select = 1;
```

```
    #140 enable_bar = 1;
```

```
join
```

```
endmodule
```



#### (b) (VHDL)

```
entity quad_2x1_mux is
```

```
    port (Y: out Std_Logic_Vector (3 downto 0); A, B: in Std_Logic_Vector (3 down to 0);
```

```
          enable_bar, select: in Std_Logic_Vector);
```

```
end quad_2x1_mux;
```

```
architecture DataFlow of quad_2x1_mux is
begin
    Y <= '0000' when enable_bar = '1';
        else B when (not enable_bar) and (select);
        else A when (not enable_bar) and (not select);
end DataFlow;
```

```
entity t_quad_2x1_mux is
    port ();
end t_quad_2x1_mux;
```

```
architecture testbench of t_quad_2x1_mux is
begin
    enable_bar <= '1' after 0 ns;
    select <= '1' after 0 ns;
    A <= '1010' after 0 ns;
    B <= '0101' after 0 ns;

    select <= '0' after 10 ns; -- channel A
    enable_bar <= '0' after 20 ns;
    A <= '0000' after 30 ns;
    A <= '1111' after 40 ns;
    enable_bar <= '1' after 50 ns;
    select <= '1' after 60 ns; == Channel B
    enable_bar <= '0' after 70 ns;
    B <= '0000' after 80 ns;
    B <= '1010' after 90 ns;
    B <= '1111' after 100 ns;
    enable_bar <= '1' after 110 ns;
    select <= '0' after 120 ns;
    select <= '1' after 130 ns;
    enable_bar <= '1' after 140 ns;
end testbench.
```

#### 4.39 (a) 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**

#### **VHDL**

**entity** Compare is

**port** (A, B: **in** Std\_Logic\_vector 3 **downto** 0; Y: **out** Std\_Logic\_Vector 5 **downto** 0);  
**end** Compare;

**architecture** Behavioral of Compare is

**begin**

**process** (A, B) **begin**

**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; **end if**; // NE, GT, GE

**end** Behavioral;

### 4.40 Verilog

**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**

#### **VHDL**

**entity** Prob\_4\_40 is

**port** (sum\_diff: **out** Std\_Logic\_Vector (3 **downto** 0); carry\_borrow: **out** Std\_Logic;

A, B: **in** Std\_Logic\_Vector (3 **downto** 0); sel\_diff: **in** Std\_Logic);

**end** Prob\_4\_40;

**architecture** DataFlow of Prob\_4\_40 is

**begin**

carry\_borrow & sum\_diff <= A – B **when** select\_diff = 1; **else** A+ B;

**end** DataFlow;

#### 4.41 (a) Verilog/SystemVerilog

```

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 ? ('0' & A) - ('0' & B) : ('0' & A) + ('0' & B);
endmodule

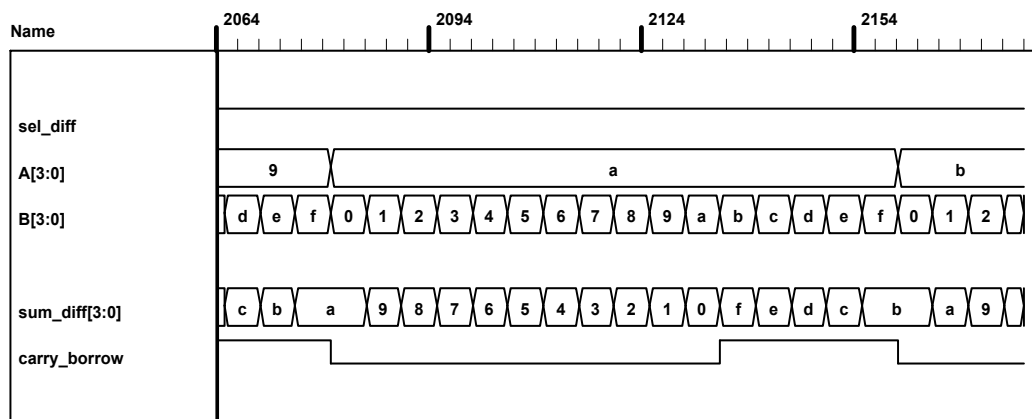
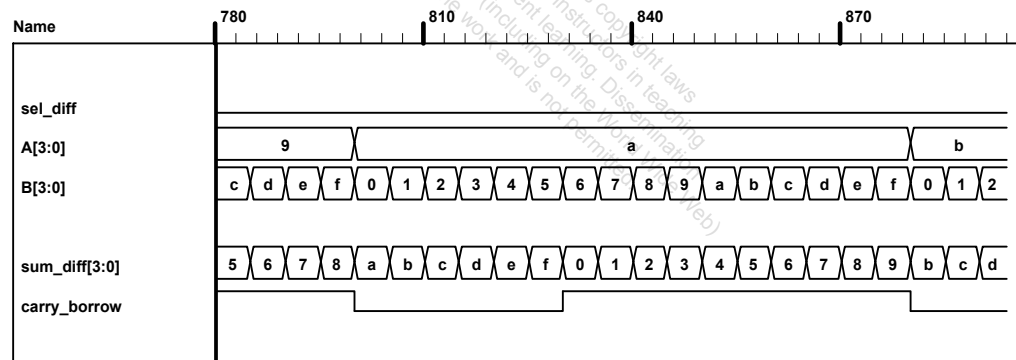
```

```

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

```



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

## (b) VHDL

```

entity Prob_4_41 is
  port (sum_diff: out Std_Logic_Vector (3 downto 0); carry_borrow: out Std_Logic;
        A, B: in Std_Logic_Vector (3 downto 0), sel_diff: in Std_Logic);
end Prob_4_41;

architecture Behavioral of Prob_4_41 is
  process (A, B, sel_diff)
  begin
    (carry_borrow & sum_diff) <= ('0' & A) + ('0' & B) when sel_diff; else ('0' & A) - ('0' & B);
  end process;
end Behavioral;

```

## 4.42 (a) Verilog/SystemVerilog

```

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

```

### VHDL

```

entity Xs3_Converter is
  port (A, B, C, D: in Std_Logic; w, x, y, z: out Std_Logic);
end Xs3_Gates;

architecture Gates of Xs3_Converter is
  signal B_bar, C_or_D_bar, Cd, C_or_D: Std_Logic;
  component or2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
  component and2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
  component not_gate port (y: out Std_Logic; xin: in Std_Logic);
begin
  G1: or2_gate port map (C_or_D => y, C => xin1, D => xin2);
  G2: not_gate port map (C_or_D_bar => y, C_or_D => xin);
  G3: not_gate port map (B_bar => y, B => xin);
  G4: and2_gate port map (Cd => y, C => xin1, D => xin2);
  G5: not_gate port map (z => y, D => xin);
  G6: or2_gate port map (y => y, CD => xin1, C_or_D_bar => xin2);
  G7: and2_gate port map (w1 => y, C_or_D_bar => xin1, B => xin2);
  G8: and2_gate port map (w2 => y, B_bar => xin1, C_or_D => xin2);
  G9: and2_gate port map (w3 => y, C_or_D => xin1, B => xin2);
  G10: or2_gate port map (x => y, w1 => xin1, w2 => xin2);
  G11: or2_gate port map (x => y, w3 => xin1, A => xin2);
end Gates

```

See declarations of and2\_gate, or2\_gate and not\_gate in previous problems (above).

(b) **Verilog**

```
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
```

**VHDL**

```
architecture Behavioral of Xs3_Converter is
begin
    process (A, B, C, D) begin
        {w, x, y, z} <= {A, B, C, D} + 4'b0011;
    end process;
```

(c) **Verilog/SystemVerilog**

```
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
```

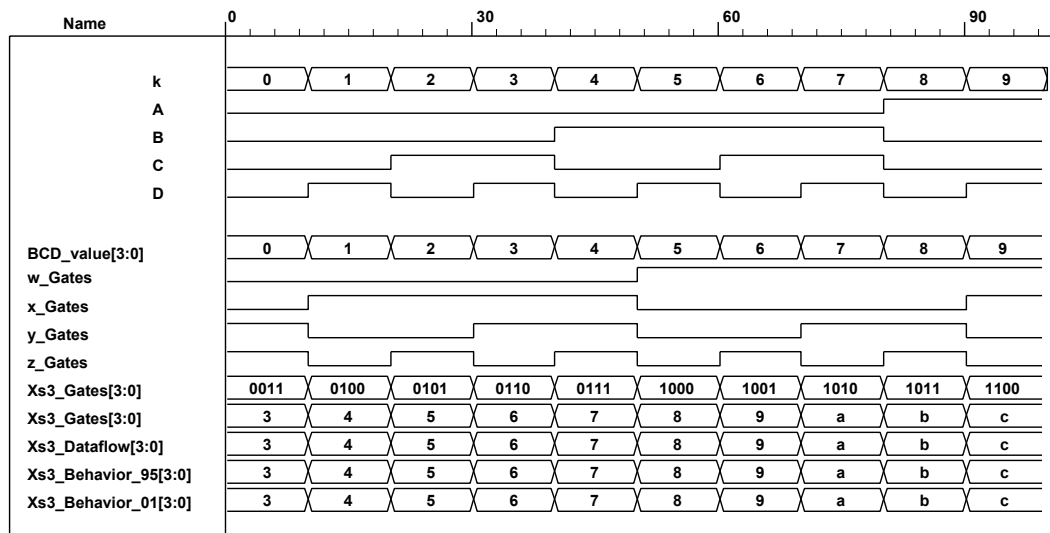
**VHDL**

**Verilog Testbench**

```
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
```



### (b)(VHDL)

```
entity Xs3_Behavior_vhdl is
    port (A, B, C, D: in std_logic; w, x, y, z: out std_logic);
end Xs3_Behavior_vhdl;

architecture Behavioral of Xs3_Behavior_vhdl is
begin
    w & x & y & z <= A & B & C & D + '0011';
end Behavioral;
```

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

### VHDL

```
entity Prob4_43_vhdl is
    port (A, B: in Std_Logic_Vector (1 downto 0); S, E: in Std_Logic;
          Q: out Std_Logic_Vector (1 downto 0);
end Prob4_43_vhdl;

architecture Behavioral of Prob4_43_vhdl is
begin
    process (A, B, S, E) begin
        Q <= A when (S = '1' and E = '1') ;
        else '0' when (S = '0' and E = '1');
        else 'z';
    end process
end Behavioral;
```

## 4.44 (a) (Verilog/SystemVerilog) (Note: Data paths are eight bits)

```
module ALU (output reg [7: 0] y, input [15: 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;
```

```

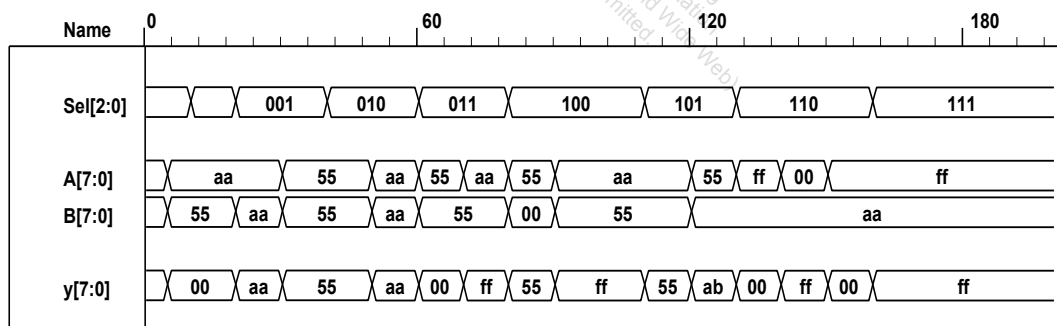
3'b011: y = A ^ B;
3'b100: y = A + B;
3'b101: y = A - B;
3'b110: y = ~A;
3'b111: y = 8'hFF;
endcase
end
endmodule

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 9<sup>th</sup> 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.

## (b) (VHDL)

entity ALU is

```

    port (y: out Std_Logic_Vector (7 downto 0); A, B: in Std_Logic_Vector (7 downto 0),
          Sel: in Std_Logic_Vector (2 downto 0));
end ALU;
architecture Behavioral of ALU is
begin
    process (A, B, Sel) begin
        y <= 0;
        case (Sel) is
            when '000' => y <= 00000000;
            when '001' => y <= A and B;
            when '010' => y <= A or B;
            when '011' => y <= A xor B;
            when '100' => y <= A + B;
            when '101' => y <= A - B;
            when '110' => y <= not A;
            when '111' => y <= '11111111';
        endcase
    end
end Behavioral

```

#### 4.45

```

(a) (Verilog)
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'bxx100: {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

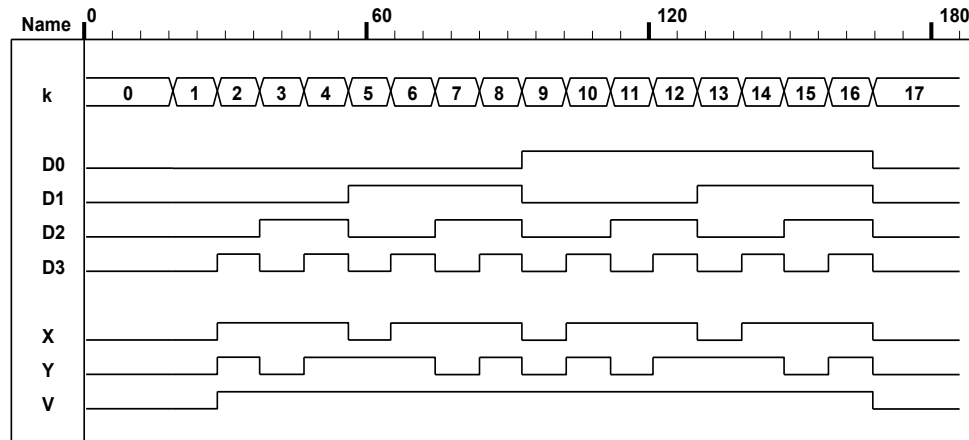
module t_priority_encoder_beh (); // V2001
    wire X, Y, V;
    reg D0, D1, D2, D3;
    integer k;

    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

```





### VHDL

```
entity priority_encoder is
    port (X, Y, V: out Std_Logic; D0, D1, D2, D3: in Std_Logic);
end priority_encoder;
```

```
architecture Behavioral of Priority_Encoder is
begin
```

```
    process @ (D0, D1, D2, D3) begin
```

```
        X <= 0;
```

```
        Y <= 0;
```

```
        V <= 0;
```

```
        case ({D0, D1, D2, D3})
```

```
            when '0000' => {X, Y, V} <= 'xx0'; -- Invalid
```

```
            when '1000' => {X, Y, V} <= '001';
```

```
            when '100'  => {X, Y, V} = '011';
```

```
            when '10'   => {X, Y, V} = '101';
```

```
            when '1'    => {X, Y, V} = '111';
```

```
            when others => {X, Y, V} = '000';
```

```
        endcase
```

```
    end
```

```
end architecture;
```

**4.46 (a)**

$$F = \Sigma(0, 2, 5, 7, 11, 14)$$

**Verilog**

```
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
```

**VHDL**

```
entity Prob_4_46a is
port (F: out Std_Logic; A, B, C, D: in Std_Logic);
end Prob_4_46a;
```

```
architecture Boolean of Prob_4_46a is
begin
```

```
    F <= (not A & not B & not C & not D)
        or (not A & not B & C & not D)
        or (not A & B & not C & D)
        or (not A & B & C & D)
        or (A & not B & C & D)
        or (A & B & C & not D);
```

```
end Boolean;
```

**(b) From prob 4.32:**

$$F = \Pi(3, 8, 12) = (A' + B' + C + D)(A + B' + C' + D')(A + B + C' + D')$$

$$F' = ABC'D' + A'BCD + A'B'CD = \Sigma(12, 7, 3)$$

$$F = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 10, 11, 13, 14, 15)$$

**Verilog**

```
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
```

---

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

## VHDL

**entity** Prob\_4\_46b is

**port** (F: **out** Std\_Logic; A, B, C, D: **in** Std\_Logic);

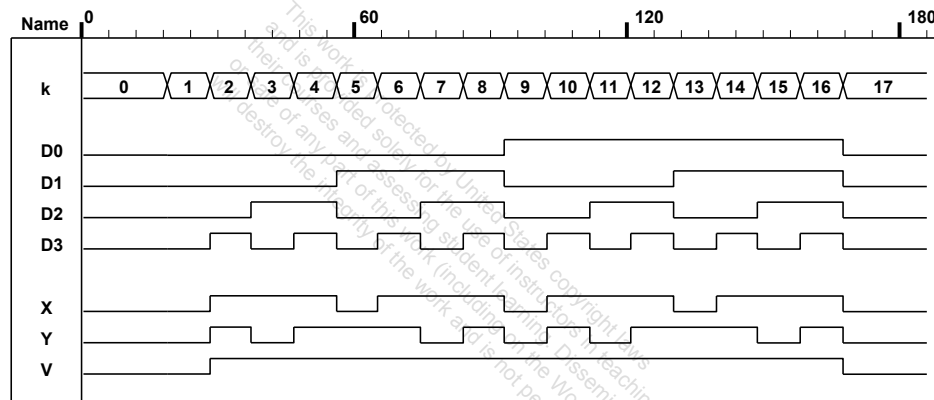
**end** Prob\_4\_46b;

**architecture** Boolean **of** Prob\_4\_46b is

**begin**

```
F <= (not A & not B & not C & not D) or (~A&~B&~C&D)
      or (not A & not B & C & not D) or (not A & B & not C & not D)
      or (not A & B & not C & D) or (not A & B & C & not D)
      or (A & not B & not C & not D) or (A & not B & not C & D)
      or (A & not B & C & not D) or (A & not B & C & D)
      or (A & B & not C & D) or (A & B & C & not D)
      or (A & B & C & D);
```

**endmodule**



#### 4.47

#### Verilog

```
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;
```

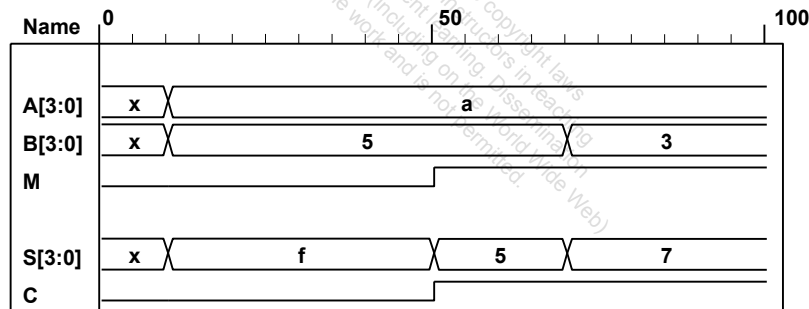
```
#10 B = 4'h5;
```

```
#50 M = 1;
```

```
#70 B = 4'h3;
```

```
join
```

```
endmodule
```



#### VHDL

```
entity Add_Sub_4_bit is
```

```
    port (
```

```
        S: out Std_Logic_Vector (3 downto 0);
```

```
        C, V: out Std_Logic;
```

```
        A, B: in Std_Logic_Vector (3 downto 0);
```

```
        M: in Std_Logic;
    );
```

```
end Add_Sub_4_bit_Dataflow;
```

```
architecture Boolean of Add_Sub_4_bit is
```

```
    signal C3: Std_Logic;
```

```
(C3, & S(2: 0)) <= A(2: 0) + ((M & M & M) xor B(2: 0)) + M;
```

```
(C & S(3)) <= A(3) + M xor B(3) + C3;
```

```
V <= C xor C3;
```

**end Boolean;**

#### 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 | 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

#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**

**VHDL**

**entity** ALU\_3state is

**port** (y\_tri: out Std\_Logic\_Vector (7 downto 0); A, B: in Std\_Logic\_Vector (7 downto 0);

Sel: in Std\_Logic\_Vector (2 downto 0); En: in Std\_Logic);

**end** ALU\_3state;

**architecture Behavioral of ALU\_3state is**

```
begin
    y_tri <= when En = '1'; else 'zzzzzzzz';

    process (A, B, Sel) begin
        y = 0;
        case (Sel)
            when '000' =>    y <= '00000000';
            when '001' =>    y <= A and B;
            when '010' =>    y <= A or B;
            when '011' =>    y <= A xor B;
            when '100' =>    y <= A + B;
            when '101' =>    y <= A - B;
            when '110' =>    y = not A;
            when '111' =>    y = '11111111';
        endcase
    end process;
end Behavioral;
```

#### 4.49// See Problem 4.1

### Verilog

**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**

### VHDL

**entity** Problem\_4\_49 **is**  
    **port** (F1, F2: **out** Std\_Logic; A, B, C, D: **in** Std\_Logic);  
**end** Prob\_4\_49;

**architecture** Gates **of** Prob\_4\_49 **is**

**signal** A\_bar: Std\_Logic;  
    **signal** B\_bar: Std\_Logic;  
    **signal** T1, T2, T3, T4: Std\_Logic;  
    **component** and2\_gate **port** (y: **out** Std\_Logic; xin1, xin2: **in** Std\_Logic);  
    **component** or2\_gate **port** (y: **out** Std\_Logic; xin1, xin2: **in** Std\_Logic);  
    **component** xor2\_gate **port** \*(y: **out** Std\_Logic; xin1, xin2: **in** Std\_Logic);  
**begin**  
    A\_bar <= not A;  
    B\_bar <= not B;

```

G1: and2_gate port_map (y => T1, xin1 => B_bar, xin2=> C);
G2 and2_gate map (y => T2, xin1 => A_bar, xin2 => B);
G3: or2_gate map (y => T3, xin1 => A, xin2 => T1);
G4: xor2_gate map (y => T4, xin1 => T2, xin2 => D);
G5: or2_gate map (y => F1, xin1 => T3, xin2 => T4);
G6: or2_gate map (y => F2, xin1 => T2, xin2 => D);
end Gates;

architecture Boolean_1 of Prob_4_49 is
    signal A_bar, B_bar, T1, T2, T3, T4: Std_Logic;
    signal T1, T2, T3, T4: Std_Logic;
begin
    A_bar <= not A;
    B_bar <= not B;
    T1 <= B_bar and C;
    T2 <= A_bar and B;
    T3 <= A or T1;
    T4 <= T2 xor D;
    F1 <= T3 or T4;
    F2 <= T2 or D;
end Boolean

architecture Boolean_2 of Problem_4_49 is

    F1 <= A or (not B and C) or (B and (not D)) or (not B and D);
    F2 <= ((not A) and B) or D;
end Boolean_2;

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

// 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)
    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
    endcase

```



```

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

VHDL
entity Prob_4_50a_vhdl is
port (code_BCD: out std_logic_vector (3 downto 0); Code_84_md_m1: in std_logic_vector (3 downto 0));
end Prob_4_50a_vhdl;

architecture Behavioral of Prob_4_50a is
begin
    process (Code_84_m2_m1) begin
        case (Code_84_m2_m1)
            when '0000' => Code_BCD <= '0000';    // 0
            when '0111' => Code_BCD <= '0001';    // 1
            when '0110' => Code_BCD <= '0010';    // 2
            when '0101' => Code_BCD <= '0011';    // 3
            when '0100' => Code_BCD <= '0100';    // 4
            when '1011' => Code_BCD <= '0101';    // 5
            when '1010' => Code_BCD <= '0110';    // 6
            when '1001' => Code_BCD <= '0111';    // 7
            when '1000' => Code_BCD <= '1000';    // 8
            when '1111' => Code_BCD <= '1001';    // 9

            when '0001' => Code_BCD <= '1010';    // 10
            when '0010' => Code_BCD <= '1011';    // 11
            when '0011' => Code_BCD <= '1100';    // 12
            when '1100' => Code_BCD <= '1101';    // 13
            when '1101' => Code_BCD <= '1110';    // 14
            when '1110' => Code_BCD <= '1111';    // 15
        end case;
    end process;
end Behavioral;

```

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

### Verilog

```

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

```

### VHDL

architecture Behavioral of Prob\_4\_50b is

```

begin
    process (Code_84_m2_m1)
        begin
            case (Code_84_m2_m1)
                when '0000' => Code_Gray <= '0000'; // 0
                when '0111' => Code_Gray <= '0001'; // 1
                when '0110' => Code_Gray <= '0011'; // 2
                when '0101' => Code_Gray <= '0010'; // 3
                when '0100' => Code_Gray <= '0110'; // 4
                when '1011' => Code_Gray <= '0111'; // 5
                when '1010' => Code_Gray <= '0101'; // 6
                when '1001' => Code_Gray <= '0100'; // 7
                when '1000' => Code_Gray <= '1100'; // 8
                when '1111' => Code_Gray <= '1101'; // 9

                when '0001' => Code_Gray <= '1111'; // 10
                when '0010' => Code_Gray <= '1110'; // 11
                when '0011' => Code_Gray <= '1010'; // 12
                when '1100' => Code_Gray <= '1011'; // 13
                when '1101' => Code_Gray <= '1001'; // 14
            endcase
        end
    endprocess
end

```

```

when '1110'=> Code_Gray <= '1000';      // 15
when OTHERS=> Code_Gray <= BLANK;
end case;
end process ;

```

**4.51** Assume 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

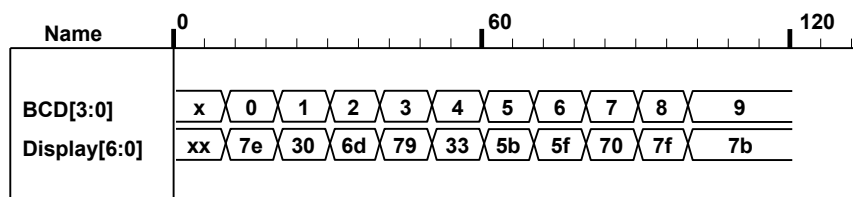
initial #120 $finish;
initial fork
    #10 BCD = 0;
    #20 BCD = 1;

```

```
#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
```



## VHDL

```
entity Seven_Seg_Display is
    port (Display: out Std_Logic_Vector (6 downto 0);
          BCD: in Std_Logic_Vector (3 downto 0);
    end Seven_Seg_Display;

architecture Behavioral of Seven_Seg_Display is
    // abc_defgnd Seven_Seg_Display;
    constant BLANK: Std_Logic_Vector := '000_0000';
    constant ZERO: Std_Logic_Vector := '111_1110'; -- hex 7e
    constant ONE: Std_Logic_Vector := '011_0000'; -- hex 30
    constant TWO: Std_Logic_Vector := '110_1101'; -- hex 6d
    constant THREE: Std_Logic_Vector := '111_1001'; -- hex 79
    constant FOUR: Std_Logic_Vector := '011_0011'; -- hex 33
    constant FIVE: Std_Logic_Vector := '101_1011'; -- hex 5b
    constant SIX: Std_Logic_Vector := '101_1111'; -- hex 5f
    constant SEVEN: Std_Logic_Vector := '111_0000'; -- hex 70
    constant EIGHT: Std_Logic_Vector := '111_1111'; -- hex 7f
    constant NINE: Std_Logic_Vector := '111_1011'; -- hex 7b
    begin
    process (BCD) begin
        case (BCD)
            when 0 => Display <= ZERO;
            when 1 => Display <= ONE;
            when 2 => Display <= TWO;
            when 3 => Display <= THREE;
            when 4 => Display <= FOUR;
            when 5 => Display <= FIVE;
            when 6 => Display <= SIX;
            when 7 => Display <= SEVEN;
            when 8 => Display <= EIGHT;
            when 9 => Display <= NINE;
            when others => Display <= BLANK;
        end case
    end process
end Behavioral
```

### 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 = 7;  
#90 BCD = 8;  
#100 BCD = 9;
```

**join**

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

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

4.52 (a) Incrementer for unsigned 4-bit numbers

**VERILOG**

```
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
```

**VHDL**

```
entity Problem_4_52a_Data_Flow is
```

```
    port (sum: out Std_Logic_Vector (3 downto 0); carry: in Std_Logic;
```

```
          A: in Std_Logic_Vector (3: 0));
```

```
end Problem_4_52;
```

```
architecture Data_Flow of Problem_4_52a is
```

```
begin
```

```
    (carry & sum) <= A + '0001';
```

```
endmodule
```

(b) Decrementer for unsigned 4-bit numbers

**Verilog**

```
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
```

Name	0																30																60																90
A[3:0]	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f																																	
diff[3:0]	f	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e																																	
borrow																																																	

**VHDL**

```
entity Problem_4_52b is
```

```
    port (diff: out Std_Logic_Vector (3 downto 0); borrow: out Std_Logic;
```

```
    A: in Std_Logic_Vector (3 downto 0);  
end Problem_42b;  
  
architecture Data_Flow of Problem_4_52b is  
begin  
    (borrow & diff) <= A - '0001';  
endmodule
```

This work is protected by United States copyright laws  
and is provided solely for the use of instructors in teaching  
their courses and assessing student learning. Dissemination  
or sale of any part of this work (including on the World Wide Web)  
will destroy the integrity of the work and is not permitted.



#### 4.53 // BCD Adder

##### Verilog

```

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

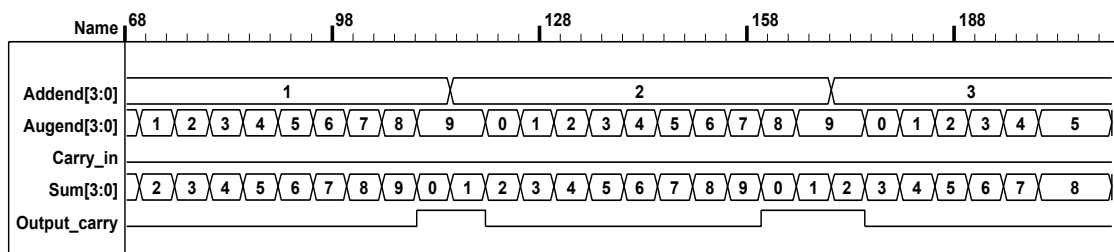
module Adder_4_bit (output carry, output [3:0] sum, input [3: 0] a, b, input c_in;
    assign {carry, sum} = a + b + c_in;
endmodule

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
    end
endmodule

```



##### VHDL

```

entity Problem_4_53_BCD_Adder is
    port(output_carry: out Std_Logic; Sum: out Std_Logic_Vector (3 downto 0);
    input [3: 0] Addend, Augend: in Std_Logic_Vector (3 downto 0);
end Problem_4_53_BCD_Adder

```

```

architecture Gates of Problem_4_53_BDC_Adder is
    signal [3: 0] Z_Addend: Std_Logic_Vector (3 downto 0);
    signal Carry_out: Std_Logic;
    signal C_out: Std_Logic;
    signal Z_sum: Std_Logic_Vector (3 downto 0);
    constant gnd: Std_Logic := '0';
    component and2_gate port (y: out Std_Logic; xin1, xin2: in Std_Logic);
    component or3_gate port (y: out Std_Logic; xin1, xin2, xin3: in Std_Logic);
    component Adder_4_bit port (c: out Std_Logic; s: out Std_Logic_Vector (3 downto 0);
        a, b: in Std_Logic_Vector (3 downto 0); cin: in Std_Logic);
begin
    Z_Addend <= ('0' & Output_carry & Output_carry & '0')
    G1: and2_gate port map (y => w1, xin1 => Z_sum(3), xin2 => Z_sum(2));
    G2: and2_gate port map (y => w2, xin1 => Z_sum(3), xin2 => Z_sum(1));
    G3: or3_gate port map (y => Output_carry, xin1 => Carry_out, xin2 => w1, xin3 => w2);

    C0: Adder_4_bit port map (c => Carry_out, s => Z_sum, a => Addend, b => Augend, cin => Carry_in);
    C1: Adder_4_bit (port map (c => C_out, s => Sum, a => Z_Addend, b => Z_sum, cin => gnd);
end gates;

entity Adder_4_bit is
    port (carry out: Std_Logic; sum: Std_Logic_Vector (3 downto 0);
        a, b: in Std_Logic_Vector (3 downto 0); input c_in: in Std_Logic);
end Adder_4_bit

architecture Behavioral of Adder_4_bit is
begin
    (carry & sum <= a + b + ('000' & c_in);
end Behavioral;

```

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

##### VHDL

```

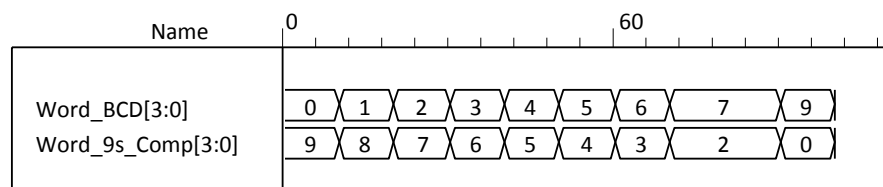
module Nines_Complementer (           // V2001
  output reg [3: 0] Word_9s_Comp,
  input      [3: 0] Word_BCD
);
  always @ (Word_BCD) begin
    Word_9s_Comp = 4'b0;
    case (Word_BCD)
      4'b0000: Word_9s_Comp = 4'b1001;    // 0 to 9
      4'b0001: Word_9s_Comp = 4'b1000;    // 1 to 8
      4'b0010: 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;
    #80 Word_BCD = 8;
    #90 Word_BCD = 9;
    #100 Word_BCD = 4'b1100;    // Confirm error detection
  join
endmodule

```



## **VHDL**

```
entity Nines_Complementer is
    port (Word_9s_Comp: out Std_Logic_Vector (3 downto 0); Word_BCD: in Std_Logic_Vector (3
downto 0));
end Nines_Complementer;
```

**architecture** Behavioral **of** Nines\_complementer **is**

**process** (Word\_BCD) **begin**

Word\_9s\_Comp <= '0000';

**case** (Word\_BCD)

**when** '0000' => Word\_9s\_Comp <= '1001'; // 0 to 9

**when** '0001' => Word\_9s\_Comp <= '1000';

**when** '0010' => Word\_9s\_Comp <= '0111'; // 2 to 7

**when** '0011' => Word\_9s\_Comp <= '0110'; // 3 to 6

**when** '0100' => Word\_9s\_Comp <= '0101'; // 4 to 5

**when** '0101' => Word\_9s\_Comp <= '0100'; // 5 to 4

**when** '0110' => Word\_9s\_Comp <= '0011'; // 6 to 3

**when** '0111' => Word\_9s\_Comp <= '0010'; // 7 to 2

**when** '1000' => Word\_9s\_Comp <= '0001'; // 8 to 1

**when** '1001' => Word\_9s\_Comp <= '0000'; // 9 to 0

**when** **others** => Word\_9s\_Comp <= '1111'; // Error detection

**end case**

**end** Behavioral

## **(b) 9s complement of Gray Code**

### **Verilog**

**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\_Gray)

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

```

## VHDL

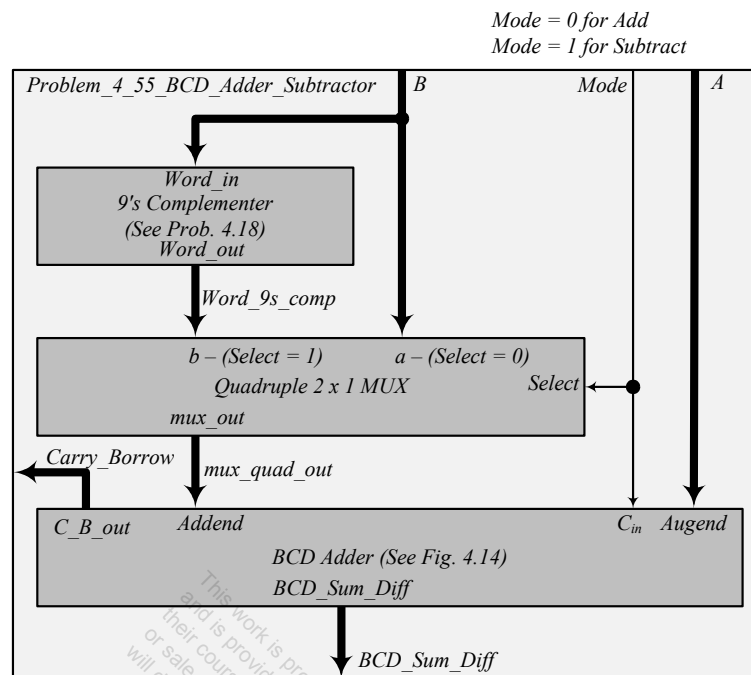
```

entity Nines_Complementer is
    port (Word_9s_Comp: out Std_Logic_Vector (3 downto 0);
          Word_Gray: in Std_Logic_Vector (3 downto 0);
end Nines_Complementer;

architecture Behavioral of Nines_Complementer is
begin
    process (Word_Gray) begin
        Word_9s_Comp <= '0000';
        case (Word_Gray)
            when '0000' => Word_9s_Comp <= '1101';    // 0 to 9
            when '0001' => Word_9s_Comp <= '1100';    // 1 to 8
            when '0010' => Word_9s_Comp <= '0100';    // 2 to 7
            when '0011' => Word_9s_Comp <= '0101';    // 3 to 6
            when '0100' => Word_9s_Comp <= '0111';    // 4 to 5
            when '0101' => Word_9s_Comp <= '0110';    // 5 to 4
            when '0110' => Word_9s_Comp <= '0010';    // 6 to 3
            when '0111' => Word_9s_Comp <= '0011';    // 7 to 2
            when '1000' => Word_9s_Comp <= '0001';    // 8 to 1
            when '1001' => Word_9s_Comp <= '0000';    // 9 to 0
            when others => Word_9s_Comp <= '1111';    // Error detection
        endcase
    end
endmodule

```

#### 4.55 From Problem 4.19:



#### Verilog

```
// 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_quad_out;
    Nines_Complementer M0 (Word_9s_Comp, B);
    Quad_2_x_1_mux M2 (mux_quad_out, Word_9s_Comp, B, Mode);
    BCD_Adder M1 (Carry_Borrow, BCD_Sum_Diff, mux_quad_out, A, Mode);
endmodule

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

```

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      C_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, C_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

```

## VHDL

```

// BCD Adder – Subtractor
entity Problem_4_55_BCD_Adder_Subtractor is

```

```

    port (BCD_Sum_Diff: out Std_Logic_Vector (3 downto 0);
          Carry_Borrow: out Std_Logic;
          B, A: in Std_Logic_Vector (3:0);
          Mode: in Std_Logic);
end Problem_4_55_BCD_Adder_Subtractor;

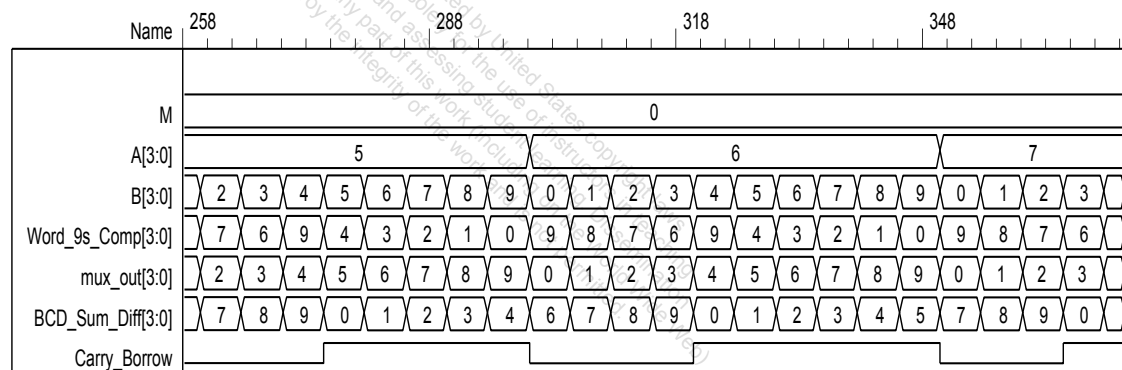
architecture Structural of Problem_4_55_BCD_Adder_Subtractor is
    signal Word_9s_comp, mux_quad_out: Std_Logic_Vector (3 downto 0);
    component Nines_Complementer port (Word_out: out Std_Logic_Vector (3 downto 0);
                                         Word_in: in Std_Logic_Vector);
    component Quad_2_x_1_mux port (mux_out: out Std_Logic_Vector (3 downto 0);
                                   B, A: in Std_Logic_Vector (3 downto 0);
                                   Mode: Std_Logic);
    component BCD_Adder port (C_B_out: out Std_Logic;
                              BCD_Sum_Diff: out Std_Logic_Vector (3 downto 0);
                              Addend, Augend: in Std_Logic_Vector (3 downto 0);
                              Cin: in Std_Logic);

begin

    M1: Nines_Complementer port map (Word_out => Word_9s_comp, Word_in => B);
    M2: Quad_2_x_1_mux port map (mux_out => mux_quad_out, b => Word_9s_comp,
                                a => B, Select => Mode);
    M3: BCD_Adder port map (C_B_out => Carry_Borrow, Addend => mux_quad_out,
                           Augend => A, cin => Mode,
                           BCD_Sum_Diff => BCD_Sum_Diff);

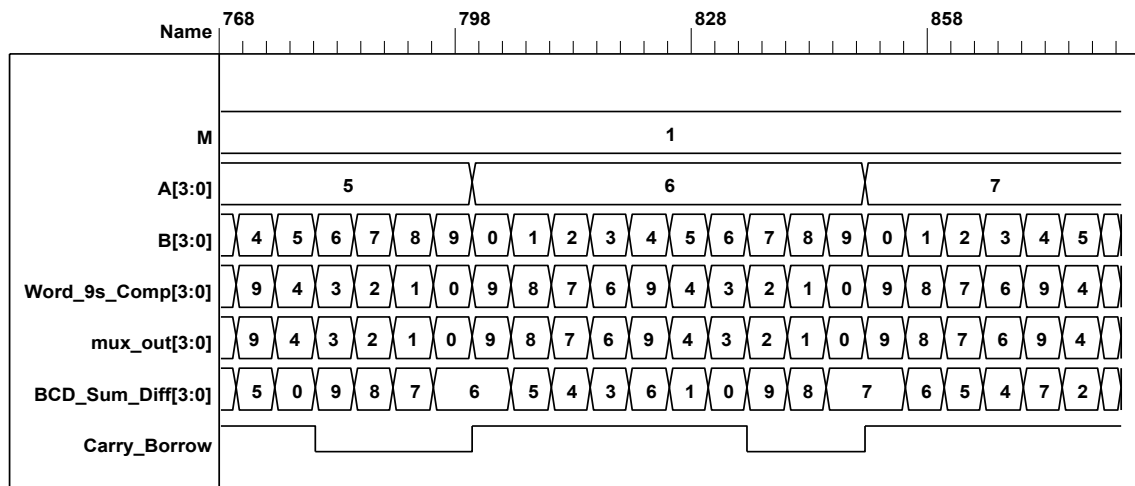
end Structural

```



Note: For subtraction, Carry\_Borrow = 1 indicates a positive result; Carry\_Borrow = 0 indicates a negative result.





#### 4.56

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

VHDL: match <= (A = B);

#### 4.57 Verilog

```

module Prob_4_57(
    input D0, D1, D2, D3,
    output reg x_in, y_in, Valid
);

always @ (D0, D1, D2, D3) begin // Note: positions numbered left to right
    case ({D0, D1, D2, D3})
        4'b0000: {x_out, y_out, Valid} = 3'bxx_0; // Invalid word
        4'b1xxx: {x_out, y_out, Valid} = 3'b00_1; // Bit 0, valid word
        4'b01xx: {x_out, y_out, Valid} = 3'b01_1; // Bit 1, valid word
        4'b001x: {x_out, y_out, Valid} = 3'b10_1; // Bit 2, valid word
        4'b0001: {x_out, y_out, Valid} = 3'b11_1; // Bit 3, valid word
    endcase
end
endmodule

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

```

#### VHDL

```
entity Prob_4_57 is
  port (D0, D1, D2, D3: in Std_Logic; x_out, y_out: out Std_Logic; Valid: out Std_Logic);
end Prob_4_57;

architecture Behavioral of Prob_4_57 is
begin
  x_out & y_out & Valid <= "000" when D0 & D1 & D2 & D3 = '0000'; else
    "001" when D0 = 1; else
    "011" when D0 = 0 and D1 = 1; else
    "101" when D0 = 0 and D1 = 0 and D2 = 1; else
    "111" when D0 & D1 & D2 & D3 = '0001'; end if;

  endcase;
end process;
end Behavioral;
```

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

#### 4.58 (a)

##### Verilog

```
//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
```

Name	609	619	629	639
sig_in[31:0]	00001111111111111111111111111111			
sig_out_V1995[31:0]	00000001111111111111111111111111			

Name	34	44	54	64
sig_in[31:0]	11110000000000000000000000000000			
sig_out_V1995[31:0]	11111110000000000000000000000000			

##### VHDL

```
entity shift_right_by_3 is
```

```
  port ( sig_out: out Std_Logic_Vector (31 downto 0), sig_in: in Std_Logic_Vector (31 downto 0));
end shift_right_by_3;
```

```
architecture Behavioral of shift_right_by_3 is
```

```
  sig_out <= (sig_in[31] & sig_in[31] & sig_in[31] & sig_in(31: 3));
end Behavioral;
```

(b)

**Verilog**

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

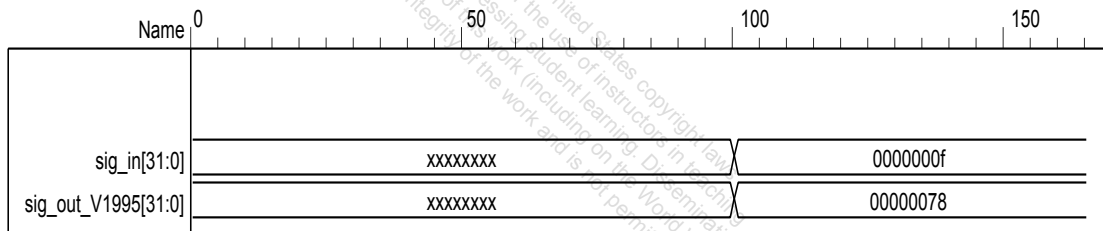
//module shift_left_by_3_V1995 (output reg [31: 0] sig_out, input [31: 0] sig_in);
//always @ (sig_in)
//    sig_out = {sig_in[28: 3], 3'b0};
endmodule
```

```
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
```



**VHDL**

```
entity shift_left_by_3 is
    port (sig_out: out Std_Logic_Vector (31 downto 0); sig_in: in Std_Logic_Vector (31 downto 0));
end shift_left_by_3;
```

```
architecture Behavioral of shift_left_by_3 is
    sig_out <= (sig_in(28:0) & sig_in'000); -- sig_out <= sig_in sla 3;
end Behavioral;
```

#### 4.59

##### Verilog

```
module BCD_to_Dec_Decoder (output reg [3: 0] Dec_out, input [3: 0] BCD_in);
  always @ (BCD_in) begin
    Dec_out = 0;
    case (BCD_in)
      4'b0000: Dec_out = 0;
      4'b0001: Dec_out = 1;
      4'b0010: Dec_out = 2;
      4'b0011: Dec_out = 3;
      4'b0100: Dec_out = 4;
      4'b0101: Dec_out = 5;
      4'b0110: Dec_out = 6;
      4'b0111: Dec_out = 7;
      4'b1xx0: Dec_out = 8;
      4'b1xx1: Dec_out = 9;
      default: Dec_out = 4'bxxxx;
    endcase
  end
endmodule
```

##### VHDL

```
entity BCD_to_Dec_Decoder (Dec_out: out integer), BCD_in: in
  Std_Logic_Vector (3 downto 0));
end BCD_to_Dec_Decoder
```

**architecture** Behavioral **of** BCD\_to\_Dec\_Decoder **is**

```
  process (BCD_in) begin
    Dec_out <= 0;
    case (BCD_in)
      when '0000' => Dec_out <= 0;
      when '0001' => Dec_out <= 1;
      when '0010' => Dec_out <= 2;
      when '0011' => Dec_out <= 3;
      when 'x100' => Dec_out <= 4;
      when 'x101' => Dec_out <= 5;
      when 'x110' => Dec_out <= 6;
      when 'x111' => Dec_out <= 7;
      when '1xx0' => Dec_out <= 8;
      when '1xx1' => Dec_out <= 9;
      when others => Dec_out <= "xxxx";
    end case;
  end process;
end Behavioral;
```

#### 4.60 Verilog

```
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.

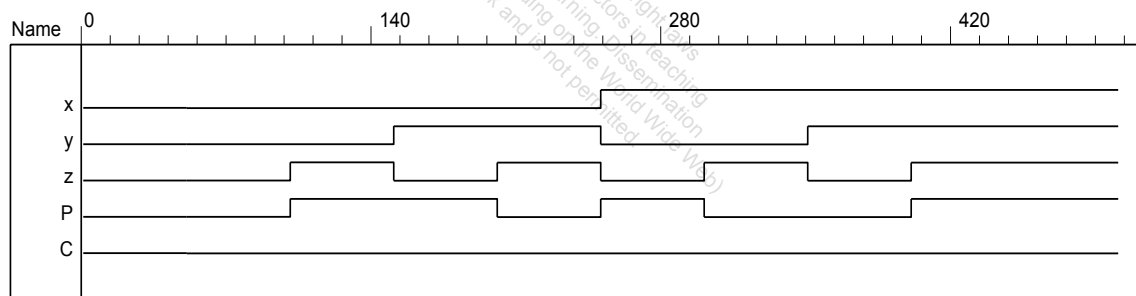
#### VHDL

```
entity Even_Parity_Checker_4 is
    port (P, C: out Std_Logic; x, y, z: in Std_Logic);
end Even_Parity_Checker_4;

architecture Boolean_Eq of Even_Parity_Checker_4 is
    component xor2_gate port (y: out bit; xin1, xin2: out bit);
begin
    G1: xor2_gate port map (y => w1, xin1 => x, xin2 => y);
    G2: xor2_gate port map (y => P, xin1 => w1, xin2 => z);
    G3: xor2_gate port map (C => w1, xin1 => w1, xin2 => w2);
    G4: xor2_gate port map (y => w2, xin1 => x, xin2 => P);
end Boolean_Eq;
```

#### 4.61 Verilog

```
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
```

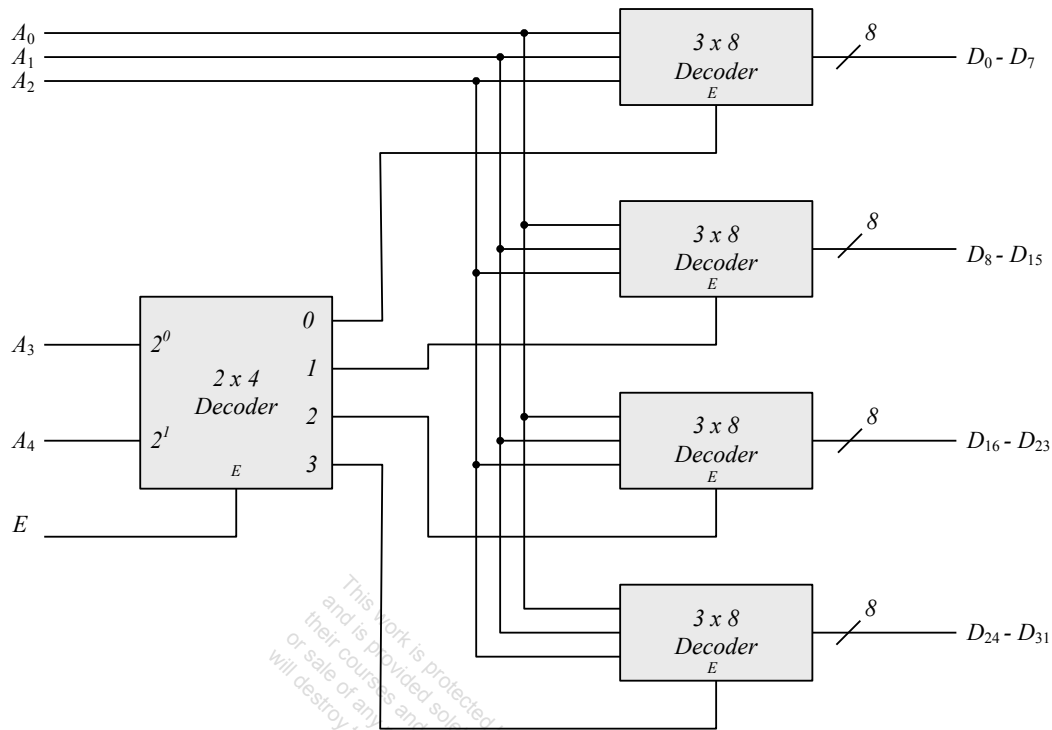


#### VHDL

```
entity Even_Parity_Checker_4 is
    port (P, C: out bit; x, y, z: in bit);
end Even_Parity_Checker_4;

architecture Boolean_Eq of Even_Parity_Checker_4 is
    w1 <= x xor y;
    P <= w1 xor z;
    C <= w1 xor w2;
    w2 <= z xor P;
end Boolean_Eq;
```

## 4.62



### Verilog

```

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

```

```

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

```

### VHDL

```

entity Decoder_5x32 is
    port (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: out Std_Logic;

```

```

        A4, A3, A2, A1, A0, E: in Std_Logic);

end Decoder_5x32;

architecture Gates of Decoder_5x32 is
    signal E3, E2, E1, E0: Std_Logic;
    component Decoder_3x8 port (D7, D6, D5, D4, D3, D2, D1, D0: out Std_Logic;
        in2, in1, in0, E: in Std_Logic);
    begin
        G1: Decoder_3x8 port map (D7, D6, D5, D4, D3, D2, D1, D0, A2, A1, A0, E0);
        G2: Decoder_3x8 (D15, D14, D13, D12, D11, D10, D9, D8, A2, A1, A0, E1);
        G3: Decoder_3x8 (D23, D22, D21, D20, D19, D18, D17, D16, in2, in1, in0, E2);
        G4: Decoder_3x8 (D31, D30, D29, D28, D27, D26, D25, D24, A2, A1, A0, E3);
        G5: Decoder_2x4 (E3, E2, E1, E0, A4, A3, E);
    end Gates;

entity Decoder_3x8 is
    port (D7, D6, D5, D4, D3, D2, D1, D0: out Std_Logic; in2, in1, in0, E: in Std_Logic);
end Decoder_3x8;

architecture Boolean_Eq of Decoder_3x8 is
    signal in0_bar, in1_bar, in2_bar: Std_Logic;
    component not_gate port (y: out Std_Logic; xin: in Std_Logic);
    component and3_gate port (y: out Std_Logic; xin1, xin2, xin3: in Std_Logic);
    component and4_gate port (y: out Std_Logic; xin1, xin2, xin3, xin4: in Std_Logic);
    begin
        G1: not_gate port map (y => in2_bar, xin => in2);
        G2: not_gate port map (y => in1_bar, xin => in1);
        G3: not_gate port map (y => in0_bar, xin => in0);
        G4: and4_gate port map (y => D0, xin1 => in2_bar, xin2 => in1_bar, xin3 => in0_bar, xin4 => E);
        G5: and4_gate port map (y => D1, xin1 => in2_bar, xin2 => in1_bar, xin3 => in0, xin4 => E);
        G6: and4_gate port map (y => D2, xin1 => in2_bar, xin2 => in1, xin3 => in0_bar, xin4 => E);
        G7: and4_gate port map (y => D3, xin1 => in2_bar, xin2 => in1, xin3 => in0, xin4 => E);
        G8: and4_gate port map (y => D4, xin1 => in2, xin2 => in1_bar, xin3 => in0_bar, xin4 => E);
        G9: and4_gate port map (y => D5, xin1 => in2, xin2 => in1_bar, xin3 => in0_bar, xin4 => E);
        G9: and4_gate port map (y => D6, xin1 => in2, xin2 => in1, xin3 => in0_bar, xin4 => E);
        G9: and4_gate port map (y => D7, xin1 => in2, xin2 => in1, xin3 => in0, xin4 => E);
    end Boolean_Eq;

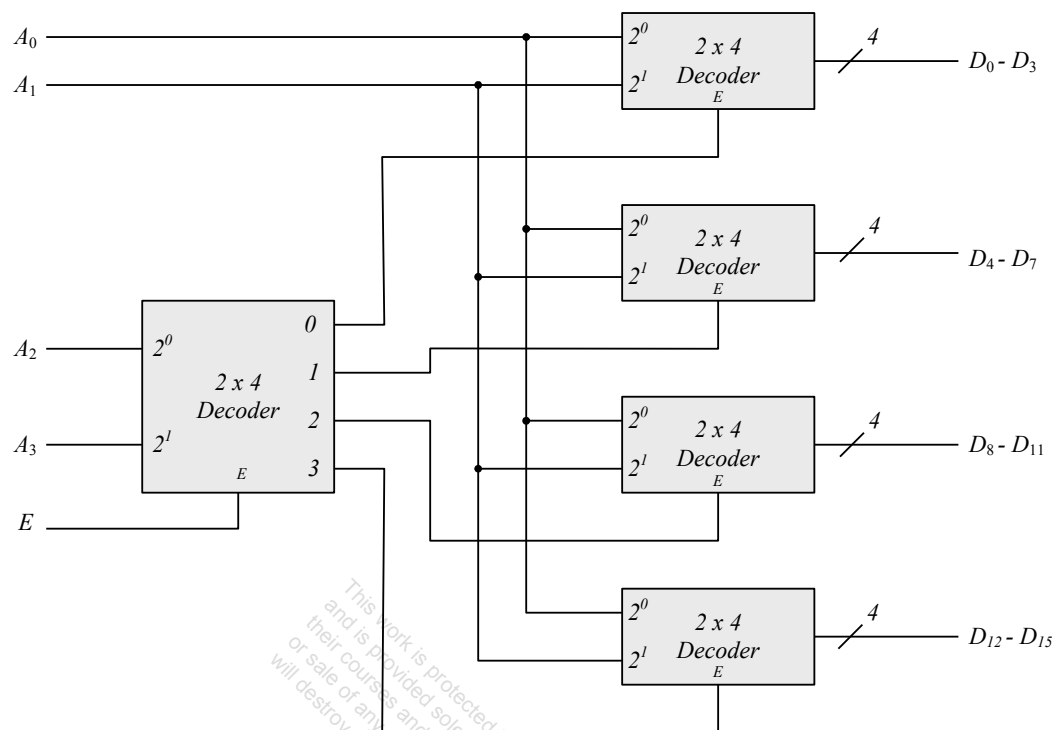
entity Decoder_2x4 is
    port (y0, y1, y2, y3: out Std_Logic; x0, x1, E: in Std_Logic);
end Decoder_2x4;

architecture Boolean_Eq of Decoder_2x4 is
    signal x0_bar, x1_bar: Std_Logic;
    begin
        y0 <= '0' when E = 'x'; else '1' when E and x0 and x1;
        y1 <= '0' when E = 'x'; else '1' when and x0 and x1_bar;
        y2 <= '0' when E = 'x'; else '1' when and x0_bar and x1;
        y3 <= '0' when E = 'x'; else '1' when and x0_bar and x1_bar;
    end Boolean_Eq;

```



### 4.63



#### Verilog

```
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
```

#### VHDL

```
entity Decoder_2x4
    port (D3, D2, D1, D0: out Std_Logic; in1, in0, E: in Std_Logic);
end Decoder_2x4;
```

#### architecture Boolean Eq of Decoder\_2x4 is

```
    signal in1_bar, in0_bar: Std_Logic;
begin
    in1_bar <= not in1;
    in0_bar <= not in0;
    D0 <= in1_bar and in0_bar and E;
    D1 <= in1_bar and in0 and E;
```

```

D2 <= in1 and in0_bar and E);
D3 <= in1 and in0 and E);
end Boolean_Eq;

entity Decoder_4x16 is
    port (D15, D14, D13, D12, D11, D10, D9, D8, D7, D6, D5, D4, D3, D2, D1, D0: Std_Logic;
          A3, A2, A1, A0, E: in Std_Logic);
end Decoder_4x16;

architecture Gates of Decoder_4x16 is
    signal E3, E2, E1, E0: Std_Logic;
    component Decoder_2x4 port (D3, D2, D1, D0: out Std_Logic; in1, in0, E0: in Std_Logic);
begin
    M0: Decoder_2x4 port map (D3, D2, D1, D0, in1, in0, E0);
    M1: Decoder_2x4 port map (D7, D6, D5, D4, in1, in0, E1);
    M2: Decoder_2x4 port map (D11, D10, D9, D8, in1, in0, E2);
    M3: Decoder_2x4 port map (D15, D14, D13, D12, in1, in0, E3);
    M4: Decoder_2x4 port map (E3, E2, E1, E0, A3, A2, E);
endmodule

```

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

## 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  $V = 1$

### Verilog

**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**

### VHDL

**entity** Prob\_4\_64 is

**port** (x, y, x, V: **out** Std\_Logic, D0, D1, D2, D3, D4,D5 D6, D7: **in** Std\_Logic);

**end** Prob\_4\_64;

**architecture** Behavioral is

**begin**

**process** (D0 D1, D2, D3, D4,D5, D6, D7)

**case** (D0 & D1 & D2 & D3 & D4 & D5 & D6 & D7)

**when** '0000\_0000' => (x & y & x & V) <= 'xxx0';

**when** '1000\_0000' => (x & y & x & V) <= '0001';

**when** '1000\_0000' => (x & y & x & V) <= '0011';

**when** '0010\_0000' => (x & y & x & V) <= '0101';

**when** '0001\_0000' => (x & y & x & V) <= '0111';

**when** '0000\_1000' => (x & y & x & V) <= '1001';

**when** '0000\_0100' => (x & y & x & V) <= '1011';

**when** '0000\_0010' => (x & y & x & V) <= '1001';

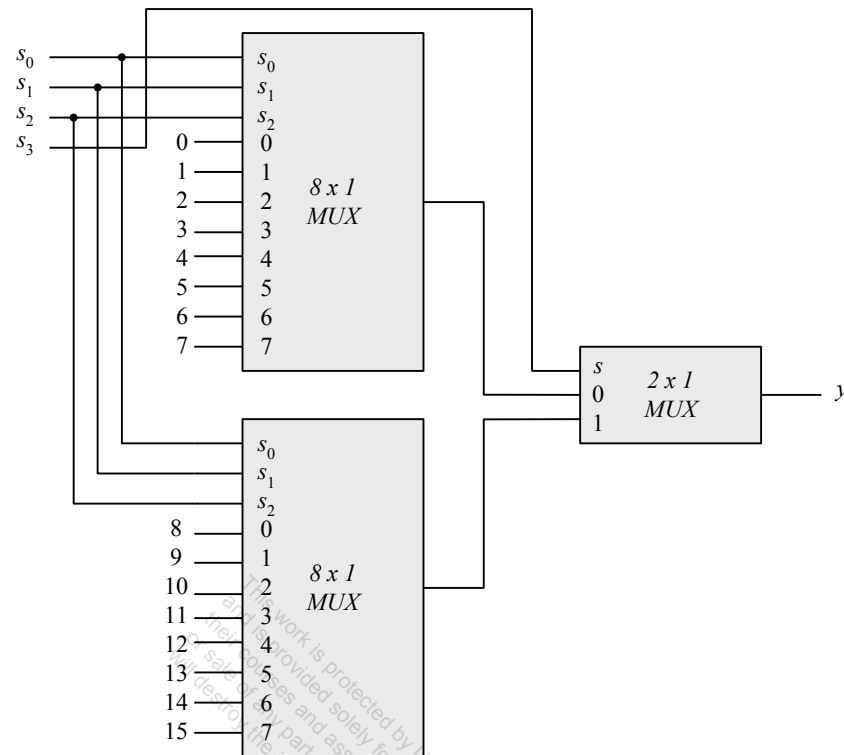
**when** '0000\_0001' => (x & y & x & V) <= '1111';

**when others:** => (x & y & x & V) <= '1010'; // Use for error detection

**endcase**

**end** Behavioral;

#### 4.65



#### Verilog

```
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, s1, 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

```

### **VHDL**

```

entity Mux_2x1 is
  port ( y_out: out Std_Logic; in1, in0, sel: in Std_Logic);
end Mux_2x1;

```

**architecture** Gates of Mux\_2x1 **is**

```

  component not_gate port (y: out, Std_Logic; xin: in Std_Logic);
  component and2_gate port (y: out, Std_Logic; xin1, xin2: in Std_Logic);
  component or2_gate port (y: out, Std_Logic; xin1, xin2: in Std_Logic);

```

**begin**

```

  G1: not_gate port map (y => sel_bar, xin => sel);
  G2: and2_gate port map (y => y0, xin1 => in0, xin2 => sel);
  G3: and2_gate port map (y => y1, xin1 => in1, xin2 => sel);
  G4: or2_gate port map (y => y1, in1 => in1, in2 => sel);
  G5: or (y_out, in0, in1, sel_bar

```

**end** Gates;

**entity** Mux\_4x1 **is**

```

  port (y_out: out Std_Logic; in3, in2, in1, in0, sel1, sel0: in Std_Logic);

```

**end** Mux\_4x1;

**architecture** Gates of Mux\_4x1 **is**

```

  component not_gate port (y: out, Std_Logic; xin: in Std_Logic);
  component and2_gate port (y: out, Std_Logic; xin1, xin2: in Std_Logic);
  component or2_gate port (y: out, Std_Logic; xin1, xin2: in Std_Logic);
  component Mux_2x1 port ( y: out Std_Logic; xin1, xin2, xin3: in Std_Logic);

```

**begin**

```

  G1: not_gate port map (y => sel_1_bar, xin => sel1);
  G2: and2_gate port map (y =: s0, xin1: sel_1_bar, xin2 => sel0);
  G3: and2_gate port map (y > s1, xin1 => sel[1], xin2 => sel0);
  G4: Mux_2x1 port map ( y => y_out, xin1 => in0, xin2 => in1, xin3 => s0);
  G5: Mux_2x1 port map ( y => y_out, xin1 => in2, xin2 => in3, xin3 => s1);

```

**end** Gates;

**entity** Mux\_8x1 **is**

```

  port (y: out Std_Logic; in7, in6, in5, in4, in3, in2, in1, in0, sel2, sel1, sel0: in Std_Logic);

```

**end** Mux\_8x1;

**architecture** Structural of Mux\_8x1 **is**

```

  component MUX_4x1 port (y_out: out Std_Logic; in3, in2, in1, in0, sel1, sel0: in Std_Logic);
  component Mux_2x1 port ( y_out: out Std_Logic; in1, in0, sel: in Std_Logic);

```

**end** Mux\_2x1;

```

  G1: Mux_4x1 port map (y_out => y_M0, in3 => in3, in2 => in2, in1 => in1, in0 => in0,
    sel1 => sel1, sel0 => sel0);
  G2: Mux_4x1 (y_out => y_M1, in3 => in7, in2 => in6, in1 => in5, in0 => in4,
    sel1 => sel1, sel0 => sel0);
  G3: Mux_2x1 M2 (y_out, in2 => y_M0, in1 => y_M1, sel => sel2);

```

**end** Structural;

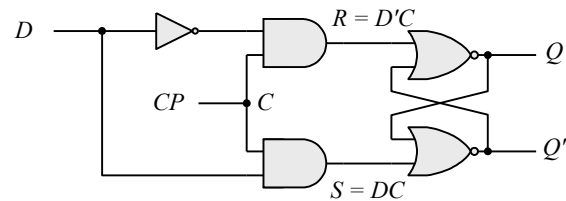
```
entity Mux_16x1 is
  port (y_out: out Std_Logic; in15, in14, in13, in12, in11, in10, in9, in8, in7, in6, in5, in4,
        in3, in2, in1, in0, sel3, sel2, sel1, sel0: in Std_Logic);
end Mux_16x1;

architecture Structural of Mux_16x1 is
  component Mux_8x1 port (y: out Std_Logic; in7, in6, in5, in4, in3, in2, in1, in0,
        sel2, sel1, sel0: in Std_Logic);
  component Mux_2x1 port ( y_out: out Std_Logic; in1, in0, sel: in Std_Logic);
begin
  G0: Mux_8x1 port map (> y_M0, in7 => in7, in6 => in6, in5 => in5, in4 => in4, in3 => in3,
        in2 => in2, in1 => in1, in0 => in0, sel2 => sel2, sel1 => sel1, sel0 => sel0);
  G1: Mux_8x1 port map (y => y_M1, in7: in15, in6 => in14, in5 => in13, in4 => in12, in3 => in11,
        in2 => in10, in1 => in9, in0 => in8, sel2 => sel2, sel1 => sel1, sel0 => sel0);
  G2: Mux_2x1 port map (y => y_out, in1 => y_M0, in0 => y_M1, sel => sel3);
end Structural;
```

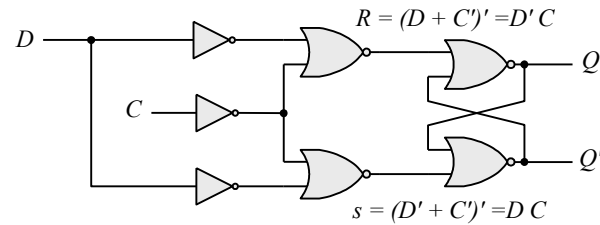
This work is protected by United States copyright laws  
and is provided solely for the use of instructors in teaching  
their courses and assessing student learning. Dissemination  
or sale of any part of this work (including on the World Wide Web)  
will destroy the integrity of the work and is not permitted.

## CHAPTER 5

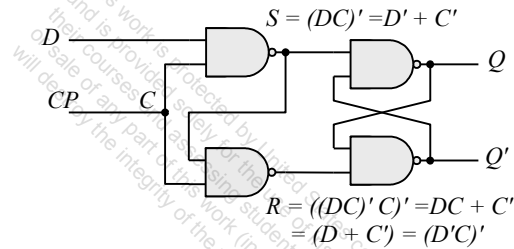
### 5.1 (a)



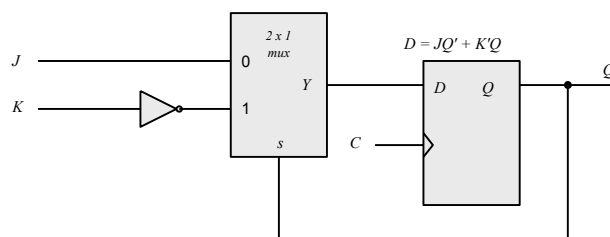
### (b)



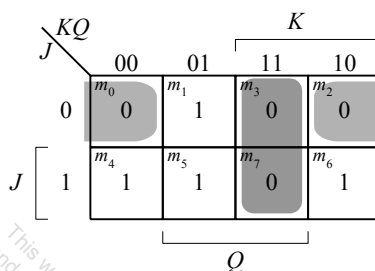
### (c)



## 5.2



$$5.3 \quad Q'(t+1) = (JQ' + K'Q)' = (J' + Q)(K + Q') = J'Q' + KQ$$



## 5.4

(a)

P	N	Q(t+1)
0	0	0
0	1	Q(t)
1	0	Q'(t)
1	1	1

(b)

P	N	Q(t)	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

(c)

Q(t)	Q(t+1)	P	N
0	0	0	x
0	1	1	x
1	0	x	0
1	1	x	1

(d) Connect P and N together.

Karnaugh map for Q(t+1) = PQ' + NQ:

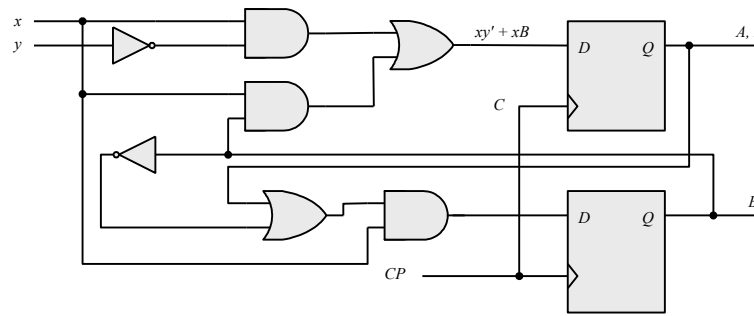
$Q(t+1) = PQ' + NQ$

## 5.5

The truth table describes a combinational circuit.  
 The state table describes a sequential circuit.  
 The characteristic table describes the operation of a flip-flop.  
 The excitation table gives the values of flip-flop inputs for a given state transition.  
 The four equations correspond to the algebraic expression of the four tables.



## 5.6

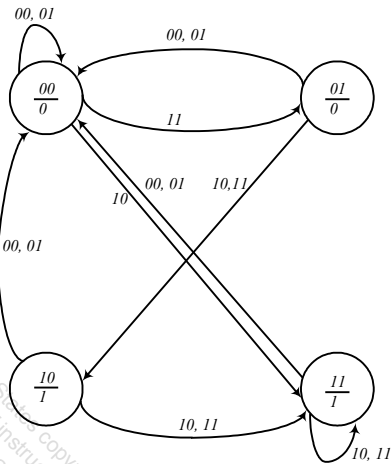


(b)

$$\begin{aligned} A(t+1) &= xy' + xB \\ B(t+1) &= xA + xB' \\ z &= A \end{aligned}$$

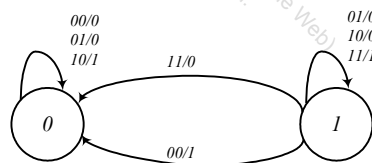
Present state		Inputs		Next state		Output
A	B	x	y	A	B	z
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

(c)



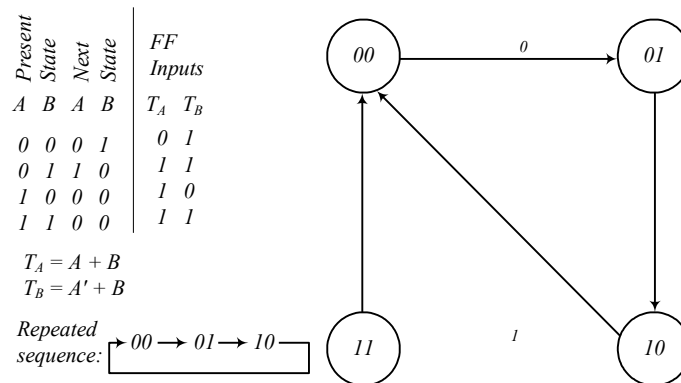
## 5.7

Present state	Inputs		Next state	Output
Q	x	y	Q	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

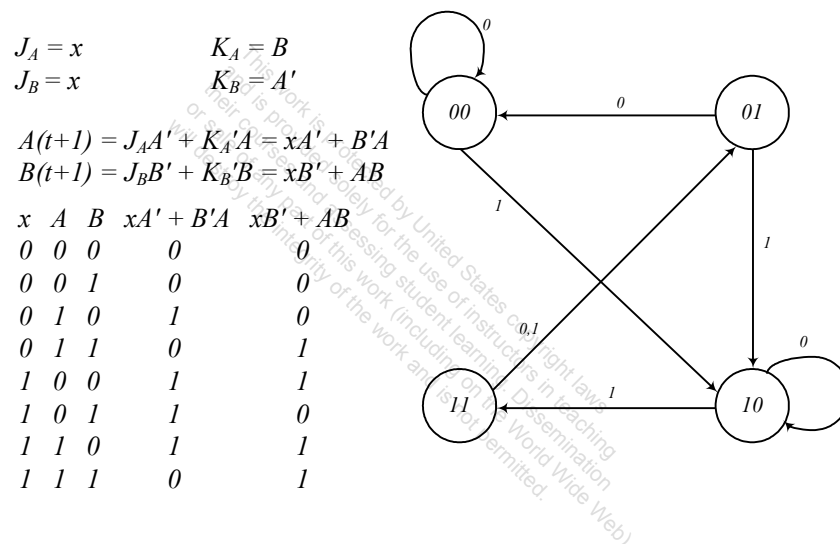


$$\begin{aligned} S &= x \oplus y \oplus Q \\ Q(t+1) &= xy + xQ + yQ \end{aligned}$$

## 5.8 A counter with a repeated sequence of 00, 01, 10.



## 5.9



# 5.10

$$J_A = Bx + B'y' \quad J_B = A'x$$

$$K_A = B'xy' \quad K_B = A + xy' \quad z = Ax'y' + Bx'y'$$

(b)

Present state		Inputs		Next state		Output	FF Inputs			
A	B	x	y	A	B	z	$J_A$	$K_A$	$J_B$	$K_B$
0	0	0	0	1	0	0	1	0	0	0
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	1	0	1	1	1	1
0	0	1	1	0	1	0	0	0	1	0
0	1	0	0	0	1	1	0	0	0	0
0	1	0	1	0	1	0	0	0	0	0
0	1	1	0	1	0	0	1	0	1	0
0	1	1	1	1	1	0	1	0	1	0
1	0	0	0	1	0	0	1	0	0	1
1	0	0	1	1	0	0	0	0	0	1
1	0	1	0	0	0	0	1	1	0	1
1	0	1	1	1	0	0	0	0	0	1
1	1	0	0	1	0	1	0	0	0	1
1	1	0	1	1	0	0	0	0	0	1
1	1	1	0	1	0	0	1	0	0	1
1	1	1	1	1	0	1	1	0	0	1

(c)

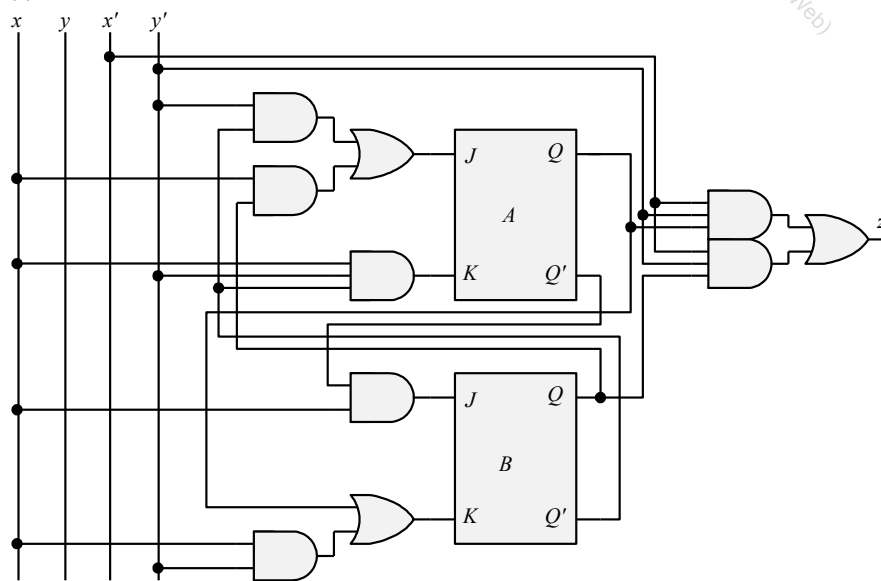
xy		x			
		00	01	11	10
AB	00	$m_0$ 1	$m_1$	$m_3$	$m_2$ 1
	01	$m_4$	$m_5$	1	$m_6$ 1
	11	$m_{12}$ 1	$m_{13}$ 1	$m_{15}$ 1	$m_{14}$ 1
	10	$m_8$ 1	$m_9$ 1	$m_{11}$ 1	$m_{10}$

$$A(t+1) = Ax' + Bx + Ay + A'B'y'$$

xy		x			
		00	01	11	10
AB	00	$m_0$	$m_1$	1	1
	01	$m_4$ 1	$m_5$ 1	$m_7$ 1	$m_6$
	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
	10	$m_8$	$m_9$	$m_{11}$	$m_{10}$

$$B(t+1) = A'B'x + A'B'(x' + y)$$

(a)



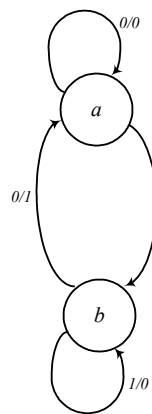
5.11

(a)

Present state: 00 00 01 00 01 11 00 01 11 10 00 01 11 10 10  
 Input: 0 1 0 1 1 0 1 1 1 0 1 1 1 1 0  
 Output: 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1  
 Next state: 00 01 00 01 11 00 01 11 10 00 01 11 10 10 00

(b)

State labels: a: 00, b: 10, c: 11, d: 01  
 c is equivalent to b  
 d is equivalent to c



(c)

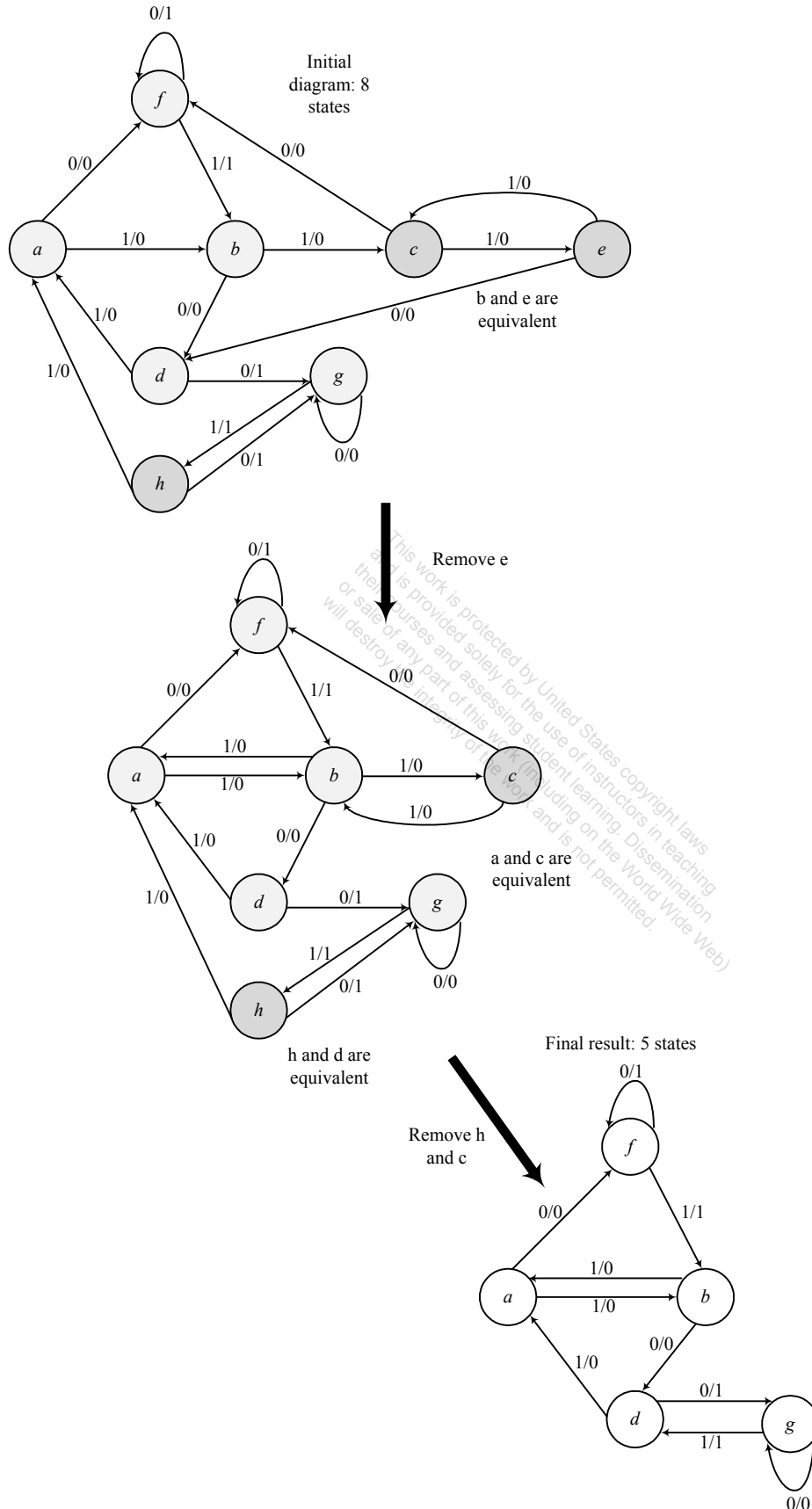
input	state	next st	output
0	0	0	0
1	0	1	0
0	1	0	0
1	1	1	1

State machine: D-flop with direct input of the input to the original machine;  
 output logic:  $y = (!\text{input}) \&\& (\text{state} == b)$

5.12

Present state	Next state		Output	
	0	1	0	1
a	f	b	0	0
b	d	a	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

**Note:** Equivalent states: b = e, a = c, h = d



### 5.13 Equivalent

(a) State:  $a f b c e d g h g g h a$   
 Input:  $0 1 1 1 0 0 1 0 0 1 1$   
 Output:  $0 1 0 0 0 1 1 1 0 1 0$

(b) State:  $a f b a b d g d g g d a$   
 Input:  $0 1 1 1 0 0 1 0 0 1 1$   
 Output:  $0 1 0 0 0 1 1 1 0 1 0$   
 Equivalent states:  $b = e, a = c, h = d$

### 5.14

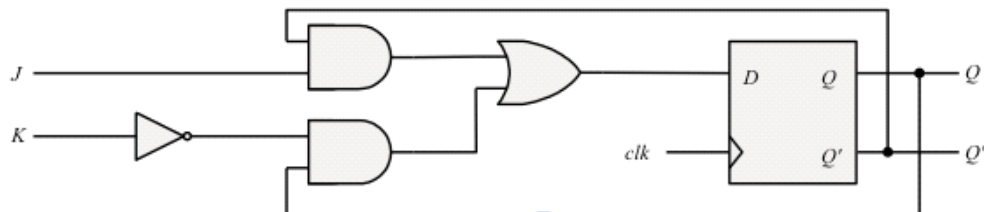
	Present state ABCDE	Next state		Output	
		x=0	x=1	x=1	x=0
a	00001	00001	00010	0	0
b	00010	00100	01000	0	0
c	00100	00001	01000	0	0
d	01000	10000	01000	0	1
e	10000	00001	01000	0	1

### 5.15 $D_Q = Q'J + QK'$

Present state	Inputs		Next state	
Q	J	K	Q	
0	0	0	0	No change
0	0	1	0	Reset to 0
0	1	0	1	Set to 1
0	1	1	1	Complement
1	0	0	1	No change
1	0	1	0	Reset to 0
1	1	0	1	Set to 1
1	1	1	0	Complement

JK		J			
		00	01	11	10
Q	0	$m_0$	$m_1$	$m_3$ 1	$m_2$ 1
	1	$m_4$ 1	$m_5$	$m_7$	$m_6$ 1
		K			

$$Q(t+1) = DQ = Q'J + K'Q$$



**5.16 (a)**  $D_A = Ax' + Bx$

$$D_B = A'x + Bx'$$

Present state		Input x	Next state	
A	B		A	B
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

		B			
		00	01	11	10
A	0	$m_0$	$m_1$	$m_3$ 1	$m_2$
	1	$m_4$ 1	$m_5$	$m_7$ 1	$m_6$ 1

$$D_A = Ax' + Bx$$

		B			
		00	01	11	10
A	0	$m_0$	$m_1$ 1	$m_3$ 1	$m_2$ 1
	1	$m_4$	$m_5$	$m_7$	$m_6$ 1

$$D_B = A'x + Bx'$$

**(b)**  $D_A = A'x + Ax'$

$$D_B = AB + Bx'$$

Present state		Input x	Next state	
A	B		A	B
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	1

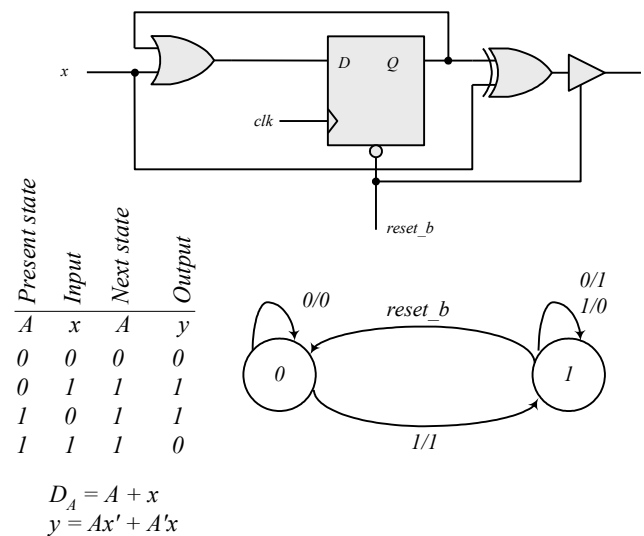
		B			
		00	01	11	10
A	0	$m_0$	$m_1$ 1	$m_3$ 1	$m_2$
	1	$m_4$ 1	$m_5$	$m_7$	$m_6$ 1

$$D_A = A'x + Ax'$$

		B			
		00	01	11	10
A	0	$m_0$	$m_1$ 1	$m_3$	$m_2$ 1
	1	$m_4$	$m_5$	$m_7$ 1	$m_6$ 1

$$D_B = AB + Bx'$$

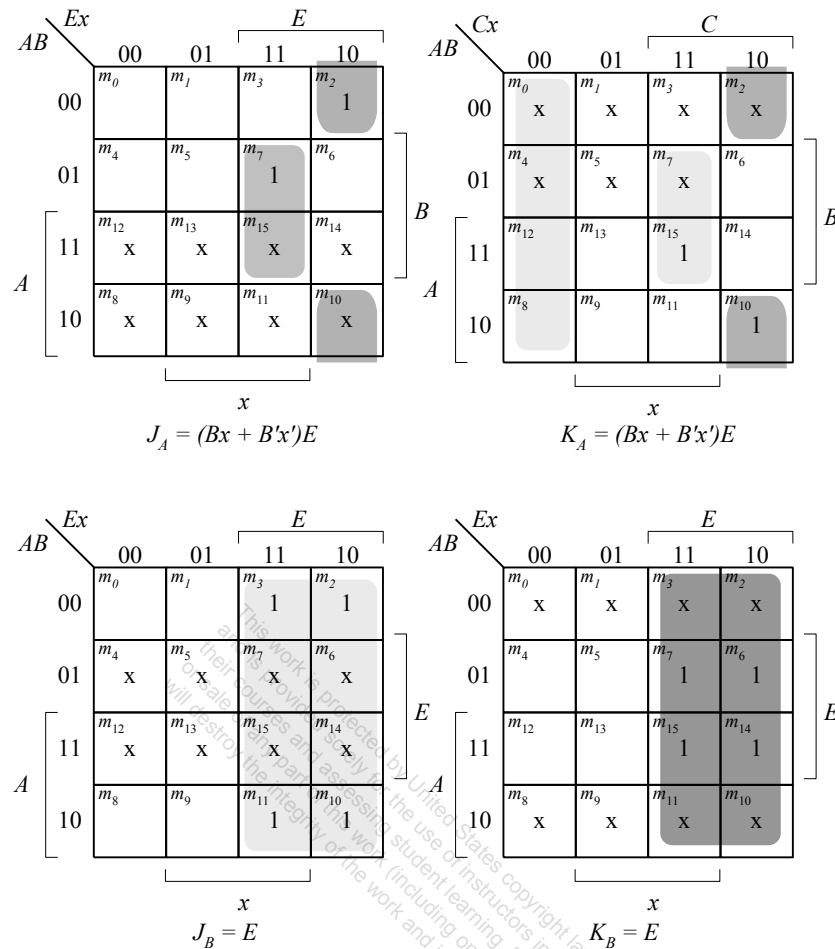
**5.17** The output is 0 for all 0 inputs until the first 1 occurs, at which time the output is 1. Thereafter, the output is the complement of the input. The state diagram has two states. In state 0: output = input; in state 1: output = input'.



### 5.18 Binary up-down counter with enable E.

Present state	Input	Next state	Flip-flop inputs			
A B	x	A B	J <sub>A</sub>	K <sub>A</sub>	J <sub>B</sub>	K <sub>B</sub>
0 0	0 1	0 0	0	x	0	x
0 0	0 1	0 0	0	x	0	x
0 0	1 0	1 1	1	x	1	x
0 0	1 1	0 1	0	x	1	x
0 1	0 0	0 1	0	x	x	0
0 1	0 1	0 1	0	x	x	0
0 1	1 0	0 1	0	x	x	1
0 1	1 1	1 0	1	x	x	1
1 0	0 0	1 0	x	0	1	0
1 0	0 1	1 0	x	0	1	0
1 0	1 0	0 1	x	1	x	1
1 0	1 1	1 1	x	0	x	1
1 1	0 0	1 1	x	0	x	0
1 1	0 1	1 1	x	0	x	0
1 1	1 0	1 1	1	0	x	1
1 1	1 1	1 1	x	1	x	1

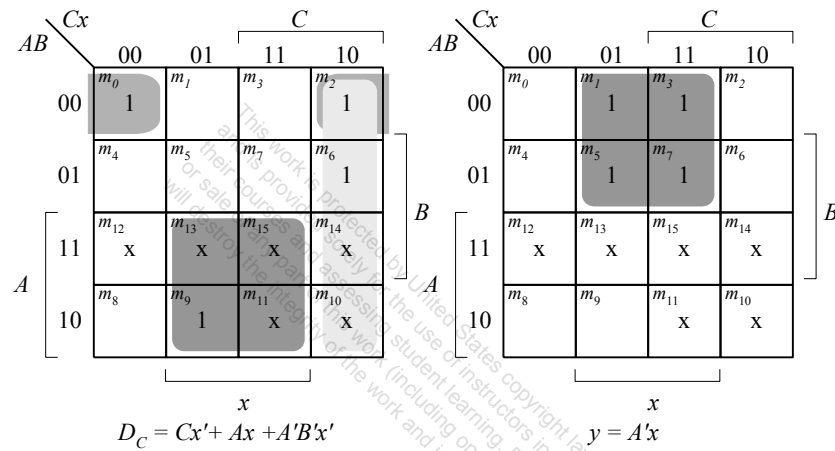
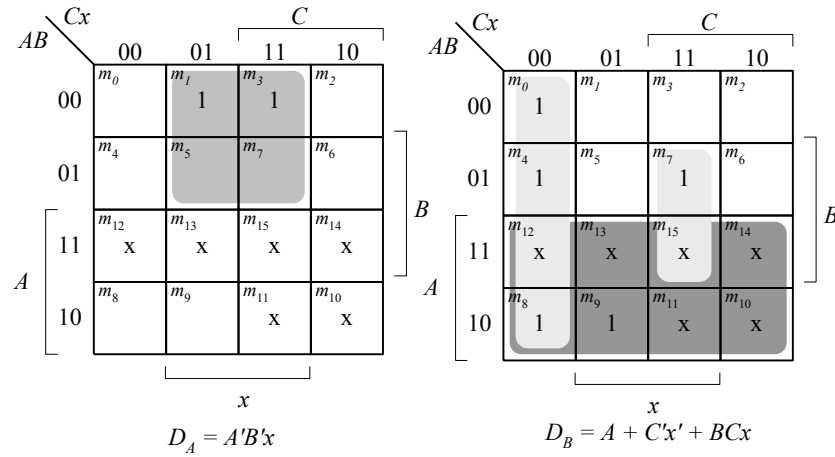




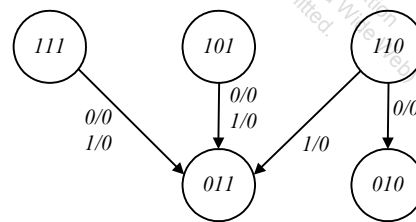
5.19 (a) Unused states (see Fig. P5.19): 101, 110, 111.

Present state	Input	Next state	Output
ABC	x	ABC	y
000	0	011	0
000	1	100	1
001	0	001	0
001	1	100	1
010	0	010	0
010	1	000	1
011	0	001	0
011	1	010	1
100	0	010	0
100	1	011	1

$$d(A, B, C, x) = \Sigma (10, 11, 12, 13, 14, 15)$$



The machine is self-correcting, i.e., the unused states transition to known states.

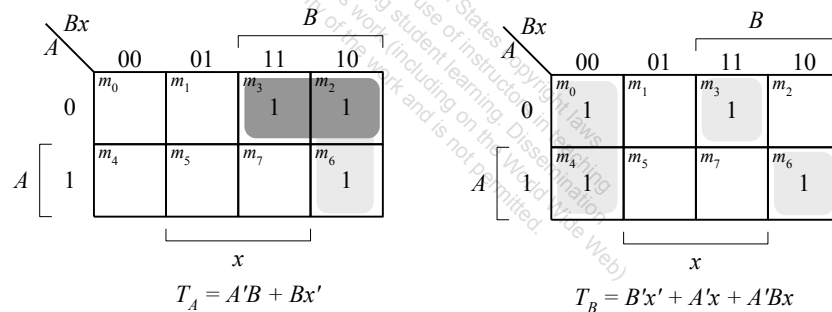


(b) With JK flip=flops, the state table is the same as in (a).

Flip-flop inputs					
$J_A$	$K_A$	$J_B$	$K_B$	$J_C$	$K_C$
0	x	1	x	1	x
1	x	0	x	0	x
0	x	0	x	x	0
1	x	0	x	x	1
0	x	x	0	0	x
0	x	x	1	0	x
0	x	x	1	x	0
0	x	x	0	x	1
x	1	1	x	0	x
x	1	1	x	1	x

$J_A = B'x$        $K_A = 1$   
 $J_B = A + C'x'$        $K_B = C'x + Cx'$   
 $J_C = Ax + A'B'x'$        $K_C = x$   
 $y = A'x$   
 The machine is self-correcting  
 because  $K_A = 1$ .

5.20 From state table 5.4:  $T_A(A, B, x) = \Sigma(2, 3, 6)$ ,  $T_B(A, B, x) = \Sigma(0, 3, 4, 6)$ .



5.21 (a) The statements associated with an **initial** keyword execute once, in sequence, with the activity expiring after the last statement completes execution; the statements associated with the **always** keyword execute repeatedly, subject to timing control (e.g., #10).

(b) Assignments to variables execute sequentially and have immediate effect. Assignments to signals execute in sequence, but their effect occurs after all of the statements have been evaluated.

## 5.22

### VHDL supplement

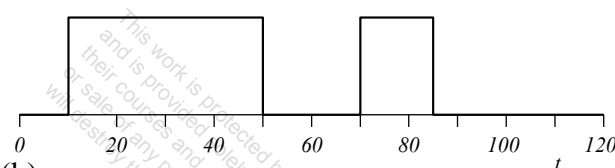
(a)

```
w := 0;
w := 1 after 10 ns;
w := 0 after 40 ns;
w := 1 after 20 ns;
w := 0 after 15 ns;
```

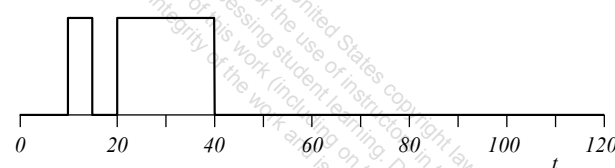
(b)

```
w <= 0;
w <= 1 after 10 ns;
w <= 0 after 40 ns;
w <= 1 after 20 ns;
w <= 0 after 15 ns;
```

(a)



(b)



## 5.23 Verilog

(a)  $RegA = 125$ ,  $RegB = 125$

(b)  $RegA = 125$ ,  $RegB = 50$

### VHDL

(a)  $RegA = 125$ ,  $RegB = 125$

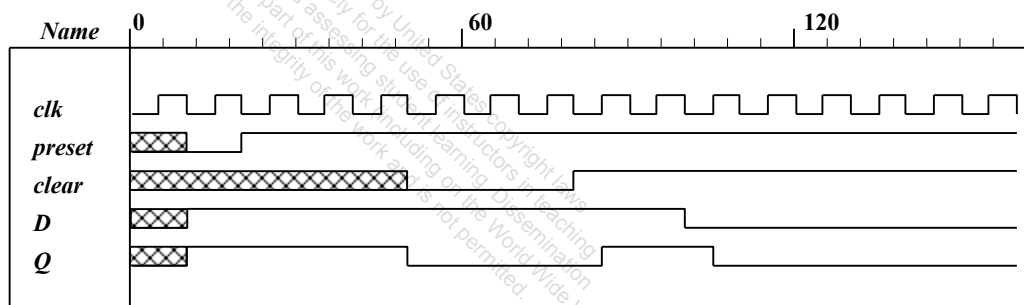
(b)  $RegA = 125$ ,  $RegB = 50$

## 5.24 (a) D flip-flop, positive edge-triggered, active-low, asynchronous preset and clear

```
module DFF (output reg Q, input D, clk, preset, clear);
    always @ (posedge clk, negedge preset, negedge clear )
        if (preset == 0) Q <= 1'b1;
        else if (clear == 0) Q <= 1'b0;
        else Q <= D;
endmodule
```

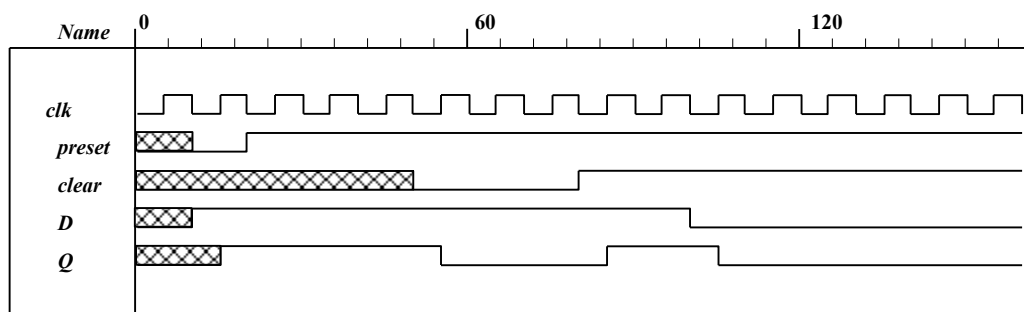
```
module t_DFF ();
    wire Q;
    reg clk, preset, clear;
    reg D;
    DFF M0 (Q, D, clk, preset, clear);

    initial #160 $finish;
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial fork
        #10 preset = 0;
        #20 preset = 1;
        #50 clear = 0;
        #80 clear = 1;
        #10 D = 1;
        #100 D = 0;
        #200 D = 1;
    join
endmodule
```



## (b) D flip-flop, positive edge-triggered, active-low, synchronous preset and clear

```
module DFF (output reg Q, input D, clk, preset, clear);
    always @ (posedge clk)
        if (preset == 0) Q <= 1'b1;
        else if (clear == 0) Q <= 1'b0;
        else Q <= D;
endmodule
```



## 5.25 Verilog

```
module Quad_Input_DFF (output reg Q, input D1, D2, D3, D4, s1, s0, clk, reset_b);
    always @ (posedge clk, negedge reset_b)
        if (reset_b == 1'b0) Q <= 0;
        else case ({s1, s0})
            2'b00: Q <= D1;
            2'b01: Q <= D2;
            2'b10: Q <= D3;
            2'b11: Q <= D4;
        endcase
endmodule
```

```
module t_Quad_Input_DFF ();
    wire Q;
    reg D1, D2, D3, D4, s1, s0, clk, reset_b;
    Quad_Input_DFF M0 (Q, D1, D2, D3, D4, s1, s0, clk, reset_b);
    initial #350 $finish;
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial fork
        begin s1 = 0; s0 = 0; end
        #40 begin s1 = 0; s0 = 1; end
        #80 begin s1 = 1; s0 = 0; end
        #120 begin s1 = 1; s0 = 0; end
        #160 begin s1 = 1; s0 = 1; end
    join
    initial fork
        begin D1 = 0; forever #10 D1 = ~D1; end
        begin D2 = 1; forever #20 D2 = ~D2; end
        begin D3 = 0; forever #10 D3 = ~D3; end
        begin D4 = 0; forever #20 D4 = ~D4; end
    join
    initial fork
        #2 reset_b = 1;
        #3 reset_b = 0;
        #4 reset_b = 1;
    join
endmodule
```

## VHDL

```
entity Quad_Input_DFF is
    port (Q: out Std_Logic; D1, D2, D3, D4, s1, s0, clk, reset_b: in Std_Logic);
end Quad_Input_DFF;
```

```
architecture Behavioral of Quad_Input_DFF is
begin
    process (clk, reset_b) begin
        if (reset_b'event and reset_b = '0') then
            Q <= '0';
        elsif (clk'event and clk = '1') then
            case (s1 & s0) is
                when '00' => Q <= D1;
                when '01' => Q <= D2;
                when '10' => Q <= D3;
                when '11' => Q <= D4;
            end case;
        end process;
    end Behavioral;
```

## 5.26 (a)

$$Q(t + 1) = JQ' + K'Q$$

When  $Q = 0$ ,  $Q(t + 1) = J$

When  $Q = 1$ ,  $Q(t + 1) = K'$

## (b)

### Verilog

```
module JK_Behavior_a (output reg Q, input J, K, CLK, reset_b);
  always @ (posedge CLK, negedge reset_b)
    if (reset_b == 0) Q <= 0; else
      if (Q == 0)    Q <= J;
      else          Q <= ~K;
endmodule
```

Alternative:

```
module JK_Behavior_b (output reg Q, input J, K, CLK, reset_b);
  always @ (posedge CLK, negedge reset_b) // Asynch reset_b
    if (reset_b == 0) Q <= 0;
    else
      case ({J, K})
        2'b00: Q <= Q;
        2'b01: Q <= 0;
        2'b10: Q <= 1;
        2'b11: Q <= ~Q;
      endcase
endmodule
```

```
module t_Prob_5_26 ();
  wire Q_a, Q_b;
  reg J, K, clk, reset_b;
  JK_Behavior_a M0 (Q_a, J, K, clk, reset_b);
  JK_Behavior_b M1 (Q_b, J, K, clk, reset_b);

  initial #100 $finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork
    #2 reset_b = 1;
    #3 reset_b = 0; // Initialize to s0
    #4 reset_b = 1;
    J = 0; K = 0;
    #20 begin J = 1; K = 0; end
    #30 begin J = 1; K = 1; end
    #40 begin J = 0; K = 1; end
    #50 begin J = 1; K = 1; end
  join
endmodule
```

### VHDL

```
entity JK_Behavior_a is
  port (Q: out Std_Logic; J, K, CLK, reset_b: in Std_Logic);
end JK_Behavior_a;
```

```
architecture Behavioral of JK_Behavior is
```

```
begin
process (CLK)
  if (CLK'event and reset_b = '0') then Q <= 0;
  elsif (Q = 0) then Q <= J;
  else Q <= not K; end if;
end Behavioral;
```

**Alternative:**

```
entity JK_Behavior_b
  port (Q: out Std_Logic; J, K, CLK, reset_b: in Std_Logic);
end JK_Behavior_b;
```

```
architecture Behavioral of JK_Behavior_b is
begin
  process (CLK, reset_b) // Asynch reset_b
  begin
    if (reset_b = '0') Q <= '0';
    else if (CLK'event and CLK = '1') then
      case (J & K) is
        when '00' => Q <= Q;
        when '01' => Q <= 0;
        when '10' => Q <= 1;
        when '11' => Q <= not Q;
      endcase
    end process;
  endmodule
```

## 5.27 Verilog

**// Mealy FSM zero detector (See Fig. 5.16)**

```
module Mealy_Zero_Detector (
  output reg y_out,
  input x_in, clock, reset
);
  reg [1: 0] state, next_state;
  parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;

  always @ (posedge clock, negedge reset) // state transition
  if (reset == 0) state <= S0;
  else state <= next_state;

  always @ (state, x_in) // Form the next state
  case (state)
    S0: begin y_out = 0; if (x_in) next_state = S1; else next_state = S0; end
    S1: begin y_out = ~x_in; if (x_in) next_state = S3; else next_state = S0; end
    S2: begin y_out = ~x_in; if (~x_in) next_state = S0; else next_state = S2; end
    S3: begin y_out = ~x_in; if (x_in) next_state = S2; else next_state = S0; end
  endcase
```

**endmodule**

```
module t_Mealy_Zero_Detector;
  wire t_y_out;
  reg t_x_in, t_clock, t_reset;
```

```
Mealy_Zero_Detector M0 (t_y_out, t_x_in, t_clock, t_reset);
```

```
initial #200 $finish;
initial begin t_clock = 0; forever #5 t_clock = ~t_clock; end
```

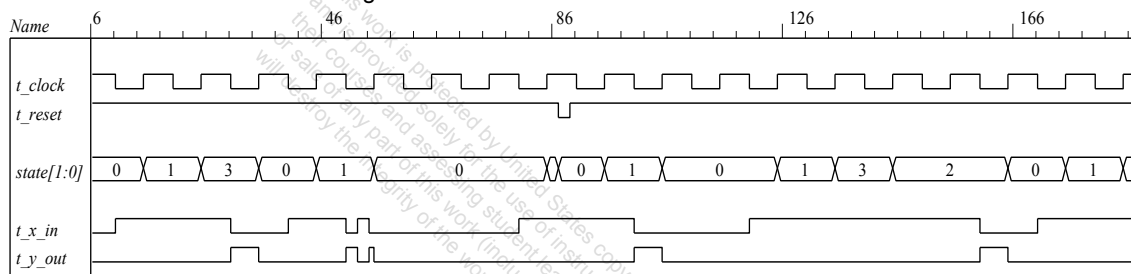


```

initial fork
    t_reset = 0;
    #2 t_reset = 1;
    #87 t_reset = 0;
    #89 t_reset = 1;
    #10 t_x_in = 1;
    #30 t_x_in = 0;
    #40 t_x_in = 1;
    #50 t_x_in = 0;
    #52 t_x_in = 1;
    #54 t_x_in = 0;
    #70 t_x_in = 1;
    #80 t_x_in = 1;
    #70 t_x_in = 0;
    #90 t_x_in = 1;
    #100 t_x_in = 0;
    #120 t_x_in = 1;
    #160 t_x_in = 0;
    #170 t_x_in = 1;
join
endmodule

```

Note: Simulation results match Fig. 5.22.



## VHDL

**entity** Mealy\_Zero\_Detector **is**

**port** (y\_out: **out** Std\_Logic; x\_in, clock, reset: **in** Std\_Logic);

**end** Mealy\_Zero\_Detector;

**architecture** Behavioral **of** Mealy\_Zero\_Detector **is**

**type** state\_type **is** S0, S1, S2, S3;

**signal** state, next\_state: state\_type;

**begin**

**process** (clock, reset)   // state transition

**begin**

**if** (reset'event **and** reset = '0') **then** state <= S0;

**else if** (clock'event **and** clock = '1') **then** state <= next\_state;

**end process**;

**process** (state, x\_in)   // Form the next state

**begin**

        y\_out <= S0;

**case** (state) **is**

**when** S0 => **if** x\_in = '0' next\_state <= S1; **else** next\_state <= S0;

**when** S1 => y\_out <= not x\_in; **if** (x\_in = '1') **then** next\_state = S3; **else** next\_state = S0;

**when** S2 => y\_out = not x\_in; **if** (x\_in = '0') **then** next\_state = S0; **else** next\_state = S2;

**when** S3 => y\_out = not x\_in; **if** (x\_in = '1') **then** next\_state = S2; **else** next\_state = S0;

**end case**;

**end process**;

**end** Behavioral;

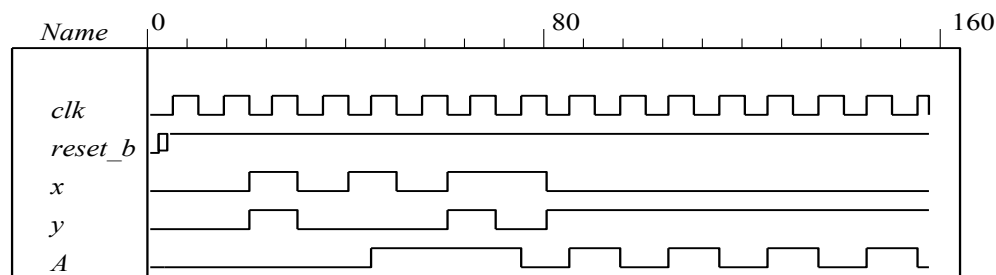
## 5.28 Verilog (a)

```
module Prob_5_28a (output A, input x, y, clk, reset_b);
    parameter s0 = 0, s1 = 1;
    reg state, next_state;
    assign A = state;
```

```
    always @ (posedge clk, negedge reset_b)
        if (reset_b == 0) state <= s0; else state <= next_state;
```

```
    always @ (state, x, y) begin
        next_state = s0;
        case (state)
            s0:    case ({x, y})
                    2'b00, 2'b11: next_state = s0;
                    2'b01, 2'b10: next_state = s1;
                endcase
            s1:    case ({x, y})
                    2'b00, 2'b11: next_state = s1;
                    2'b01, 2'b10: next_state = s0;
                endcase
        endcase
    end
endmodule
```

```
module t_Prob_5_28a ();
    wire A;
    reg x, y, clk, reset_b;
    Prob_5_28a M0 (A, x, y, clk, reset_b);
    initial #350 $finish;
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial fork
        #2 reset_b = 1;
        #3 reset_b = 0;    // Initialize to s0
        #4 reset_b = 1;
        x = 0; y = 0;
        #20 begin x = 1; y = 1; end
        #30 begin x = 0; y = 0; end
        #40 begin x = 1; y = 0; end
        #50 begin x = 0; y = 0; end
        #60 begin x = 1; y = 1; end
        #70 begin x = 1; y = 0; end
        #80 begin x = 0; y = 1; end
    join
endmodule
```



## VHDL

```
entity Prob_5_28a is
    port A: out Std_Logic; x, y, clk, reset_b: in Std_Logic);
end Prob_5_28a;
```

```
architecture Behavioral of Prob_5_28a is
```

```

type state_type is (S0, s1);
begin
    A <= state;
then
process (clk, reset_b)
    if (reset_b = '0') state <= s0;
    else if (clk'event and clk = '1') then state <= next_state;
    end if;
end process;

process (state, x, y)
begin
    next_state <= s0;
    case (state) is
        when s0 => case (x & y)
            when '00', '11' => next_state <= s0;
            when '01', '10' => next_state <= s1;
            end case;

        when s1 => case (x & y)
            when '00', '11' => next_state <= s1;
            when '01', '10' => next_state <= s0;
            end case;

    end case;
end Behavioral;

```

#### (b) Verilog

```

module Prob_5_28b (output A, input x, y, Clock, reset_b);
    xor (w1, x, y);
    xor (w2, w1, A);
    DFF M0 (A, w2, Clock, reset_b);
endmodule

```

```

module DFF (output reg Q, input D, Clock, reset_b);
    always @ (posedge Clock, negedge reset_b)
        if (reset_b == 0) Q <= 0;
        else Q <= D;
endmodule

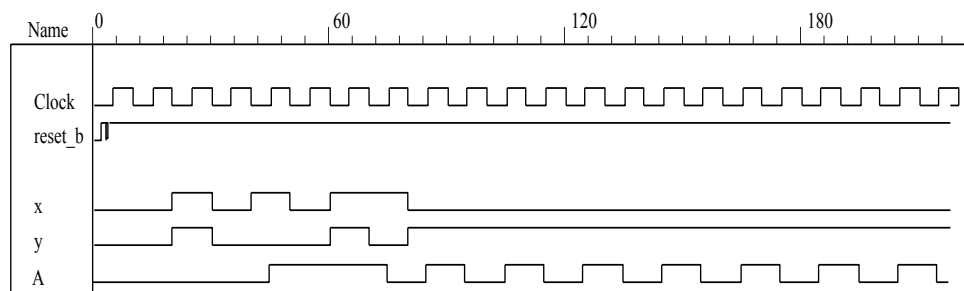
```

#### Testbench – Verilog

```

module t_Prob_5_28b ();
    wire A;
    reg x, y, clk, reset_b;
    Prob_5_28b M0 (A, x, y, clk, reset_b);
    initial #350 $finish;
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial fork
        #2 reset_b = 1;
        #3 reset_b = 0;           // Initialize to s0
        #4 reset_b = 1;
        x = 0; y = 0;
        #20 begin x = 1; y = 1; end
        #30 begin x = 0; y = 0; end
        #40 begin x = 1; y = 0; end
        #50 begin x = 0; y = 0; end
        #60 begin x = 1; y = 1; end
        #70 begin x = 1; y = 0; end
        #80 begin x = 0; y = 1; end
    join
endmodule

```



## VHDL

**entity** Prob\_5\_28b **is**

**port**(A: **out** Std\_Logic; x, y, Clock, reset\_b: **in** Std\_Logic);

**end** Prob\_5\_28b;

**architecture** Structural of Prob\_5\_28b **is**

**component** xor2\_gate **port** (y: **out** Std\_Logic; xin1, xin2: **in** Std\_Logic);

**component** DFF **port** (Q: **out** Std\_Logic; D, Clock, reset\_b: **in** Std\_Logic);

**begin**

G1: xor2\_gate **port map** (y => w1, x => xin1; y => xine2)

G2: xor2\_gate **port map** (y => w2, xin1 => w1, xin2 => A);

G3: DFF **port map** (Q => A, D => w2, Clock => Clock, reset\_b => reset\_b);

**end** Structural;

**entity** DFF **is**

**port** (Q: **out** Std\_Logic; D, Clock, reset\_b: **in** Std\_Logic);

**end** DFF;

**architecture** Behavioral of DFF **is**

**begin**

**process** (Clock, reset\_b)

**if** (reset\_b'event and = '0') Q <= 0;

**elsif** (Clock'event and Clock = '1') **then** Q <= D;

**end if**;

**end process**;

**end** Behavioral;

**entity** xor2\_gate **is**

**port** (y: **out** Std\_Logic; xin1, xin2: **in** Std\_Logic);

**end** xor2\_gate;

**architecture** Boolean\_eq of xor2gate **is**

**begin**

y <= xin1 **xor** xin2;

**end** Boolean\_eq;

## Testbench - VHDL

**entity** t\_Prob\_5\_28b () **is**;

**end** t\_Prob\_5.28b;

**architecture** TestBench of t\_Prob\_5\_28b **is**

**signal** t\_A, t\_x, t\_y, t\_clk, t\_reset\_b: Std\_Logic;

**signal** A;

**component** Prob\_5\_28b **port** (A: **out** Std\_Logic; x, y, Clock, reset\_b: **in** Std\_Logic);

**begin**

**M0:** Prob\_5\_28b M0 (t\_A =>A, t\_x => x, t\_y => y, t\_clk => clk, t\_reset\_b => reset\_b);

**process** – clock for testbench

**begin**

t\_clk <= '0';

**wait** for 5 ns;

t\_clk <= '1';

**wait** for 5 ns;

**end process;**

**process**

**begin**

t\_x <= '0';

t\_y <= '0'

t\_reset\_b = '1' after 2ns;

t\_reset\_b = '0' after 3ns; // Initialize to s0

t\_reset\_b = '1' after 4 ns;

t\_x <= '1' **after** 20 ns;

t\_y <= '1' **after** 20 ns;

t\_x <= '0' **after** 30 ns

t\_y <= '0' **after** 30 ns;

t\_x <= '1' after 40 ns;

t\_x <= '0' after 50 ns;

t\_x <= '1' **after** 60 ns;

t\_y <= '1' **after** 60 ns;

t\_y <= '0' **after** 70 ns;

t\_x <= '0' **after** 80 ns;

t\_y <= '1' **after** 80 ns;

**end process;**

**end TestBench;**

(C

(c) Use results of (b) and (c).

### Verilog

```
module t_Prob_5_28c ();
  wire A_a, A_b;
  reg x, y, clk, reset_b;
  Prob_5_28a M0 (A_a, x, y, clk, reset_b);
  Prob_5_28b M1 (A_b, x, y, clk, reset_b);

  initial #350 $finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork
    #2 reset_b = 1;
    #3 reset_b = 0;      // Initialize to s0
    #4 reset_b = 1;
    x = 0; y = 0;
    #20 begin x = 1; y = 1; end
    #30 begin x = 0; y = 0; end
    #40 begin x = 1; y = 0; end
    #50 begin x = 0; y = 0; end
    #60 begin x = 1; y = 1; end
    #70 begin x = 1; y = 0; end
    #80 begin x = 0; y = 1; end
  join
endmodule
```

### VHDL

```
entity t_Prob_5_28c ();
end t_Prob_5_28c;

architecture TestBench of t_Prob_5_28c is
  signal A_a, A_b;
  signal t_x, t_y, t_clk, t_reset_b;
  component Prob_5_28a port(A: out Std_Logic; x, y, Clock, reset_b: in Std_Logic);
  component Prob_5_28b port(A: out Std_Logic; x, y, Clock, reset_b: in Std_Logic);
begin
  M_a: Prob_5_28a (A_a => A, t_x => x, t_y => y, t_clk => clk, t_reset_b => reset_b);
  M_b: Prob_5_28b (A_b => A, t_x => x, t_y => y, t_clk => clk, t_reset_b => reset_b);
```

**process** – clock for testbench

```
begin
  t_clk <= '0';
  wait for 5 ns;
  t_clk <= '1';
  wait for 5 ns;
end process;
```

**process**

```
begin
  t_x <= '0';
  t_y <= '0';
  t_reset_b = '1' after 2ns;
  t_reset_b = '0' after 3ns;      // Initialize to s0
  t_reset_b = '1' after 4ns;

  t_x <= '1' after 20 ns;
  t_y <= '1' after 20 ns;

  t_x <= '0' after 30 ns;
  t_y <= '0' after 30 ns;

  t_x <= '1' after 40 ns;
```

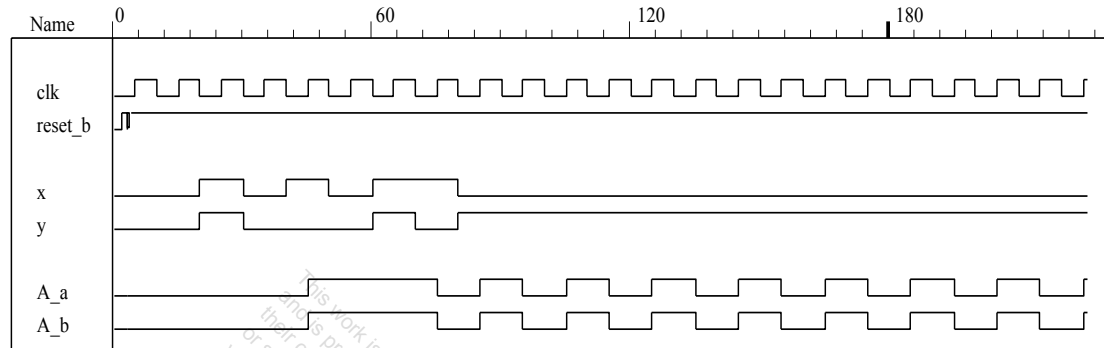
```

        t_x <= '0' after 50 ns;

        t_x <= '1' after 60 ns;
        t_y <= '1' after 60 ns;
        t_y <= '0' after 70 ns;

        t_x <= '0' after 80 ns;
        t_y <= '1' after 80 ns;
    end process;
end TestBench;

```



## 5.29

### Verilog

```

module Prob_5_29 (output reg y_out, input x_in, clock, reset_b);
    parameter s0 = 3'b000, s1 = 3'b001, s2 = 3'b010, s3 = 3'b011, s4 = 3'b100;
    reg [2: 0] state, next_state;

    always @ (posedge clock, negedge reset_b)
        if (reset_b == 0) state <= s0;
        else state <= next_state;

    always @ (state, x_in) begin
        y_out = 0;
        next_state = s0;
        case (state)
            s0:    if (x_in) begin next_state = s4; y_out = 1; end else begin next_state = s3; y_out = 0; end
            s1:    if (x_in) begin next_state = s4; y_out = 1; end else begin next_state = s1; y_out = 0; end
            s2:    if (x_in) begin next_state = s0; y_out = 1; end else begin next_state = s2; y_out = 0; end
            s3:    if (x_in) begin next_state = s2; y_out = 1; end else begin next_state = s1; y_out = 0; end
            s4:    if (x_in) begin next_state = s3; y_out = 0; end else begin next_state = s2; y_out = 0; end
            default: next_state = 3'bxxx;
        endcase
    end
endmodule

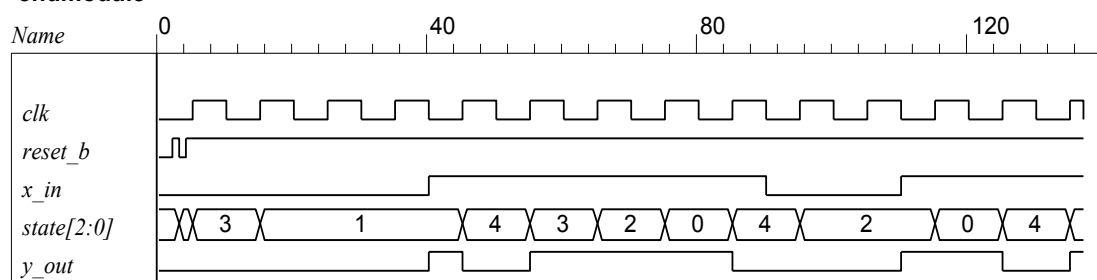
module t_Prob_5_29 ();
    wire y_out;
    reg x_in, clk, reset_b;

    Prob_5_29 M0 (y_out, x_in, clk, reset_b);

    initial #350$finish;
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial fork
        #2 reset_b = 1;
        #3 reset_b = 0;    // Initialize to s0
        #4 reset_b = 1;

        // Trace the state diagram and monitor y_out
        x_in = 0;          // Drive from s0 to s3 to S1 and park
        #40 x_in = 1;      // Drive to s4 to s3 to s2 to s0 to s4 and loop
        #90 x_in = 0;      // Drive from s0 to s3 to s2 and part
        #110 x_in = 1;     // Drive s0 to s4 etc
    join
endmodule

```



### VHDL

```

entity Prob_5_29
    port (y_out: out Std_Logic; x_in, clock, reset_b: in Std_Logic);
end Prob_5_29;

    architecture Behavioral of Prob_5_29 is;

```



```

type state_type is integer range 0 to 4;
state_type state, next_state;
constant s0 = 1, s1 = 1, s2 = 2; s3 = 3; s4 = 4;

process (clock, reset_b)
begin
    if (reset_b'event and reset_b = '0') then state <= s0;
    elsif (clock
'event and clock = '1') then state <= next_state;
    end if;

    process (state, x_in) begin
        y_out <= '0';
        next_state <= s0;
        case (state)
        when s0 => if (x_in) next_state <= s4; y_out <= '1'; else next_state <= s3; y_out <= '0'; end if;
        when s1 => if (x_in) next_state <= s4; y_out <= '1'; else next_state <= s1; y_out <= '0'; end if;
        when s2 => if (x_in) next_state <= s0; y_out <= '1'; else next_state <= s2; y_out <= '0'; end if;
        when s3 => if (x_in) next_state <= s2; y_out <= '1'; else next_state <= s1; y_out <= '0'; end if;
        when s4 => if (x_in) next_state <= s3; y_out <= '0'; else next_state <= s2; y_out <= '0'; end if;
        default: next_state = s0;
        endcase;
    end
end Behavioral;

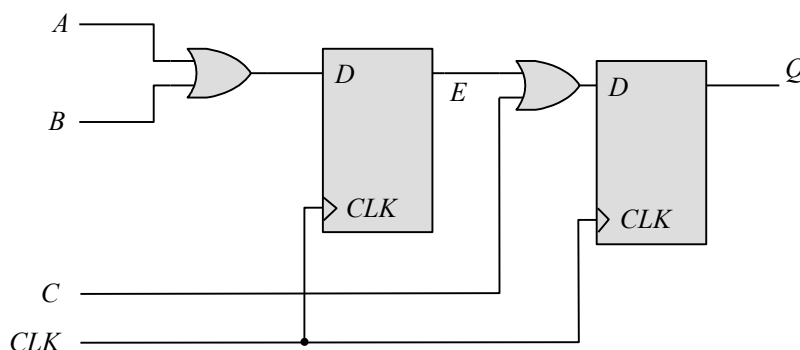
module t_Prob_5_29 ();
    signal t_y_out;
    signal t_x_in, t_clk, t_reset_b;
    component Prob_5_29 port (y_out: out Std_Logic; x_in, clk, reset_b: in Std_Logic);
begin
    M0: Prob_5_29 port map (t_y_out => y_out; t_x_in => x_in, t_clk => clk, t_reset_b => reset_b);

    process clk = 0; forever #5 clk = ~clk; end
    initial fork
        #2 reset_b = '1';
        #3 reset_b = '0';           // Initialize to s0
        #4 reset_b = '1';

        x_in = '0';                // Trace the state diagram and monitor y_out
        #40 x_in = '1';            // Drive from s0 to s3 to S1 and park
        #90 x_in = '0';            // Drive to s4 to s3 to s2 to s0 to s4 and loop
        #110 x_in = '1';           // Drive from s0 to s3 to s2 and part
                                   // Drive s0 to s4 etc
    join
endmodule

```

### 5.30



### 5.31 (a)

```

module Seq_Ckt (input A, B, C, CLK, output reg Q);
    reg E;
    always @ (posedge CLK)
    begin
        Q = E & C;
        E = A | B;
    end
endmodule

```

### (b)

```

process (CLK) begin
    if CLK'event and CLK = '1' then
        Q := E and C;
        E := A or B;
    end if;
end process;

```

Note: The statements must be written in an order than produces the effect of concurrent assignments.

## 5.32 Verilog

```
initial begin
    enable = 0; A = 1; B = 0; C = 0; D = 1; E = 1; F = 1;
    #10 A = 0; B = 1; C = 1;
    #10 A = 1; B = 0; D = 1; E = 0;
    #10 B = 1; E = 1; F = 0;
    #10 enable = 1;
        B = 0; D = 0; F = 1;
    #10 B = 1;
    #10 B = 0; D = 1;
    #10 B = 1;
end
```

```
initial fork
    enable = 0; A = 1; B = 0; C = 0; D = 1; E = 1; F = 1;
    #10 begin A = 0; B = 1; end
    #20 begin A = 1; B = 0; D = 1; E = 0; end
    #30 begin B = 1; E = 1; F = 0; end
    #40 begin B = 0; D = 0; F = 1; end
    #50 begin B = 1; end
    #60 begin B = 0; D = 1; end
    #70 begin B = 1; end
join
```

## VHDL

```
process ( )
    enable <= '0'; A <= '1'; B <= '0'; C <= '0'; D <= '0'; E <= '1'; F <= '1';
    wait for 10 ns;
    a <= '0'; B <= '1'; C <= '1';
    wait for 10 ns;
    A <= '1'; B <= '0'; D <= '1'; E <= '0'; B <= '1'; E <= '1'; F <= '0';
    wait for 10 ns;
    enable <= '1';
    B <= '0'; D <= 0; F <= '1';
    wait for 10 ns;
    B <= '1';
    wait for 10 ns;
    B <= '0';
    D <= '1';
    wait for 10 ns;
    B <= '1';
end process;
```

- 5.33** Signal transitions that are caused by input signals that change on the active edge of the clock race with the clock itself to reach the affected flip-flops, and the outcome is indeterminate (unpredictable). Conversely, changes caused by inputs that are synchronized to the inactive edge of the clock reach stability before the active edge, with predictable outputs of the flip-flops that are affected by the inputs.

### 5.34 Verilog

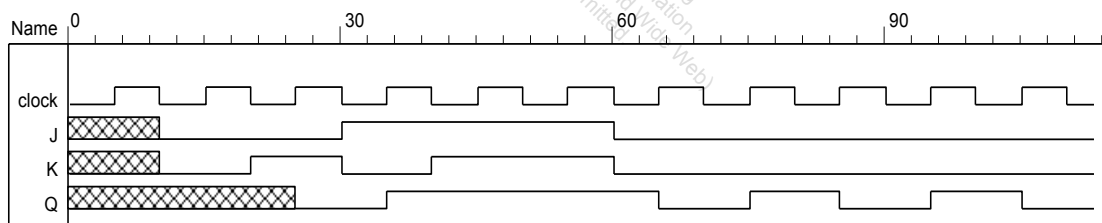
```
module JK_flop_Prob_5_34 (output Q, input J, K, clk);
    wire K_bar;
    D_flop M0 (Q, D, clk);
    Mux M1 (D, J, K_bar, Q);
    Inverter M2 (K_bar, K);
endmodule
```

```
module D_flop (output reg Q, input D, clk);
    always @ (posedge clk) Q <= D;
endmodule
```

```
module Inverter (output y_bar, input y);
    assign y_bar = ~y;
endmodule
```

```
module Mux (output y, input a, b, select);
    assign y = select ? a : b;
endmodule
```

```
module t_JK_flop_Prob_5_34 ();
    wire Q;
    reg J, K, clock;
    JK_flop_Prob_5_34 M0 (Q, J, K, clock);
    initial #500 $finish;
    initial begin clock = 0; forever #5 clock = ~clock; end
    initial fork
        #10 begin J = 0; K = 0; end // toggle Q unknown
        #20 begin J = 0; K = 1; end // set Q to 0
        #30 begin J = 1; K = 0; end // set q to 1
        #40 begin J = 1; K = 1; end // no change
        #60 begin J = 0; K = 0; end // toggle Q
    join
endmodule
```



### VHDL

```
entity JK_flop_Prob_5_34
    port (output Q: out Std_Logic; J, K, clk: in Std_Logic);
end JK_flop_Prob_5_34;
```

```
architecture Structural of JK_flop_Problem_5_34 is
    signal K_bar;
    component D_flop port (Q: out Std_Logic; D, clk: in Std_Logic);
    component Mux port (y: out Std_Logic; a, b: in Std_Logic; select: in Std_Logic);
    component Inverter port (y: out Std_Logic; in xin Std_Logic);
begin
    M0: D_flop (Q => Q, D => D, clk => clk);
    M1: Mux (y => y, a => J, b => K_bar, select => Q);
    Inverter M2 (y_bar => K_bar, y => K);
end
```

**end Structural**

```
entity D_flop is
  port (Q: out Std_Logic; D, clk: in Std_Logic);
end D_flop;
```

```
architecture Behavioral
  process (clk, D);
  begin
    if (clk'event and clk = '1') then Q <= D;
  end Behavioral;
```

```
entity Inverter is
  port (y_bar: out Std_Logic; y: in Std_Logic);
end Inverter;
```

```
architecture Boolean_Eq of Inverter is
  y_bar <= not y;
endmodule
```

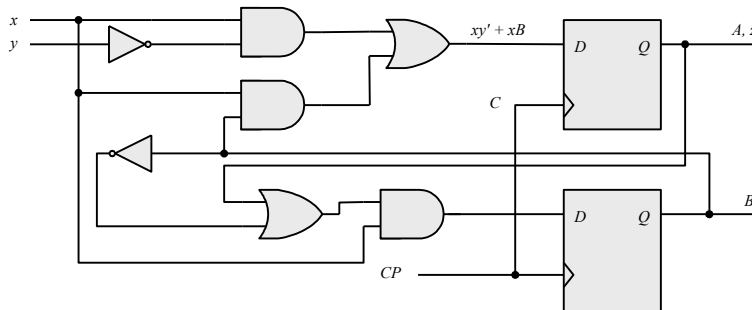
```
entity Mux is
  port (y: out Std_Logic; a, b, select: in Std_Logic);
end Mux;
```

```
architecture Behavior of Mux is
```

```
begin
  process (a, b, select)
  begin
    if select = '0' then y <= a;
    elsif select = '1' then y <= b;
    else y <= 'x';
    end if;
  end process;
end Behavior;
```

### 5.35

From Problem 5.6:



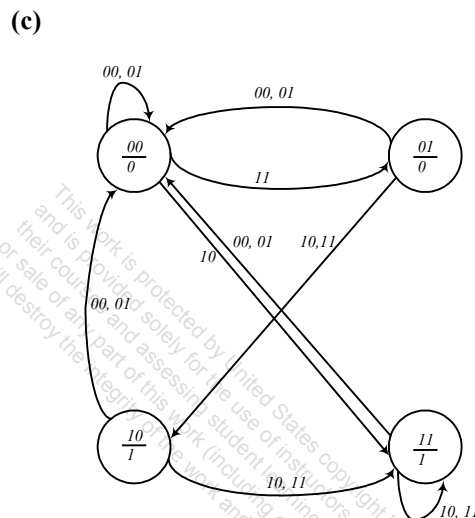
(b)

$$A(t+1) = xy' + xB$$

$$B(t+1) = xA + xB'$$

$$z = A$$

Present state		Inputs		Next state		Output
A	B	x	y	A	B	z
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1



### Verilog

```

module Prob_5_35 (output out_z, input in_x, in_y, clk, reset_b);
    reg [1:0] state, next_state;
    assign out_z = ((state == 2'b10) || (state == 2'b11));

    always @ (posedge clk) if (reset_b == 1'b0) state <= 2'b00; else state <= next_state;

    always @ (state, in_x, in_y) begin
        next_state = 2'b00;
        case (state)
            2'b00: if (((in_x, in_y) == 2'b00) || ((in_x, in_y) == 2'b01)) next_state = 2'b00;
                   else if ((in_x, in_y) == 2'b10) next_state = 2'b11;
                   else if ((in_x, in_y) == 2'b11) next_state = 2'b01;

            2'b01: if (((in_x, in_y) == 2'b00) || ((in_x, in_y) == 2'b01)) next_state = 2'b00;
                   else if (((in_x, in_y) == 2'b10) || ((in_x, in_y) == 2'b11)) next_state = 2'b10;

            2'b10: if (((in_x, in_y) == 2'b00) || ((in_x, in_y) == 2'b01)) next_state = 2'b00;
                   else if (((in_x, in_y) == 2'b10) || ((in_x, in_y) == 2'b11)) next_state = 2'b11;

            2'b11: if (((in_x, in_y) == 2'b00) || ((in_x, in_y) == 2'b01)) next_state = 2'b00;
                   else if (((in_x, in_y) == 2'b10) || ((in_x, in_y) == 2'b11)) next_state = 2'b11;
            default: next_state = 2'b00;
        endcase
    end
endmodule

```

```

module t_Prob_5_35 ();
  wire out_z;
  reg in_x, in_y, clk, reset_b;

  Prob_5_35 M0 (out_z, in_x, in_y, clk, reset_b);

  initial #250 $finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork
    reset_b = 0;
    #20 reset_b = 1;

    #50 {in_x, in_y} = 2'b00;      // Remain in 2'b00
    #60 {in_x, in_y} = 2'b01;      // Remain in 2'b01
    #70 {in_x, in_y} = 2'b11;      // Transition to 2'b01
    #90 {in_x, in_y} = 2'b00;      // Transition to 2'b00
    #110 {in_x, in_y} = 2'b11;     // Transition to 2'b01
    #120 {in_x, in_y} = 2'b01;     // Transition to 2'b00

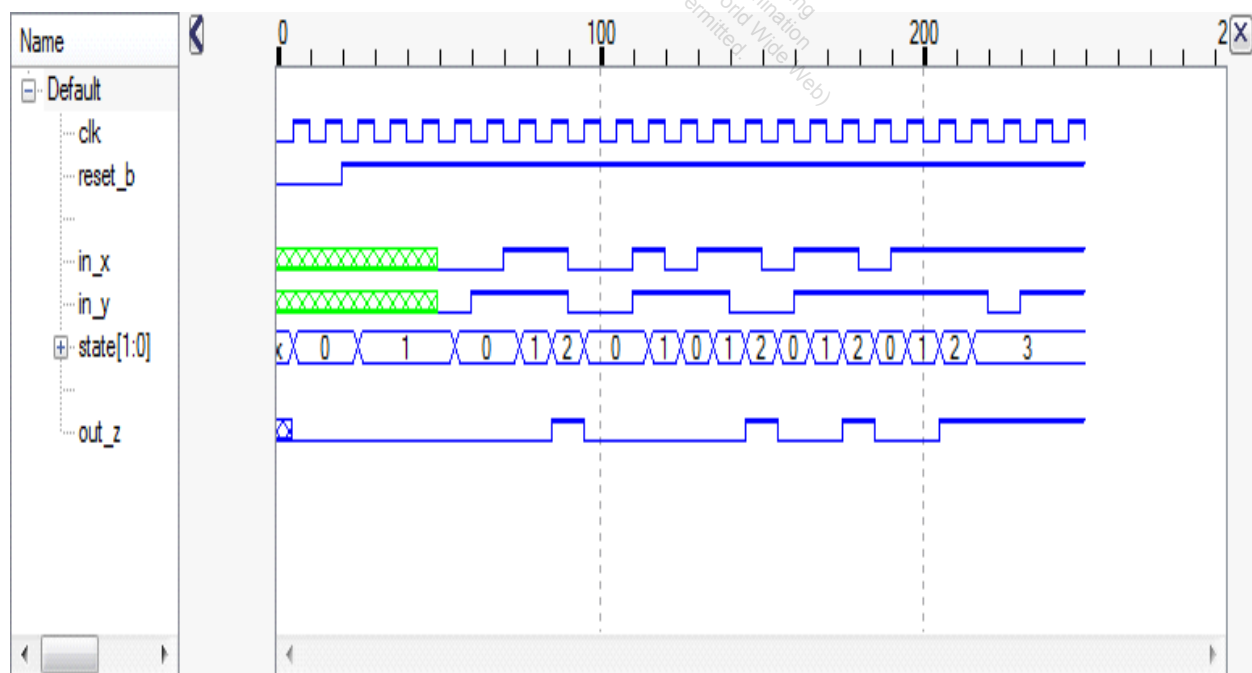
    #130 {in_x, in_y} = 2'b11;     // Transition to 2'b01
    #140 {in_x, in_y} = 2'b10;     // Transition to 2'b10
    #150 {in_x, in_y} = 2'b00;     // Transition to 2'b00
    #160 {in_x, in_y} = 2'b11;     // Transition to 2'b01

    #170 {in_x, in_y} = 2'b11;     // Transition to 2'b10
    #180 {in_x, in_y} = 2'b01;     // Transition to 2'b00

    #190 {in_x, in_y} = 2'b11;     // Transition to 2'b01
    #200 {in_x, in_y} = 2'b11;     // Transition to 2'b10
    #210 {in_x, in_y} = 2'b11;     // Transition to 2'b11

    #220 {in_x, in_y} = 2'b10;     // Remain in 2'b11
    #230 {in_x, in_y} = 2'b11;     // Remain in 2'b11
  join
endmodule

```



VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
entity Prob_5_35_vhdl is
  port (out_z: out Bit; in_x, in_y: in Boolean; clk, reset_b: in Bit);
end Prob_5_35_vhdl;
```

```
architecture Behavioral of Prob_5_35_vhdl is
```

```
  signal state, next_state: Std_Logic_Vector (1 downto 0);
```

```
begin
```

```
  process (state) begin
```

```
    out_z <= '0';
```

```
    if (state = "10" or state = "11") then out_z <= '1';
```

```
    end if;
```

```
  end process;
```

```
  process (clk, reset_b) begin
```

```
    if (reset_b = '0') then state <= "00";
```

```
    elsif (clk'event and clk = '1') then state <= next_state; end if;
```

```
  end process;
```

```
  process (state, in_x, in_y) begin
```

```
    next_state <= "00";
```

```
    case (state) is
```

```
      when "00" => if ((not in_x and not in_y) or (not in_x and in_y)) then next_state <= '0' & '0';
                   elsif ((not in_x and not in_y) or (in_x and not in_y)) then next_state <= ('0' & '0');
                   elsif (in_x and not in_y) then next_state <= "11";
                   elsif (in_x and in_y) then next_state <= "01";
                   end if;
```

```
      when "01" => if ((not in_x and not in_y) or (not in_x and in_y)) then next_state <= "00";
                   elsif ((in_x and not in_y) or (in_x and in_y)) then next_state <= "10";
                   end if;
```

```
      when "10" => if ((not in_x and not in_y) or (not in_x and in_y)) then next_state <= "00";
                   elsif ((in_x and not in_y) or (in_x and in_y)) then next_state <= "11";
                   end if;
```

```
      when "11"=> if ((not in_x and not in_y) or (not in_x and in_y)) then next_state <= "00";
                   elsif ((in_x and not in_y) or (in_x and in_y)) then next_state <= "11";
                   end if;
```

```
      when others => next_state <= "00";
```

```
    end case;
```

```
  end process;
```

```
end Behavioral;
```

**5.36** Note: See Problem 5.8 (counter with repeated sequence: (A, B) = 00, 01, 10, 00 ....

// See Fig. P5.8



## VERILOG

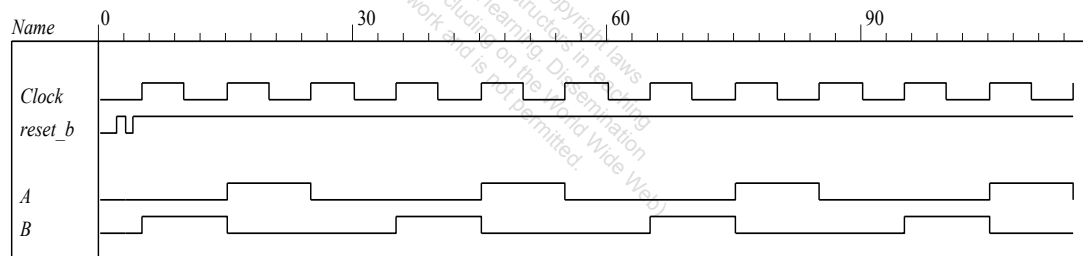
```
module Problem_5_36 (output A, B, input Clock, reset_b);
  wire T_A, T_B, A_b, B_b;
  or (T_A, A, B);
  or (T_B, A_b, B);
  T_flop M0 (A, A_b, T_A, Clock, reset_b);
  T_flop M1 (B, B_b, T_B, Clock, reset_b);
endmodule
```

```
module T_flop (output reg Q, output QB, input T, Clock, reset_b);
  assign QB = ~ Q;
  always @ (posedge Clock, negedge reset_b)
    if (reset_b == 0) Q <= 0;
    else if (T) Q <= !Q;
endmodule
```

```
module t_Problem_5_36 ();
  wire A, B;
  reg Clock, reset_b;

  Problem_5_36 M0 (A, B, Clock, reset_b);

  initial #350$finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end
  initial fork
    #2 reset_b = 1;
    #3 reset_b = 0;
    #4 reset_b = 1;
  join
endmodule
```



## VHDL

```
entity Problem_5_36_vhdl (output A, B: out bit; input Clock, reset_b: out bit) is
end Problem_5_36_vhdl;
```

architecture Structural of Problem\_5\_36\_vhdl is

```
  component or2_gate port (y: out, bit; xin1, xin2: in bit);
  component T_flop port (Q, Q_b: in bit, T, Clock, reset_b: in bit);
```

begin

```
  G1: or2_gate port map (y => T_A, xin1 => A, xin2 => B);
  G2: or2_gate port map (y => T_B, xin1 => A_b, xin2 => B);
  G3: T_flop port map (Q => A, Q_b => A_b, T => T_A, Clock => Clock, reset_b => reset_b);
  G4: T_flop port map (Q => B, Q_b => B_b, T => T_B, Clock => Clock, reset_b => reset_b);
```

end Structural;

entity T\_flop is

```
  port (Q, QB: out bit; T, Clock, reset_b: in bit);
```

end T\_flop;

```

architecture Behavioral of T_flop is
    QB <= not Q;
    process (Clock, reset_b) is
        begin
            if (reset_b '0') then Q <= 0;
            elsif (T) Q <= Q;
        end process;
    end Behavioral;

module t_Problem_5_36 ();
    wire A, B;
    reg Clock, reset_b;

    Problem_5_36 M0 (A, B, Clock, reset_b);

    initial #350 $finish;
    initial begin Clock = 0; forever #5 Clock = ~Clock; end
    initial fork
        #2 reset_b = 1;
        #3 reset_b = 0;
        #4 reset_b = 1;
    join
endmodule

```

### 5.37 Verilog

```

module Prob_5_37_Fig_5_25 (output reg y, input x_in, clock, reset_b);

    parameter a = 3'b000, b = 3'b001, c = 3'b010, d = 3'b011, e = 3'b100, f = 3'b101, g = 3'b110;
    reg [2:0] state, next_state;

    always @ (posedge clock, negedge reset_b)
        if (reset_b == 0) state <= a;
        else state <= next_state;

    always @ (state, x_in) begin
        y = 0;
        next_state = a;
        case (state)
            a: begin y = 0; if (x_in == 0) next_state = a; else next_state = b; end

            b: begin y = 0; if (x_in == 0) next_state = c; else next_state = d; end

            c: begin y = 0; if (x_in == 0) next_state = a; else next_state = d; end

            d: if (x_in == 0) begin y = 0; next_state = e; end
                else begin y = 1; next_state = f; end

            e: if (x_in == 0) begin y = 0; next_state = a; end
                else begin y = 1; next_state = f; end

            f: if (x_in == 0) begin y = 0; next_state = g; end
                else begin y = 1; next_state = f; end

            g: if (x_in == 0) begin y = 0; next_state = a; end
                else begin y = 1; next_state = f; end

            default: next_state = a;
        endcase
    end
endmodule

```

```

module Prob_5_37_Fig_5_26 (output reg y, input x_in, clock, reset_b);
  parameter a = 3'b000, b = 3'b001, c = 3'b010, d = 3'b011, e = 3'b100;
  reg [2: 0] state, next_state;

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= a;
    else state <= next_state;
  always @ (state, x_in) begin
    y = 0;
    next_state = a;
    case (state)
      a:    begin y = 0; if (x_in == 0) next_state = a; else next_state = b; end

      b:    begin y = 0; if (x_in == 0) next_state = c; else next_state = d; end

      c:    begin y = 0; if (x_in == 0) next_state = a; else next_state = d; end

      d:    if (x_in == 0) begin y = 0; next_state = e; end
            else begin y = 1; next_state = d; end

      e:    if (x_in == 0) begin y = 0; next_state = a; end
            else begin y = 1; next_state = d; end

      default: next_state = a;
    endcase
  end
endmodule

```

```

module t_Problem_5_37 ();
  wire y_Fig_5_25, y_Fig_5_26;
  reg x_in, clock, reset_b;

  Problem_5_37_Fig_5_25 M0 (y_Fig_5_25, x_in, clock, reset_b);
  Problem_5_37_Fig_5_26 M1 (y_Fig_5_26, x_in, clock, reset_b);

  wire [2: 0] state_25 = M0.state;
  wire [2: 0] state_26 = M1.state;

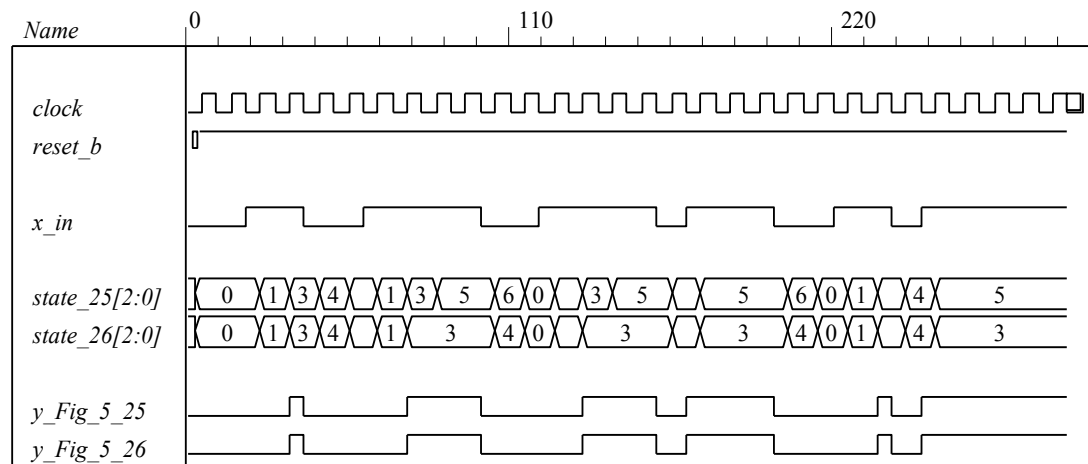
  initial #350 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    x_in = 0;
    #2 reset_b = 1;
    #3 reset_b = 0;
    #4 reset_b = 1;
    #20 x_in = 1;
    #40 x_in = 0; // abdea, abdea

    #60 x_in = 1;
    #100 x_in = 0; // abdf....fga, abd ... dea

    #120 x_in = 1;
    #160 x_in = 0;
    #170 x_in = 1;
    #200 x_in = 0; // abdf....fgf...fga, abd ...ded...ea

    #220 x_in = 1;
    #240 x_in = 0;
    #250 x_in = 1; // abdef... // abded...
  join
endmodule

```



## VHDL

```
entity Prob_5_37_Fig_5_25_vhdl is  
  port (y: out bit; xin, clock, reset_b: out bit);  
end Prob_5_37_Fig_5_25_vhdl;
```

**architecture Behavioral of Prob\_5\_37\_Fig\_5\_25\_vhdl is**

```

constant a: Bit_Vector (2 downto 0) := "000";
constant b: Bit_Vector (2 downto 0) := "001";
constant c: Bit_Vector (2 downto 0) := "010";
constant d: Bit_Vector (2 downto 0) := "011";
constant e: Bit_Vector (2 downto 0) := "100";
constant f: Bit_Vector (2 downto 0) := "101";
constant g: Bit_Vector (2 downto 0) := "110";
signal state, next_state: Bit_Vector (2 downto 0);
begin

```

```

begin
    process (clock, reset_b) begin
        if (reset_b = '0') then state <= a;
        elsif (clock'event and clock = '1') then state <= next_state;
        end if;
    end process;

    process (state, x_in) begin
        y <= '0';
        next_state <= a;
        case (state) is
            when a => y <= '0'; if (x_in = '0') then next_state <= a; else next_state <= b; end if;

            when b => y <= '0'; if (x_in = '0') then next_state <= c; else next_state <= d; end if;

            when c => y <= '0'; if (x_in = '0') then next_state <= a; else next_state <= d; end if;

            when d => if (x_in = '0') then y <= '0'; next_state <= e;
                        else y <= '1'; next_state <= f; end if;

            when e => if (x_in = '0') then y <= '0'; next_state <= a;
                        else y <= '1'; next_state <= f; end if;

            when f => if (x_in = '0') then y <= '0'; next_state <= g;
                        else y <= '1'; next_state <= f; end if;
        end case;
    end process;
end

```

```

when g =>    if (x_in = '0') then y <= '0'; next_state <= a;
                else y <= '1'; next_state <= f; end if;

when others => next_state <= a;
end case;
end process;
end Behavioral;

entity Prob_5_37_Fig_5_26_vhdl is
    port (y: out bit; x_in, clock, reset_b: in bit);
end Prob_5_37_Fig_5_26_vhdl;

architecture Behavioral of Prob_5_37_Fig_5_26_vhdl is
    constant a: Bit_Vector := "000";
    constant b: Bit_Vector := "001";
    constant c: Bit_Vector := "010";
    constant d: Bit_Vector := "011";
    constant e: Bit_Vector := "100";
    signal state, next_state: Bit_Vector (2 downto 0);
begin

    process (clock, reset_b) begin
        if reset_b = '0' then state <= a;
        elsif clock'event and clock = '1' then state <= next_state; end if;
    end process;

    process (state, x_in) begin
        y <= '0';
        next_state <= a;
        case (state) is
            when a =>    y <= '0'; if x_in = '0' then next_state <= a; else next_state <= b; end if;

            when b =>    y <= '0'; if x_in = '0' then next_state <= c; else next_state <= d; end if;

            when c =>    y <= '0'; if x_in = '0' then next_state <= a; else next_state <= d; end if;

            when d =>    if x_in = '0' then y <= '0'; next_state <= e;
                        else y <= '1'; next_state <= d; end if;

            when e =>    if x_in = '0' then y <= '0'; next_state <= a;
                        else y <= '1'; next_state <= d; end if;

            when others => next_state <= a;
        end case;
    end process;
end Behavioral;

```

### 5.38 (a) Verilog

```

module Prob_5_38a (input x_in, clock, reset_b);
    parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;
    reg [1: 0] state, next_state;

    always @ (posedge clock, negedge reset_b)
        if (reset_b == 0) state <= s0;
        else state <= next_state;

```

```

always @ (state, x_in) begin
    next_state = s0;
case (state)
    s0:    if (x_in == 0) next_state = s0;
           else if (x_in == 1) next_state = s3;

    s1:    if (x_in == 0) next_state = s1;
           else if (x_in == 1) next_state = s2;

    s2:    if (x_in == 0) next_state = s2;
           else if (x_in == 1) next_state = s0;

    s3:    if (x_in == 0) next_state = s3;
           else if (x_in == 1) next_state = s1;
    default:    next_state = s0;
endcase
end
endmodule

```

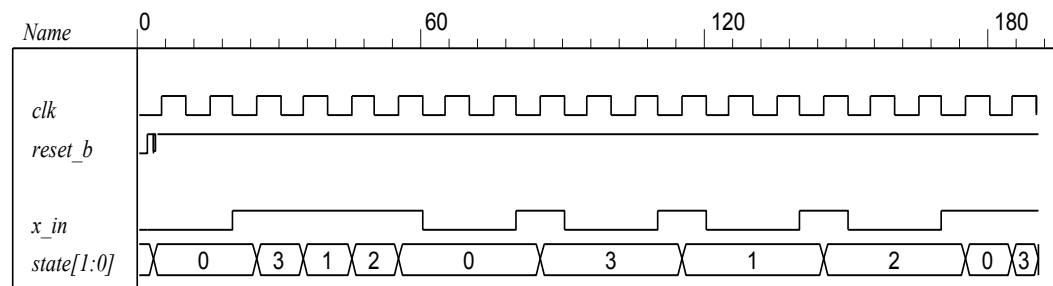
```

module t_Prob_5_38a ();
    reg x_in, clk, reset_b;

    Prob_5_38a M0 ( x_in, clk, reset_b);

    initial #350 $finish;
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial fork
        #2 reset_b = 1;
        #3 reset_b = 0;           // Initialize to s0
        #4 reset_b = 1;
        #2 x_in = 0;
        #20 x_in = 1;
        #60 x_in = 0;
        #80 x_in = 1;
        #90 x_in = 0;
        #110 x_in = 1;
        #120 x_in = 0;
        #140 x_in = 1;
        #150 x_in = 0;
        #170 x_in = 1;
    join
endmodule

```



### VHDL

```

entity Prob_5_38a_vhdl is
    port (x_in, clock, reset_b: in bit);
end Prob_5_38a_vhdl;

```

```

architecture Behavioral of Prob_5_38a_vhdl is
  constant s0: bit_vector (1 downto 0) := "00";
  constant s1: bit_vector (1 downto 0) := "01";
  constant s2: bit_vector (1 downto 0) := "10";
  constant s3: bit_vector (1 downto 0) := "11";
  signal state, next_state: bit_vector (1 downto 0);
begin
  process (clock, reset_b) begin
    if (reset_b = '0') then state <= s0;
    else state <= next_state; end if;
  end process;

  process (state, x_in) begin
    next_state <= s0;
    case (state) is
      when s0 => if x_in = '0' then next_state <= s0;
        elsif x_in = '1' then next_state <= s3; end if;

      when s1 => if x_in = '0' then next_state <= s1;
        elsif x_in = '1' then next_state <= s2; end if;

      when s2 => if x_in = '0' then next_state <= s2;
        elsif x_in = '1' then next_state <= s0; end if;

      when s3 => if x_in = '0' then next_state <= s3;
        elsif x_in = '1' then next_state <= s1; end if;

      when others => next_state <= s0;
    end case;
  end process;
end Behavioral;

```

(b)

### Verilog

```

module Prob_5_38b (input x_in, clock, reset_b);
  parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;
  reg [1: 0] state, next_state;

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= s0;
    else state <= next_state;

  always @ (state, x_in) begin
    next_state = s0;
    case (state)
      s0: if (x_in == 0) next_state = s0;
        else if (x_in == 1) next_state = s3;

      s1: if (x_in == 0) next_state = s1;
        else if (x_in == 1) next_state = s2;

      s2: if (x_in == 0) next_state = s2;
        else if (x_in == 1) next_state = s0;

      s3: if (x_in == 0) next_state = s3;
        else if (x_in == 1) next_state = s1;
      default: next_state = s0;
    endcase
  end

```

```

end
endmodule

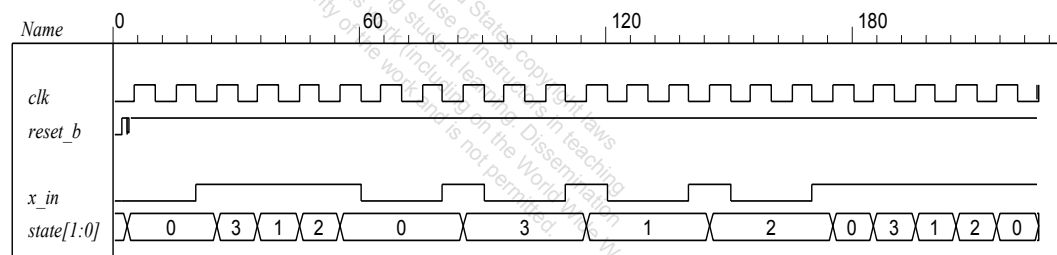
module t_Prob_5_38b ();

reg x_in, clk, reset_b;

Prob_5_38b M0 ( x_in, clk, reset_b);

initial #350$finish;
initial begin clk = 0; forever #5 clk = ~clk; end
initial fork
    #2 reset_b = 1;
    #3 reset_b = 0;    // Initialize to s0
    #4 reset_b = 1;
    #2 x_in = 0;
    #20 x_in = 1;
    #60 x_in = 0;
    #80 x_in = 1;
    #90 x_in = 0;
    #110 x_in = 1;
    #120 x_in = 0;
    #140 x_in = 1;
    #150 x_in = 0;
    #170 x_in = 1;
join
endmodule

```



## VHDL

```

entity Prob_5_38b_vhdl is
    port (x_in, clock, reset_b: in bit);
end Prob_5_38b_vhdl;

architecture Behavioral of Prob_5_38b_vhdl is
    constant s0: bit_vector (1 downto 0) := "00";
    constant s1: bit_vector (1 downto 0) := "01";
    constant s2: bit_vector (1 downto 0) := "10";
    constant s3: bit_vector (1 downto 0) := "11";
    signal state, next_state: bit_vector (1 downto 0);
begin
    process (clock, reset_b) begin
        if (reset_b = '0') then state <= s0;
        elsif clock'event and clock = '1' then state <= next_state;
        end if;
    end process;

    process (state, x_in) begin
        next_state <= s0;
        case (state) is

```



```

    when s0 => if x_in = '0' then next_state <= s0;
               elsif x_in = '1' then next_state <= s3; end if;

    when s1 => if x_in = '0' then next_state <= s1;
               elsif x_in = '1' then next_state <= s2; end if;

    when s2 => if x_in = '0' then next_state <= s2;
               elsif x_in = '1' then next_state <= s0; end if;

    when s3 => if x_in = '0' then next_state <= s3;
               elsif x_in = '1' then next_state <= s1; end if;

    when others => next_state = s0;
end case;
end process;
end Behavioral;

```

### 5.39

#### Verilog

```

module Serial_2s_Comp (output reg B_out, input B_in, clk, reset_b);
// See problem 5.17
parameter S_0 = 1'b0, S_1 = 1'b1;
reg state, next_state;
always @ (posedge clk, negedge reset_b) begin
    if (reset_b == 0) state <= S_0;
    else state <= next_state;
end

always @ (state, B_in) begin
    B_out = 0;
    case (state)
        S_0: if (B_in == 0) begin next_state = S_0; B_out = 0; end
            else if (B_in == 1) begin next_state = S_1; B_out = 1; end

        S_1: begin next_state = S_1; B_out = ~B_in; end
        default: next_state = S_0;

    endcase
end
endmodule

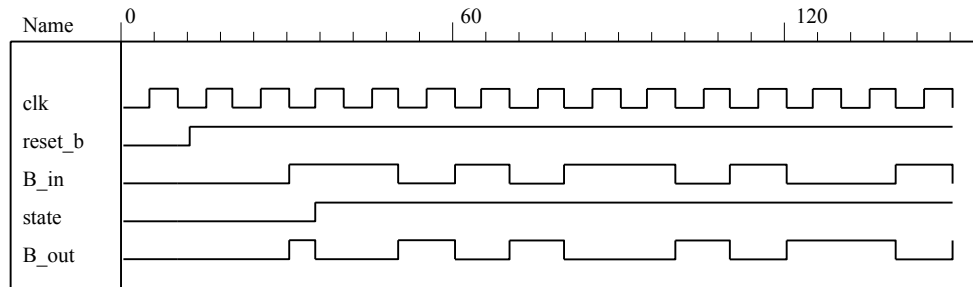
module t_Serial_2s_Comp ();
wire B_in, B_out;
reg clk, reset_b;
reg [15: 0] data;
assign B_in = data[0];

always @ (negedge clk, negedge reset_b)
    if (reset_b == 0) data <= 16'ha5ac; else data <= data >> 1; // Sample bit stream

Serial_2s_Comp M0 (B_out, B_in, clk, reset_b);

initial #150 $finish;
initial begin clk = 0; forever #5 clk = ~clk; end
initial fork
    #10 reset_b = 0;
    #12 reset_b = 1;
join
endmodule

```



### VHDL

// See problem 5.17

```
entity Prob_5_39_Serial_2s_Comp_vhdl is
    port (B_out: out bit; B_in, clk, reset_b: in bit);
end Prob_5_39_Serial_2s_Comp_vhdl;
```

```
architecture Behavioral of Prob_5_39_Serial_2s_Comp_vhdl is
```

```
    constant S_0: bit := '0';
    constant S_1: bit := '1';
    signal state, next_state: bit;
```

```
begin
```

```
    process (clk, reset_b) begin
        if reset_b = '0' then state <= S_0;
        elsif clock'event and clock = '1' then state <= next_state; end if;
    end process;
```

```
    process (state, B_in) begin
```

```
        B_out <= '0';
        case (state) is
            when S_0 => if B_in = '0' then next_state <= S_0; B_out <= '0';
            elsif B_in = '1' then next_state <= S_1; B_out <= '1'; end if;
```

```
            when S_1 => next_state <= S_1; B_out <= not B_in;
```

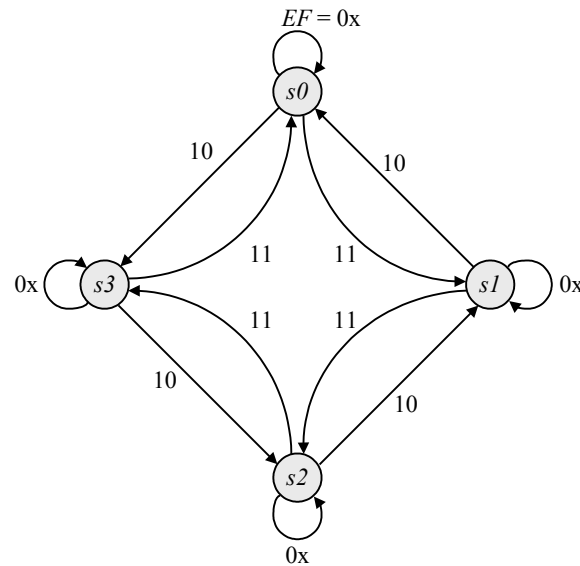
```
            when others => next_state <= S_0;
```

```
        end case;
```

```
    end process;
```

```
end Behavioral;
```

## 5.40



### Verilog

```

module Prob_5_40 (input E, F, clock, reset_b);
  parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;
  reg [1: 0] state, next_state;

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= s0;
    else state <= next_state;

  always @ (state, E, F) begin
    next_state = s0;
    case (state)
      s0:   if (E == 0) next_state = s0;
           else if (F == 1) next_state = s1; else next_state = s3;

      s1:   if (E == 0) next_state = s1;
           else if (F == 1) next_state = s2; else next_state = s0;

      s2:   if (E == 0) next_state = s2;
           else if (F == 1) next_state = s3; else next_state = s1;

      s3:   if (E == 0) next_state = s3;
           else if (F == 1) next_state = s0; else next_state = s2;
    default: next_state = s0;
    endcase
  end
endmodule

module t_Prob_5_40 ();

  reg E, F, clk, reset_b;

  Prob_5_40 M0 ( E, F, clk, reset_b);

  initial #350$finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork
    #2 reset_b = 1;
    #3 reset_b = 0;    // Initialize to s0
  join

```

The timing diagram illustrates the operation of the 4-bit counter circuit. The signals shown are:

- clk**: A periodic clock signal.
- reset\_b**: An active-low reset signal that goes low at time 0.
- E**: A combinational output that is high when the state is 0, 1, 2, or 3.
- F**: A combinational output that is high when the state is 0, 1, 2, or 3.
- state[1:0]**: The 2-bit state output, showing the sequence of states: 0, 1, 2, 3, 0, 1, 2, 3, 2, 1, 0, 3, 2, 1.

```

elseif F = '1' then next_state <= s3; else next_state <= s1;
end if;

```

```

    when s3 => if (E = '0') then next_state <= s3;
                elsif (F = '1') then next_state <= s0; else next_state <= s2;
                end if;

    when others => next_state <= s0;
end case;
end process;
end Behavioral;

```

## 5.41 Verilog

```

module Prob_5_41 (output reg y_out, input x_in, clock, reset_b);
    parameter s0 = 3'b000, s1 = 3'b001, s2 = 3'b010, s3 = 3'b011, s4 = 3'b100;
    reg [2: 0] state, next_state;

    always @ (posedge clock, negedge reset_b)
        if (reset_b == 0) state <= s0;
        else state <= next_state;

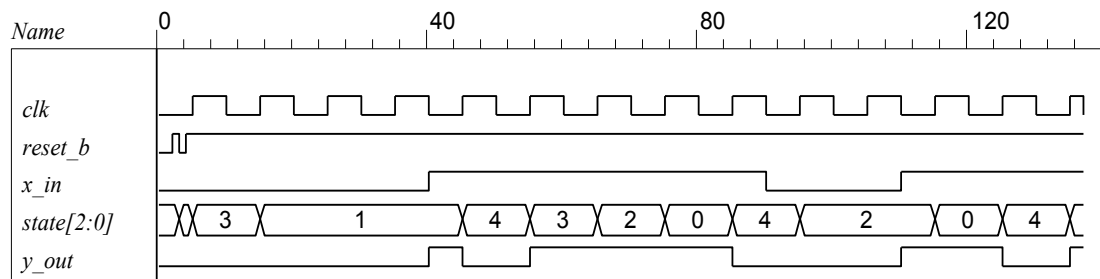
    always @ (state, x_in) begin
        y_out = 0;
        next_state = s0;
        case (state)
            s0:    if (x_in) begin next_state = s4; y_out = 1; end else begin next_state = s3; y_out = 0; end
            s1:    if (x_in) begin next_state = s4; y_out = 1; end else begin next_state = s1; y_out = 0; end
            s2:    if (x_in) begin next_state = s0; y_out = 1; end else begin next_state = s2; y_out = 0; end
            s3:    if (x_in) begin next_state = s2; y_out = 1; end else begin next_state = s1; y_out = 0; end
            s4:    if (x_in) begin next_state = s3; y_out = 0; end else begin next_state = s2; y_out = 0; end
            default: next_state = 3'bxxx;
        endcase
    end
endmodule

module t_Prob_5_41 ();
    wire y_out;
    reg x_in, clk, reset_b;

    Prob_5_41 M0 (y_out, x_in, clk, reset_b);

    initial #350$finish;
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial fork
        #2 reset_b = 1;
        #3 reset_b = 0;    // Initialize to s0
        #4 reset_b = 1;
        // Trace the state diagram and monitor y_out
        x_in = 0;    // Drive from s0 to s3 to S1 and park
        #40 x_in = 1;    // Drive to s4 to s3 to s2 to s0 to s4 and loop
        #90 x_in = 0;    // Drive from s0 to s3 to s2 and part
        #110 x_in = 1;    // Drive s0 to s4 etc
    join
endmodule

```



### VHDL

```

library IEEE;
use IEEE_Std_logic_1164.all
entity Prob_5_41_vhdl is
    port (y_out: out bit; x_in, clock, reset_b: in bit);
end Prob_5_41_vhdl;

architecture Behavioral of Prob_5_41_vhdl is
    constant s0: Std_Logic_vector (2 downto 0) := "000";
    constant s1: Std_Logic_vector (2 downto 0) := "001";
    constant s2: Std_Logic_vector (2 downto 0) := "010";
    constant s3: Std_Logic_vector (2 downto 0) := "011";
    constant s4: Std_Logic_vector (2 downto 0) := "100";
    signal state, next_state: Std_Logic_vector (2 downto 0);
begin
    process (clock, reset_b) begin
        if reset_b = '0' then state <= s0;
        elsif clock'event and clock = '1' then state <= next_state; end if;
    end process;

    process (state, x_in) begin
        y_out <= '0';
        next_state <= s0;
        case (state) is
            when s0 => if x_in = '1' then next_state <= s4; y_out <= '1';
                       else next_state <= s3; y_out <= '0'; end if;

            when s1 => if x_in = '1' then next_state <= s4; y_out <= '1';
                       else next_state <= s1; y_out <= '0'; end if;

            when s2 => if x_in = '1' then next_state <= s0; y_out <= '1';
                       else next_state <= s2; y_out <= '0'; end if;

            when s3 => if x_in = '1' then next_state <= s2; y_out <= '1';
                       else next_state <= s1; y_out <= '0'; end if;

            when s4 => if x_in = '1' then next_state <= s3; y_out <= '0';
                       else next_state <= s2; y_out <= '0'; end if;

            when others => next_state <= "xxx";
        end case;
    end process;
end Behavioral;
    
```

## 5.42

### Verilog

```
module Prob_5_42 (output A, B, B_bar, y, input x, clk, reset_b);
```

```
// See Fig. 5.29
```

```
  wire w1, w2, w3, D1, D2;
```

```
  and (w1, A, x);
```

```
  and (w2, B, x);
```

```
  or (D_A, w1, w2);
```

```
  and (w3, B_bar, x);
```

```
  and (y, A, B);
```

```
  or (D_B, w1, w3);
```

```
  DFF M0_A (A, D_A, clk, reset_b);
```

```
  DFF M0_B (B, D_B, clk, reset_b);
```

```
  not (B_bar, B);
```

```
endmodule
```

```
module DFF (output reg Q, input data, clk, reset_b);
```

```
  always @ (posedge clk, negedge reset_b)
```

```
  if (reset_b == 0) Q <= 0; else Q <= data;
```

```
endmodule
```

```
module t_Prob_5_42 ();
```

```
  wire A, B, B_bar, y;
```

```
  reg bit_in, clk, reset_b;
```

```
  wire [1:0] state;
```

```
  assign state = {A, B};
```

```
  wire detect = y;
```

```
  Prob_5_42 M0 (A, B, B_bar, y, bit_in, clk, reset_b);
```

```
// Patterns from Problem 5.45.
```

```
initial #350$finish;
```

```
initial begin clk = 0; forever #5 clk = ~clk; end
```

```
initial fork
```

```
  #2 reset_b = 1;
```

```
  #3 reset_b = 0;
```

```
  #4 reset_b = 1;
```

```
    // Trace the state diagram and monitor detect (assert in S3)
```

```
    bit_in = 0; // Park in S0
```

```
  #20 bit_in = 1; // Drive to S0
```

```
  #30 bit_in = 0; // Drive to S1 and back to S0 (2 clocks)
```

```
  #50 bit_in = 1;
```

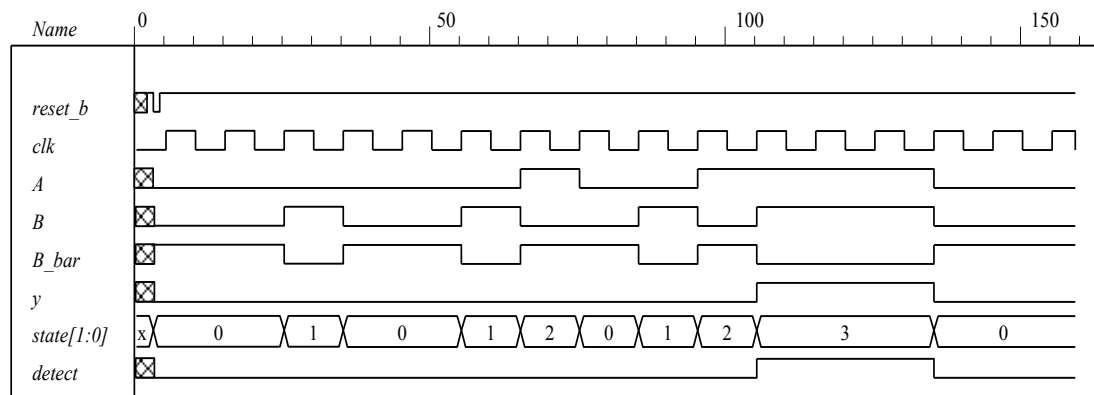
```
  #70 bit_in = 0; // Drive to S2 and back to S0 (3 clocks)
```

```
  #80 bit_in = 1;
```

```
  #130 bit_in = 0; // Drive to S3, park, then and back to S0
```

```
join
```

```
endmodule
```



## VHDL

**entity** Prob\_5\_42\_vhdl **is** -- See Fig. 5.29

**port** (A, B: **buffer bit**; A\_bar, B\_bar: **buffer bit**; y: **out bit**; xin, clk, reset\_b: **in bit**);

**end** Prob\_5\_42\_vhdl;

**architecture** Structural of Prob\_5\_42\_vhdl **is**

**signal** w1, w2, w3, D\_A, D\_B: **bit**;

**component** and2\_gate **port** (y: **out bit**; xin1, xin2: **in bit**); **end component**;

**component** or2\_gate **port** (y: **out bit**; xin1, xin2: **in bit**); **end component**;

**component** not\_gate **port** (y: **out bit**; xin: **in bit**); **end component**;

**component** D\_FF **port** (Q: **out bit**; data, clk, reset\_b: **in bit**); **end component**;

**begin**

G1: and2\_gate **port map** (y => w1, xin1 => A, xin2 => xin);

G2: and2\_gate **port map** (y => w2, xin1 => B, xin2 => xin);

G3: or2\_gate **port map** (y => D\_A, xin1 => w1, xin2 => w2);

G4: and2\_gate **port map** (y => w3, xin1 => B\_bar, xin2 => xin);

G5: and2\_gate **port map** (y => y, xin1 => A, xin2 => B);

G6: or2\_gate **port map** (y => D\_B, xin1 => w1, xin2 => w3);

G7: not\_gate **port map** (y => B\_bar, xin => B);

M0\_A: D\_FF **port map** (Q => A, data => D\_A, clk => clk, reset\_b => reset\_b);

M0\_B: D\_FF **port map** (Q => B, data => D\_B, clk => clk, reset\_b => reset\_b);

**end** Structural;

**entity** and2\_gate **is**

**port** (y: **out bit**; xin1, xin2: **in bit**);

**end** and2\_gate;

**architecture** Behavioral of and2\_gate **is**

**begin**

y <= xin1 **and** xin2;

**end** Behavioral;

**entity** or2\_gate **is**

**port** (y: **out bit**; xin1, xin2: **in bit**);

**end** or2\_gate;

**architecture** Behavioral of or2\_gate **is**

**begin**

y <= xin1 **or** xin2;

**end** Behavioral;

**entity** not\_gate **is**

**port** (y: **out bit**; xin: **in bit**);

**end** not\_gate;



```

architecture Behavioral of not_gate is
begin
    y <= not xin;
end Behavioral;

entity D_FF is
    port (Q: out bit; data, clk, reset_b: in bit);
end D_FF;

architecture Behavioral of D_FF is
begin
    process (clk, reset_b) begin
        if (reset_b = '0') then Q <= '0';
        elsif clk'event and clk = '1' then Q <= data;
        end if;
    end process;
end Behavioral;

```

### 5.43

#### Verilog

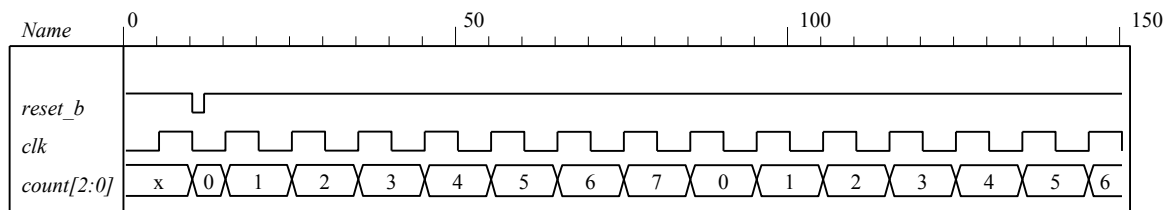
```

module Binary_Counter_3_bit (output [2: 0] count, input clk, reset_b)
    always @ (posedge clk) if (reset_b == 0) count <= 0; else count <= next_count;
    always @ (count) begin
        case (state)
            3'b000: count = 3'b001;
            3'b001: count = 3'b010;
            3'b010: count = 3'b011;
            3'b011: count = 3'b100;
            3'b100: count = 3'b001;
            3'b101: count = 3'b010;
            3'b110: count = 3'b011;
            3'b111: count = 3'b100;
            default: count = 3'b000;
        endcase
    end
endmodule

module t_Binary_Counter_3_bit ()
    wire [2: 0] count;
    reg clk, reset_b;
    Binary_Counter_3_bit M0 ( count, clk, reset_b)

    initial #150 $finish;
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial fork
        reset = 1;
        #10 reset = 0;
        #12 reset = 1;
    endmodule

```



Alternative: structural model.

```

module Prob_5_41 (output A2, A1, A0, input T, clk, reset_bar);

```

```

wire toggle_A2;

T_flop M0 (A0, T, clk, reset_bar);
T_flop M1 (A1, A0, clk, reset_bar);
T_flop M2 (A2, toggle_A2, clk, reset_bar);
and (toggle_A2, A0, A1);
endmodule

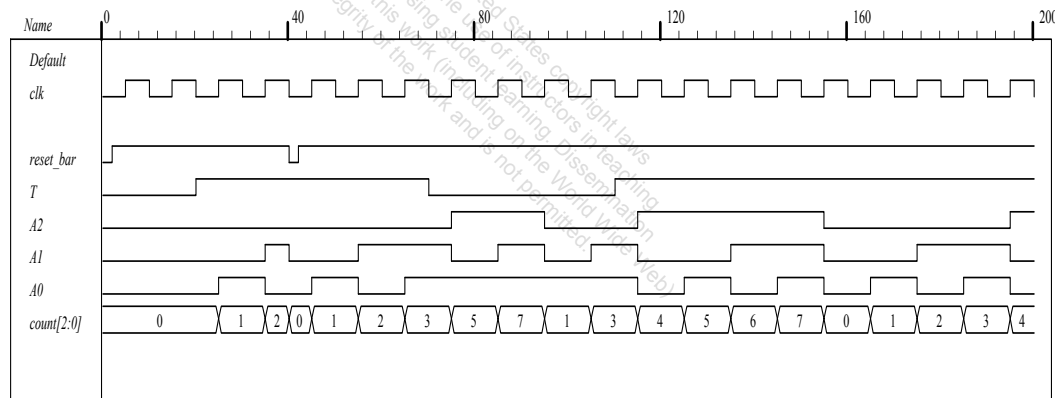
module T_flop (output reg Q, input T, clk, reset_bar);
  always @ (posedge clk, negedge reset_bar)
    if (!reset_bar) Q <= 0; else if (T) Q <= ~Q; else Q <= Q;
endmodule

module t_Prob_5_41;
  wire A2, A1, A0;
  wire [2: 0] count = {A2, A1, A0};
  reg T, clk, reset_bar;
  Prob_5_41 M0 (A2, A1, A0, T, clk, reset_bar);

  initial #200 $finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork reset_bar = 0; #2 reset_bar = 1; #40 reset_bar = 0; #42 reset_bar = 1; join
  initial fork T = 0; #20 T = 1; #70 T = 0; #110 T = 1; join
endmodule

```

If the input to *A0* is changed to 0 the counter counts incorrectly. It resumes a correct counting sequence when *T* is changed back to 1.



## VHDL

```

library IEEE;
use IEEE_Std_Logic_1164.all;

entity Prob_5_43_Binary_Counter_3_bit_vhdl is
  port (count: out bit_vector (2 downto 0; clk, reset_b: in bit);
end Prob_5_43_Binary_Counter_3_bit_vhdl;

architecture Behavioral of Prob_5_43_Binary_Counter_3_bit_vhdl is
begin
  process (clk) begin
    if reset_b = '0' then count <= "000";
    elsif clk'event and clk = '1' then count <= next_count; end if;
  end process;

```

```

process (count) begin
  case (state)
    when "000" => count <= "001";
    when "001" => count <= "010";
    when "010" => count <= "011";
    when "011" => count <= "100";
    when "100" => count <= "001";
    when "101" => count <= "010";
    when "110" => count <= "011";
    when "111" => count <= "100";
    when others => count <= "000";
  end case;
end process;
end Behavioral;

```

## 5.44

### VERILOG

```

module DFF_asynch_reset (output reg Q, input data, clk, reset);
  always @ (posedge clk, posedge reset) // Asynchronous reset
  if (reset) Q <= 0; else Q <= data;
endmodule

```

```

module t_DFF_asynch_reset ();
  reg data, clk, reset;
  wire Q;

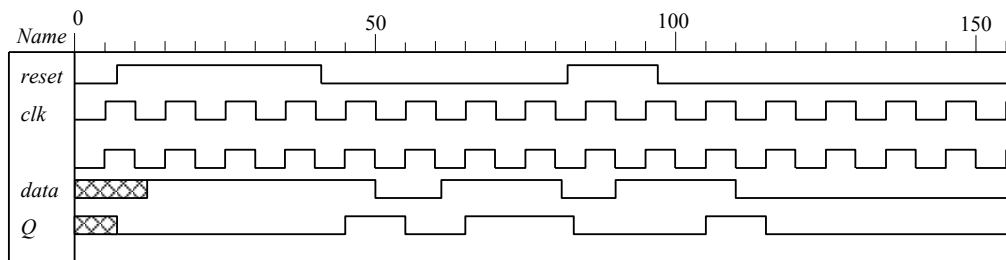
  DFF_asynch_reset M0 (Q, data, clk, reset);

```

```

initial #150 $finish;
initial begin clk = 0; forever #5 clk = ~clk; end
initial fork
  reset = 0;
  #7 reset = 1;
  #41 reset = 0;
  #82 reset = 1;
  #97 reset = 0;
  #12 data = 1;
  #50 data = 0;
  #60 data = 1;
  #80 data = 0;
  #90 data = 1;
  #110 data = 0;
join
endmodule

```



### VHDL

```

entity Prob_5_44_DFF_asynch_reset is
  port (Q: out bit; data, clk, reset: in bit);
end Prob_5_44_DFF_asynch_reset;

```

```

architecture Behavioral of Prob_5_44_DFF_asynch_reset is
begin

```

```

process (clk, reset)      -- Asynchronous reset
begin
    if reset = '1' then Q <= '0';
    elsif clk'event and clk = '1' then Q <= data; end if;
    end process;
end Behavioral;

```

## 5.45

### Verilog

```

module Prob_5_45_Seq_Detector (output detect, input bit_in, clk, reset_b);
    parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3;
    reg [1: 0] state, next_state;

```

```

    assign detect = (state == S3);
    always @ (posedge clk, negedge reset_b)
    if (reset_b == 0) state <= S0; else state <= next_state;

```

```

    always @ (state, bit_in) begin
        next_state = S0;
        case (state)
            0:    if (bit_in) next_state = S1; else state = S0;
            1:    if (bit_in) next_state = S2; else next_state = S0;
            2:    if (bit_in) next_state = S3; else state = S0;
            3:    if (bit_in) next_state = S3; else next_state = S0;
            default: next_state = S0;
        endcase
    end
endmodule

```

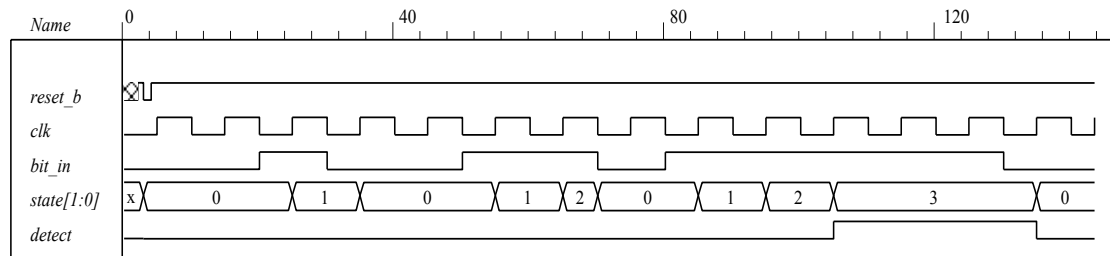
```

module t_Prob_5_45_Seq_Detector ();
    wire detect;
    reg bit_in, clk, reset_b;

    Prob_5_45_Seq_Detector M0 (detect, bit_in, clk, reset_b);

    initial #350$finish;
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial fork
        #2 reset_b = 1;
        #3 reset_b = 0;
        #4 reset_b = 1;
        // Trace the state diagram and monitor detect (assert in S3)
        bit_in = 0; // Park in S0
        #20 bit_in = 1; // Drive to S0
        #30 bit_in = 0; // Drive to S1 and back to S0 (2 clocks)
        #50 bit_in = 1;
        #70 bit_in = 0; // Drive to S2 and back to S0 (3 clocks)
        #80 bit_in = 1;
        #130 bit_in = 0; // Drive to S3, park, then and back to S0
    join
endmodule

```



## VHDL

```
library IEEE;
use IEEE_Std_Logic_1164.all;
```

```
entity Seq_Detector_vhdl is
```

```
    port (detect: out bit; bit_in, clk, reset_b: in bit);
```

```
end Prob_5_45_Seq_Detector_vhdl;
```

```
architecture Behavioral of Prob_5_45_Seq_Detector_vhdl is
```

```
    constant S0: Std_Logic_vector (1 downto 0) := "00";
```

```
    constant S1: Std_Logic_vector (1 downto 0) := "01";
```

```
    constant S2: Std_Logic_vector (1 downto 0) := "10";
```

```
    constant S3: Std_Logic_vector (1 downto 0) := "11";
```

```
    signal state, next_state: Std_Logic_vector (1 downto 0);
```

```
begin
```

```
    detect <= '1' when state = S3;
```

```
    process (clk, reset_b) begin
```

```
        if reset_b = '0' then state <= S0;
```

```
        elsif clk'event and clk = '1' then state <= next_state; end if;
```

```
    end process;
```

```
    process (state, bit_in) begin
```

```
        next_state <= S0;
```

```
        case (state) is
```

```
            when S0 => if bit_in = '1' then next_state <= S1; else state <= S0; end if;
```

```
            when S1 => if bit_in = '1' then next_state <= S2; else next_state <= S0; end if;
```

```
            when S2 => if bit_in = '1' then next_state <= S3; else state <= S0; end if;
```

```
            when S3 => if bit_in = '1' then next_state <= S3; else next_state <= S0; end if;
```

```
            when others => next_state <= S0;
```

```
        end case;
```

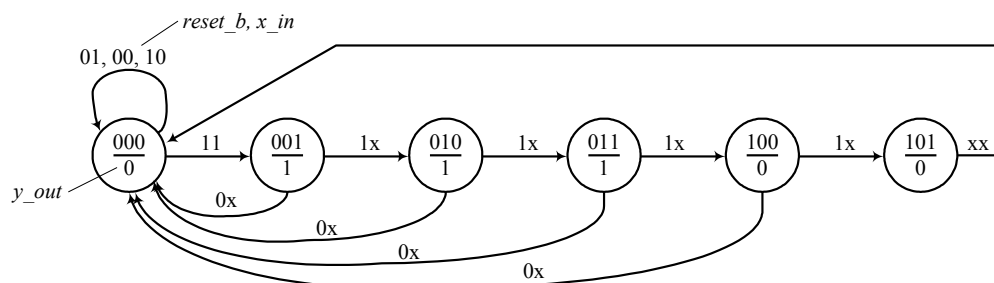
```
    end process;
```

```
end Behavioral;
```

## 5.46 Pending simulation results

Assumption: Synchronous active-low reset

Moore machine



Verify that machine remains in state 000 while reset\_b is asserted, independently of x\_in.  
 Verify that machine makes transition from 000 to 001 if not reset\_b and if x\_in is asserted.  
 Verify that state transitions from 000 through 101 are correct.  
 Verify reset\_b "on the fly."  
 Verify that y\_out is asserted correctly.

#### Verilog

```

module Prob_5_46 (output y_out, input x_in, clk, reset_b);
    reg [2:0] state, next_state;

    assign y_out = (state == 3'b001) || (state == 3'b010) || (state == 3'b011);
    always @ (posedge clk)
        if (reset_b == 1'b0) state <= 3'b000; else state <= next_state;

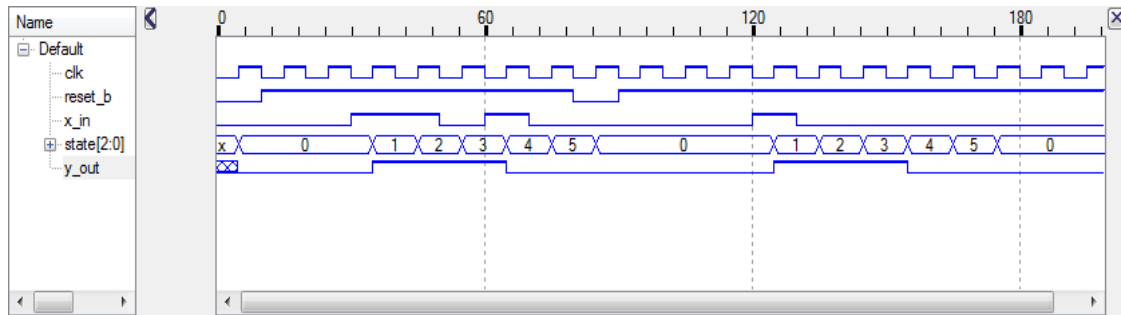
    always @ (x_in, state) begin
        next_state = 3'b000;
        case (state)
            3'b000: if (x_in) next_state = 3'b001; else next_state = 3'b000;
            3'b001: next_state = 3'b010;
            3'b010: next_state = 3'b011;
            3'b011: next_state = 3'b100;
            3'b100: next_state = 3'b101;
            3'b101: next_state = 3'b000;
            default: next_state = 3'b000;
        endcase
    end
endmodule

module t_Prob_5_46 ();
    reg x_in, clk, reset_b;
    wire y_out;

    Prob_5_46 M0 (y_out, x_in, clk, reset_b);

    initial #200 $finish;
    initial begin clk = 0; forever #5 clk = !clk; end
    initial fork
        reset_b = 0;
        #10 reset_b = 1;
        #80 reset_b = 0;
        #90 reset_b = 1;

        x_in = 0;
        #30 x_in = 1;
        #40 x_in = 1;
        #50 x_in = 0;
        #60 x_in = 1;
        #70 x_in = 0;
        #120 x_in = 1;
        #130 x_in = 0;
    join
endmodule
    
```



# **VHDL**

**entity** Prob\_5\_46 is

**port** (output y\_out, input x\_in, clk, reset\_b);

**end** Prob\_5\_46;

**architecture** Behavioral is

**signal** state, next\_state: bit\_vector (2 downto 0);

**begin**

y\_out <= '1' **when** (state = "001") **or** (state = "010") **or** (state = "011");

**process** (clk) **begin**

**if** reset\_b = '0' **then** state <= "000";

**else if** clk'event **and** clk = '1' **then** state <= next\_state; **end if**;

**end process**;

**process** (x\_in, state) **begin**

next\_state <= "000";

**case** (state) **is**

**when** 3'b000 => **if** x\_in = '1' **then** next\_state <= "001";

**else** next\_state = 3"000"; **end if**;

**when** "001" => next\_state <= "010";

**when** "010" => next\_state <= "011";

**when** "011" => next\_state <= "100";

**when** "100" => next\_state <= "101";

**when** "101" => next\_state <= "000";

**when others** => next\_state <= "000";

**end case**;

**end process**;

**end Behavioral**;

## 5.47

Assume synchronous active-low reset.

### Verilog

```
module Prob_5_47 (output reg [3:0] y_out, input Run, clk, reset_b);
  always @ (posedge clk)
    if (reset_b == 1'b0) y_out <= 4'b0000;
    else if (Run && (y_out < 4'b1110)) y_out <= y_out + 2'b10;
    else if (Run && (y_out == 4'b1110)) y_out <= 4'b0000;
    else y_out <= y_out; // redundant statement and may be omitted
endmodule
```

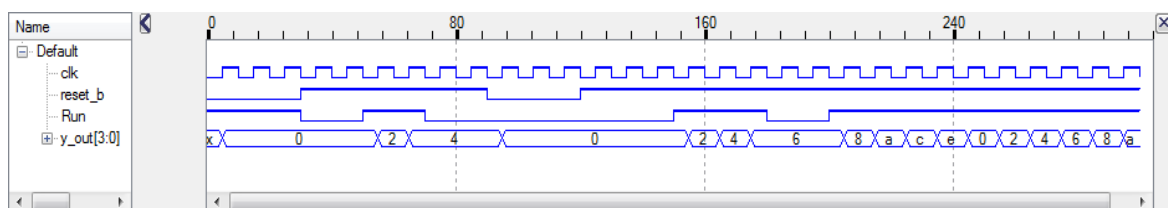
// Verify that counting is prevented while reset\_b is asserted, independently of Run  
 // Verify that counting is initiated by Run if reset\_b is de-asserted  
 // Verify reset on-the-fly  
 // Verify that deasserting Run suspends counting  
 // Verify wrap-around of counter.

```
module t_Prob_5_47 ();
  reg Run, clk, reset_b;
  wire [3:0] y_out;

  Prob_5_47 M0 (y_out, Run, clk, reset_b);

  initial #300 $finish;
  initial begin clk = 0; forever #5 clk = !clk; end
  initial fork
    reset_b = 0;
    #30 reset_b = 1;

    Run = 1; // Attempt to run is overridden by reset_b
    #30 Run = 0;
    #50 Run = 1; // Initiate counting
    #70 Run = 0; // Pause
    #90 reset_b = 0; // reset on-the-fly
    #120 reset_b = 1; // De-assert reset_b
    #150 Run = 1; // Resume counting
    #180 Run = 0; // Pause counting
    #200 Run = 1; // Resume counting
  join
endmodule
```





# VHDL

```
entity Prob_5_47_vhdl is
  port (y_out: out bit_vector (3 downto 0); input Run, clk, reset_b: in bit);
end Prob_5_47_vhdl;
```

**architecture** Behavioral **of** Prob\_5\_47\_vhdl **is**

**begin**

**process** (clk)

```
  if reset_b = '0' then y_out <= "0000";
  elsif clk'event and clk = '1' then
    if (Run = '1' and (y_out < "1110")) then y_out <= y_out + "10";
    elsif (Run = '1' and (y_out = "1110")) then y_out <= "0000";
    else y_out <= y_out;    // redundant statement and may be omitted
  end if;
```

**end** Behavioral;

// Verify that counting is prevented while reset\_b is asserted, independently of Run  
 // Verify that counting is initiated by Run if reset\_b is de-asserted  
 // Verify reset on-the-fly  
 // Verify that deasserting Run suspends counting  
 // Verify wrap-around of counter.

**entity** t\_Prob\_5\_47\_vhdl **is**

**port** ();

**end** t\_Prob\_5\_47\_vhdl;

**architecture** Behavioral **of** t\_Prob\_5\_47\_vhdl **is**

**signal** t\_Run, t\_clk, t\_reset\_b: bit;

**signal** y\_out: bit\_vector (3 **downto** 0);

**component** Prob\_5\_47 M0

**port map** (y\_out => t\_y\_out, Run => t\_Run, clk => \_clk, reset\_b => t\_reset\_b);

**end component**;

**begin**

G0: Prob\_5\_47\_vhdl

**port map** (y\_out => t\_y\_out, Run => t\_Run, clk => t\_clk, reset\_b => t\_reset\_b);

**process** – clock for testbench

**begin**

t\_clk <= '0';

**wait** for 5 ns;

t\_clk <= '1';

**wait** for 5 ns;

**end process**;

t\_reset\_b <= '0';

-- concurrent signal assignments

t\_reset\_b <= '1' **after** 30 ns;

t\_Run <= '1' **after** 30 ns;

// Attempt to run is overridden by reset\_b

t\_Run <= '0' **after** 30 ns;;

t\_Run <= '1'; **after** 50 ns;

// Initiate counting

t\_Run <= '0' **after** 70 ns;

// Pause

t\_reset\_b <= '0' **after** 90 ns;

// reset on-the-fly

t\_reset\_b <= '1' **after** 120 ns;

// De-assert reset\_b

t\_Run <= '1' **after** 150 ns;

// Resume counting

t\_Run <= '0' **after** 180 ns;

// Pause counting

t\_Run <= '1' **after** 200 ns;

// Resume counting

**end** Behavioral;

## 5.48

Assume "a" is the reset state.

Verilog

```

module Prob_5_48 (output reg y_out, input x_in, clk, reset_b);
  parameter s_a = 2'd0;
  parameter s_b = 2'd1;
  parameter s_c = 2'd2;
  parameter s_d = 2'd3;
  reg [1: 0] state, next_state;

  always @ (posedge clk)
    if (reset_b == 1'b0) state <= s_a;
    else state <= next_state;

  always @ (state, x_in) begin
    next_state = s_a;
    y_out = 0;
    case (state)
      s_a: if (x_in == 1'b0) begin next_state = s_b; y_out = 1; end
          else begin next_state = s_c; y_out = 0; end
      s_b: if (x_in == 1'b0) begin next_state = s_c; y_out = 0; end
          else begin next_state = s_d; y_out = 1; end
      s_c: if (x_in == 1'b0) begin next_state = s_b; y_out = 0; end
          else begin next_state = s_d; y_out = 1; end
      s_d: if (x_in == 1'b0) begin next_state = s_c; y_out = 1; end
          else begin next_state = s_a; y_out = 0; end
      default: begin next_state = s_a; y_out = 0; end
    endcase
  end
endmodule

```

Verify reset action.

Verify state transitions.

Transition to a; hold x\_in = 0 and get loop bc...

Transition to a; hold x\_in = 1 and get loop acda...

Transitions to b; hold x\_in = 1 and get loop bdacd...

Transition to d; hold x\_in = 0 and get loop dcdbc...

Confirm Mealy outputs at each state/input pair

Verify reset on-the-fly.

```

module t_Prob_5_48 ();
  reg x_in, clk, reset_b;
  wire y_out;

  Prob_5_48 M0 (y_out, x_in, clk, reset_b);

  initial #400 $finish;
  initial begin clk = 0; forever #5 clk = !clk; end
  initial fork
    reset_b = 0;
    #30 reset_b = 1;
    #30 x_in = 0;          // loop abcbcbc...

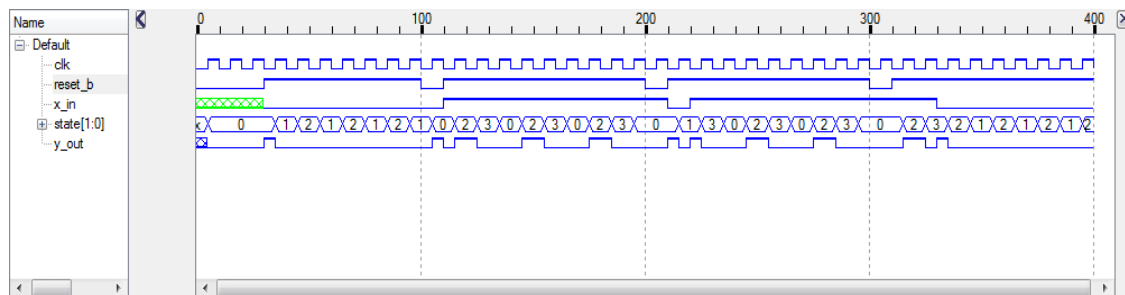
    #100 reset_b = 0;
    #110 reset_b = 1;
    #110 x_in = 1;         // loop acdacda...

    #200 reset_b = 0;

```

```
#210 reset_b = 1;
#210 x_in = 0;
#220 x_in = 1;      // loop bdacdacd...

#300 reset_b = 0;
#310 reset_b = 1;
#310 x_in = 1;
#330 x_in = 0;      // loop acdcbcbc....
join
endmodule
```



## 5.49

Assume "a" is the reset state.

### Verilog

```
module Prob_5_49 (output reg y_out, input x_in, clk, reset_b);
    parameter s_a = 2'd0;
    parameter s_b = 2'd1;
    parameter s_c = 2'd2;
    parameter s_d = 2'd3;
    reg [1:0] state, next_state;

    always @ (posedge clk)
        if (reset_b == 1'b0) state <= s_a;
        else state <= next_state;

    always @ (state, x_in) begin
        next_state = s_a;
        y_out = 1'b0;
        case (state)
            s_a: if (x_in == 1'b0) next_state = s_b;
                else next_state = s_c;
            s_b: begin y_out = 1'b1; if (x_in == 1'b0) next_state = s_c;
                else next_state = s_d; end
            s_c: begin y_out = 1'b1; if (x_in == 1'b0) next_state = s_b;
                else next_state = s_d; end
            s_d: if (x_in == 1'b0) next_state = s_c;
                else next_state = s_a;
            default: next_state = s_a;
        endcase
    end
endmodule
```

```
// Verify reset action.
// Verify state transitions.
// Transition to a; hold x_in = 0 and get loop abcbc...
```

```
// Transition to a; hold x_in = 1 and get loop acda...
// Transitions to b; hold x_in = 1 and get loop bdacd...
// Transition to d; hold x_in = 0 and get loop dcdbc...
// Confirm Moore outputs at each state
// Verify reset on-the-fly.
```

```
module t_Prob_5_49 ();
reg x_in, Run, clk, reset_b;
wire y_out;

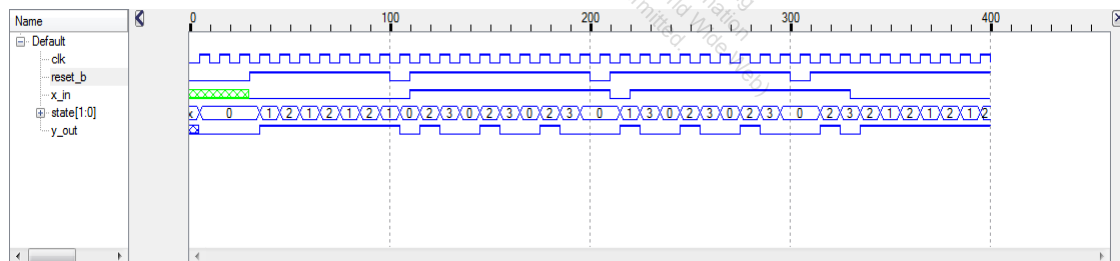
Prob_5_49 M0 (y_out, x_in, clk, reset_b);

initial #400 $finish;
initial begin clk = 0; forever #5 clk = !clk; end
initial fork
  reset_b = 0;
  #30 reset_b = 1;
  #30 x_in = 0;           // loop abcbcbcb...

  #100 reset_b = 0;
  #110 reset_b = 1;
  #110 x_in = 1;         // loop acdacda...

  #200 reset_b = 0;
  #210 reset_b = 1;
  #210 x_in = 0;
  #220 x_in = 1;         // loop bdacdacd...

  #300 reset_b = 0;
  #310 reset_b = 1;
  #310 x_in = 1;
  #330 x_in = 0;         // loop acdcbcbcb...
join
endmodule
```



## VHDL

```
entity Prob_5_49_vhdl is
  port (y_out: out bit; x_in, clk, reset_b: in bit);
end Prob_5_49_vhdl;
```

```
architecture Behavioral of Prob_5_49_vhdl is
  constant s_a: bit_vector (1 downto 0) := "00";
  constant s_b: bit_vector (1 downto 0) := "01";
  constant s_c: bit_vector (1 downto 0) := "10";
  constant s_d: bit_vector (1 downto 0) := "11";
  signal state, next_state: bit_vector (1 downto 0);
begin
  process (clk) begin
    if (reset_b = '0') then state <= s_a;
```

```

    elsif clk'event and clk = '1' then state <= next_state; end if;
end process;

process (state, x_in) begin
    next_state <= s_a;
    y_out <= '0';
    case (state) is
        when s_a => if x_in = '0' then next_state <= s_b;
                     else next_state <= s_c; end if;

        when s_b => y_out <= '1'; if x_in = '0' then next_state <= s_c;
                     else next_state <= s_d; end if;

        when s_c => y_out <= '1'; if x_in = '0' then next_state <= s_b;
                     else next_state <= s_d; end if;

        when s_d => if x_in = '0' then next_state <= s_c;
                     else next_state <= s_a; end if;

        when others => next_state <= s_a;
    end case;
end process;
end Behavioral;

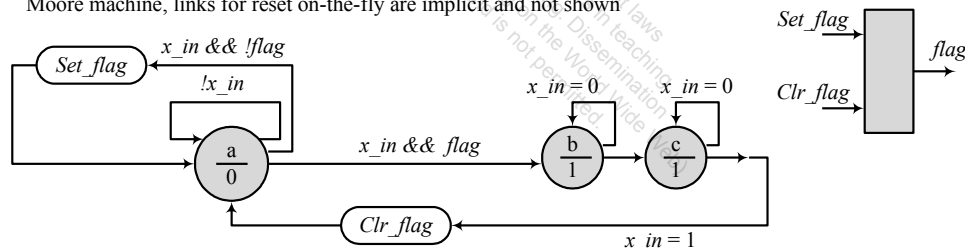
```

### 5.50

The machine is to remain in its initial state until a second sample of the input is detected to be 1. A flag will be set when the first sample is obtained. This will enable the machine to detect the presence of the second sample while being in the initial state. The machine is to assert its output upon detection of the second sample and to continue asserting the output until the fourth sample is detected.

Assumption: Synchronous active-low reset

Moore machine, links for reset on-the-fly are implicit and not shown



Note: the output signal  $y\_out$  is a Moore-type output. The control signals  $Set\_flag$  and  $Clr\_flag$  are not.

### Verilog

```

module Prob_5_50 (output y_out, input x_in, clk, reset_b);
    parameter s_a = 2'd0;
    parameter s_b = 2'd1;
    parameter s_c = 2'd2;

    reg Set_flag;
    reg Clr_flag;
    reg [1:0] state, next_state;
    assign y_out = (state == s_b) || (state == s_c);
    always @ (posedge clk)
        if (reset_b == 1'b0) state <= s_a;
        else state <= next_state;

```

```

always @ (state, x_in, flag) begin
    next_state = s_a;
    Set_flag = 0; // Assign by exception
    Clr_flag = 0;
    case (state)
        s_a: if ((x_in == 1'b1) && (flag == 1'b0))
            begin next_state = s_a; Set_flag = 1; end
            else if ((x_in == 1'b1) && (flag == 1'b1))
            begin next_state = s_b; Set_flag = 0; end
            else if (x_in == 1'b0) next_state = s_a;
        s_b: if (x_in == 1'b0) next_state = s_b;
            else begin next_state = s_c; Clr_flag = 1; end
        s_c: if (x_in == 1'b0) next_state = s_c;
            else next_state = s_a;
        default: begin next_state = s_a; Clr_flag = 1'b1; end
    endcase
end

always @ (posedge clk)
    if (reset_b == 1'b0) flag <= 0;
    else if (Set_flag) flag <= 1'b1;
    else if (Clr_flag) flag <= 1'b0;
endmodule

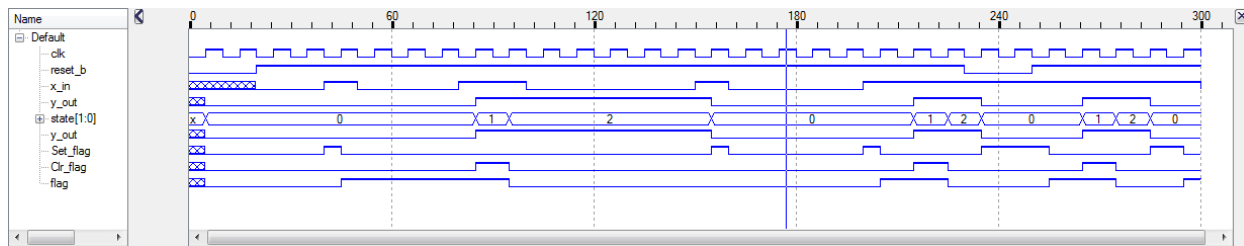
// Verify reset action
// Verify detection of first input
// Verify wait for second input
// Verify transition at detection of second input
// Verify with between detection of input
// Verify transition to s_d at fourth detection of input
// Verify return to s_a and clearing of flag after fourth input
// Verify reset on-the-fly

module t_Prob_5_50 ();
    wire y_out;
    reg x_in, clk, reset_b;

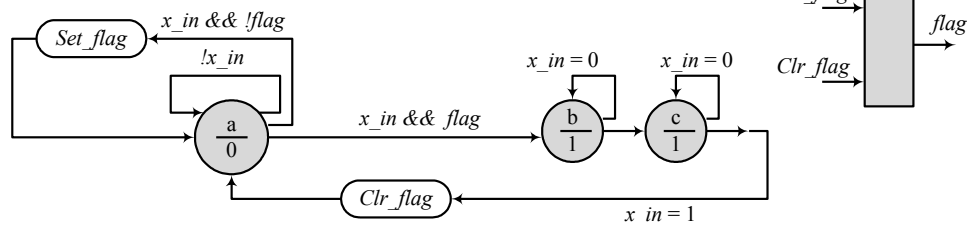
    Prob_5_50 M0 (y_out, x_in, clk, reset_b);

    initial #500 $finish;
    initial begin clk = 0; forever #5 clk = !clk; end
    initial fork
        reset_b = 1'b0;
        #20 reset_b = 1;
        #20 x_in = 1'b0;
        #40 x_in = 1'b1;
        #50 x_in = 1'b0;
        #80 x_in = 1'b1;
        #100 x_in = 0;
        #150 x_in = 1'b1;
        #160 x_in = 1'b0;
        #200 x_in = 1'b1;
        #230 reset_b = 1'b0;
        #250 reset_b = 1'b1;
        #300 x_in = 1'b0;
    join
endmodule

```



Assumption: Synchronous active-low reset  
Moore machine, links for reset on-the-fly are implicit and not shown



### VHDL

**entity** Prob\_5\_50\_vhdl is

**port** (y\_out: **out bit**; x\_in, clk, reset\_b: **in bit**);

**end** Prob\_5\_50\_vhdl;

**architecture** Behavioral of Prob\_5\_50\_vhdl is

**constant** s\_a: bit\_vector := "00";

**constant** s\_b: bit\_vector := "01";

**constant** s\_c: bit\_vector := "10";

**signal** Set\_flag: **bit**;

**signal** Clr\_flag: **bit**;

**signal** state, next\_state: bit\_vector (1 **downto** 0);

**begin**

y\_out <= '1' **when** state = s\_b **or** state = s\_c;

**process** (clk) **begin**

**if** (reset\_b = '0') **then** state <= s\_a;

**else** state <= next\_state; **end if**;

**end process**;

**process** (state, x\_in, flag) **begin**

next\_state <= s\_a;

Set\_flag <= 0;

Clr\_flag <= 0;

**case** (state) **is**

**when** s\_a => **if** x\_in = '1' **and** flag = '0'

**then** next\_state <= s\_a; Set\_flag <= '1';

**elsif** x\_in = '1' **and** flag = '1'

**then** next\_state <= s\_b; Set\_flag <= 0;

**elsif** x\_in = '0' **then** next\_state <= s\_a;

**end if**;

**when** s\_b => **if** x\_in = '0' **then** next\_state <= s\_b;

**else** next\_state <= s\_c; Clr\_flag <= '1';

**end if**;

**when** s\_c => **if** x\_in = '0' **then** next\_state <= s\_c;

**else** Clr\_flag <= '1'; next\_state <= s\_a; **end if**;

**end case**;

```

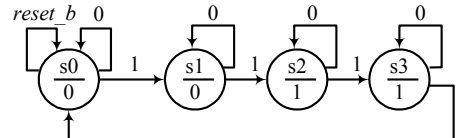
end process;

process (clk) begin
  if reset_b = '0' then flag <= '0';
  elsif Set_flag = '1' then flag <= '1';
  elsif Clr_flag='1' then flag <= '0';
  end if;
end process;
end Behavioral;

```

5.51

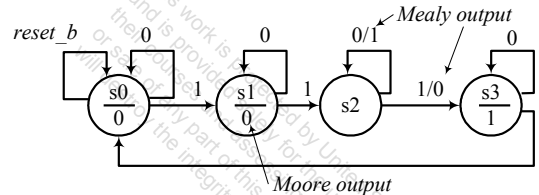
Assumption: Synchronous active-low reset  
Moore machine, links for reset on-the-fly are implicit and not shown



Note: the machine is a Moore machine.

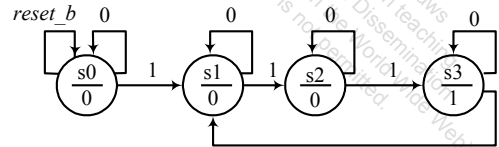
5.52

Assumption: Synchronous active-low reset  
Moore/Mealy machine, links for reset on-the-fly are implicit and not shown



5.53

Assumption: Synchronous active-low reset  
Moore machine, links for reset on-the-fly are implicit and not shown



Verilog

```

module Prob_5_53 (output reg y, input clk, reset_b, x_in);
  parameter s0 = 2'd0;
  parameter s1 = 2'd1;
  parameter s2 = 2'd2;
  parameter s3 = 2'd3;
  reg [1:0] state, next_state;

  always @ (posedge clk, negedge reset_b)
    if (reset_b == 1'b0) then state <= s0;
    else state <= next_state;

  always @ (state, xin) begin
    y = 0;
    next_state = s0;
    case (state)
      s0: begin y = 1'b0; if (xin) next_state = s1; else next_state = s0; end
      s1: begin y = 1'b0; if (xin) next_state = s2; else next_state = s1; end
      s2: begin y = 1'b0; if (xin) next_state = s3; else next_state = s2; end
      s3: begin y = 1'b1; if (xin) next_state = s1; else next_state = s3; end
    endcase
  end

```

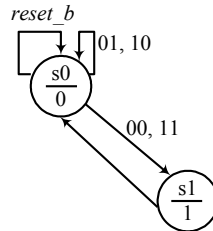


```

        default: begin y = 1'b0; next_state = s0; end
    endcase
endmodule

```

5.54



### Verilog

```

module Prob_5_54 (input xin1, xin2, clk, reset_b, output reg y_out);
    parameter s0 = 1'b0;
    parameter s1 = 1'b1;
    reg [1: 0] state, next_state;

    always @ (posedge clk, negedge reset_b) // state transitions
        if (reset_b == 1'b0) state <= s0;
        else state <= next_state;

    always @ (state, xin1, xin2) begin // next state
        next_state = s0; ;
        case (state)
            s0: if (xin1 == xin2) next_state = s1; else next_state = s0;
            s1: next_state = s0;
            default: next_state = s0;
        endcase
    end

    always @ (state, xin1, xin2) begin //output
        y_out = 1'b0;
        case (state)
            s0: y_out = 1'b0;
            s1: y_out = 1'b1;
            default: y_out = 1'b0;
        endcase
    end
endmodule

```

### VHDL

```

entity Prob_5_54_vhdl is
    port (xin1, xin2, clk, reset_b: in bit; y_out: out bit);
end Prob_5_54_vhdl;

architecture Behavioral of Prob_5_54_vhdl is
    constant s0: bit := '0';
    constant s1: bit := '1';
    signal state, next_state: bit_vector (1 downto 0);
begin
    process (clk, reset_b) begin // state transitions
        if (reset_b = '0') then state <= s0;
        elsif clk'event and clk = '1' then state <= next_state; end if;
    end process;

```

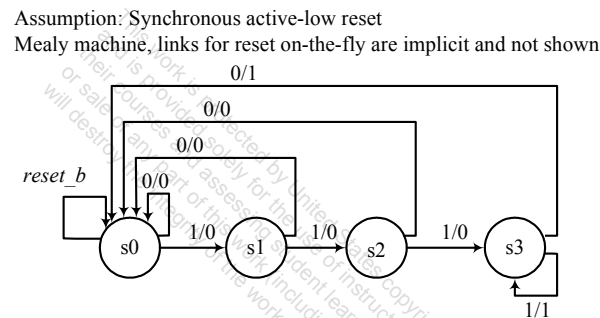
```

process (state, xin1, xin2) begin           // next state
    next_state <= s0; ;
    case (state) is
        s0: if (xin1 = xin2) then next_state <= s1; else next_state <= s0;
        s1: next_state <= s0;
        default: next_state <= s0;
    end case;
end process;

process (state, xin1, xin2) begin           //output
    y_out <= '1'b0;
    case (state)
        when s0: y_out <= '0';
        when s1: y_out <= '0';
        when others: y_out <= '0';
    end case;
end process;
end Behavioral;

```

## 5.55



## Verilog

```

module Prob_5_55 (input x_in, clk, reset_b, output reg y);
    parameter s0 = 1'd0;
    parameter s1 = 1'd1;
    parameter s2 = 1'd2;
    parameter s3 = 1'd3;
    reg [1:0] state, next_state;

    always @ (posedge clk, negedge reset_b)
        if (reset_b == 1'b0) state <= s0;
        else state <= next_state;

    always @ (state, x_in) begin
        y = 1'b0;
        next_state = s0;

        case (state)           // Mealy machine
            s0: if (x_in) begin y = 1'b0; next_state = s1; end
                else begin y = 1'b0; next_state = s0; end
            s1: if (x_in) begin y = 1'b0; next_state = s2; end
                else begin y = 1'b0; next_state = s0; end
            s2: if (x_in) begin y = 1'b0; next_state = s3; end
                else begin y = 1'b0; next_state = s0; end
            s3: if (x_in) begin y = 1'b1; next_state = s3; end
                else begin y = 1'b0; next_state = s0; end
            default: begin y = 1'b0; next_state = s0; end
        endcase
    end

```

```

    end
endmodule

```

## VHDL

```

entity Prob_5_55_vhdl is
    port (x_in, clk, reset_b: out bit; y: out bit);
end Prob_5_55_vhdl;

architecture Behavioral of Prob_5_55_vhdl is
    constant s0: bit_vector (1 downto 0) := "00";
    constant s1: bit_vector (1 downto 0) := "01";
    constant s2: bit_vector (1 downto 0) := "10";
    constant s3: bit_vector (1 downto 0) := "11";
    signal state, next_state: bit_vector (1 downto 0);
begin
    process (clk, reset_b)
        if (reset_b = '0') then state <= s0; end if;
        else if clk'event and clk = '1' then state <= next_state; end if;

    process ( clk, reset_b) begin
        y <= '0';
        next_state <= s0;

        case (state)
            when s0:    y <= '1'b0; if (x_in = '1') then next_state <= s1; else next_state <= s0; end if;

            when s1:    y <= '1'b0; if (x_in = '1') then next_state <= s2; else next_state <= s1; end if;

            when s2:    y <= '1'b0; if (x_in = '1') then next_state <= s3; else next_state <= s2; end if;

            when s3:    y <= '1'b1; if (x_in = '1') then next_state <= s1; else next_state <= s3; end if;

            when others: y <= '1'b0; next_state <= s0;
        endcase
    end process;
end Behavioral;

```

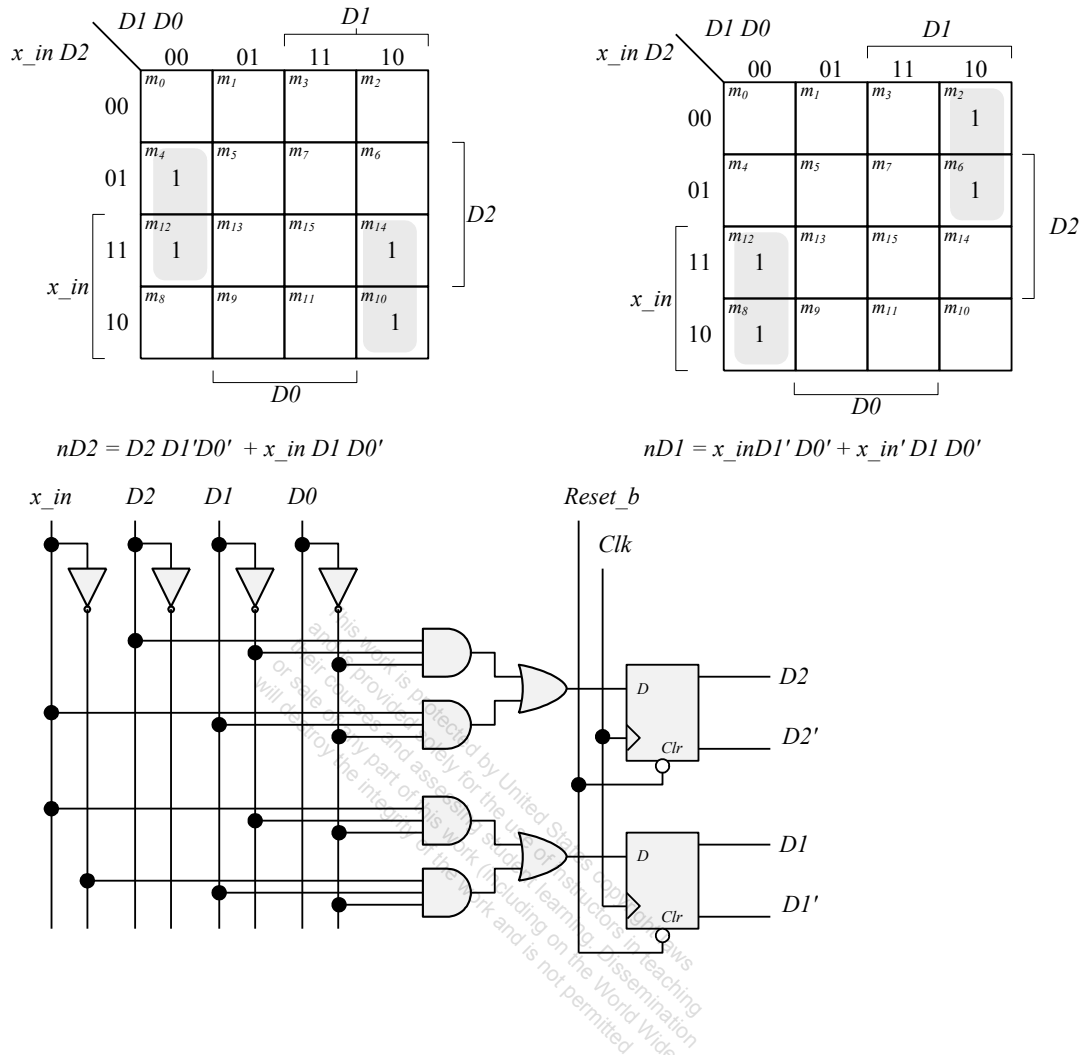
## 5.56

reset_b	x_in	D2	D1	D0	nD2	nD1	nD0
0	x	0	0	0	0	0	0
0	x	0	0	1	0	0	0
0	x	0	1	0	0	0	0
0	x	0	1	1	0	0	0
0	x	1	0	0	0	0	0
0	x	1	0	1	0	0	0
0	x	1	1	0	0	0	0
0	x	1	1	1	0	0	0
1	0	0	0	0	0	0	0
1	1	0	0	0	0	1	0
1	x	0	0	1	0	0	0
1	0	0	1	0	0	1	0
1	1	0	1	0	1	0	0
1	x	0	1	1	0	0	0
1	0	1	0	0	1	0	0
1	1	1	0	0	1	1	0
1	x	1	0	1	0	0	0
1	0	1	1	0	1	1	0
1	1	1	1	0	0	0	0
1	x	1	1	1	0	0	0

For reset\_b = 1:

$$nD2 = (x\_in \ D2'D1D0') \parallel (x\_in' \ D2 \ D1' \ D0') \parallel (x\_in \ D2 \ D1' \ D0') \parallel (x\_in \ D2 \ D1 \ D0')$$

$$nD1 = (x\_in \ D2' \ D1' \ D0') \parallel (x\_in' \ D2' \ D1 \ D0') \parallel (x\_in \ D2 \ D1' \ D0') \parallel (x\_in' \ D2 \ D1 \ D0')$$



## 5.57

Assume synchronous active-low reset. Assume that the counter is controlled by assertion of *Run*.

**Verilog** (Simple version below increments the count by 2, but fails if count is ever not 0, 2, 4, or 6)

```
module Prob_5_57 (output reg [2:0] y_out, input Run, clk, reset_b);
    always @ (posedge clk)
        if (reset_b == 1'b0) y_out <= 3'b000;
        else if (Run && (y_out < 3'b110)) y_out <= y_out + 3'b010;
        else if (Run && (y_out == 3'b110)) y_out <= 3'b000;
        else y_out <= y_out; // redundant statement and may be omitted
endmodule
```

Safer alternative:

```
module Prob_5_57_safe (output reg [2:0] y_out, input Run, clk, reset_b);
    reg [2:0] next_y_out;

    always @ (posedge clk)
        if (reset_b == 1'b0) y_out <= 3'b000; else y_out <= next_y_out;

    always @ (y_out, Run) begin
        y_out = 3'd0; // assign by exception
        case (y_out)

```

```

3'd0:  if (Run) next_y_out = 3'd2;
3'd1:  next_y_out = 3'd0;
3'd2:  if (Run) next_y_out = 3'd4;
3'd3:  next_y_out = 3'd0;
3'd4:  if (Run) next_y_out = 3'd6;
3'd5:  next_y_out = 3'd0;
3'd6:  if (Run) next_count = 3'd0;
3'd7:  next_y_out = 3'd0; // Can be omitted – covered by default
default: if (Run) next_y_out = 3'd0;
    endcase;
    end;
endmodule

// Verify that counting is prevented while reset_b is asserted, independently of Run
// Verify that counting is initiated by Run if reset_b is de-asserted
// Verify reset on-the-fly
// Verify that deasserting Run suspends counting
// Verify wrap-around of counter.

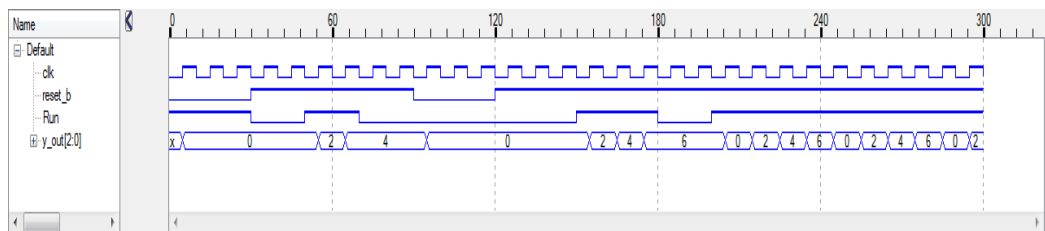
module t_Prob_5_57 ();
    reg Run, clk, reset_b;
    wire [2:0] y_out;

    Prob_5_57 M0 (y_out, Run, clk, reset_b);

    initial #300 $finish;
    initial begin clk = 0; forever #5 clk = !clk; end
    initial fork
        reset_b = 0;
        #30 reset_b = 1;

        Run = 1;          // Attempt to run is overridden by reset_b
        #30 Run = 0;
        #50 Run = 1;      // Initiate counting
        #70 Run = 0;      // Pause
        #90 reset_b = 0;  // reset on-the-fly
        #120 reset_b = 1; // De-assert reset_b
        #150 Run = 1;     // Resume counting
        #180 Run = 0;     // Pause counting
        #200 Run = 1;     // Resume counting
    join
endmodule

```



## VHDL

```

entity Prob_5_57_vhdl is
    port (y_out: buffer bit_vector (2 downto 0); Run, clk, reset_b: in bit);
end Prob_5_57_vhdl;

```

```

architecture Behavioral of Prob_5_57_vhdl is
    signal next_y_out: bit_vector (2 downto 0);

```

```

begin
  process (clk) begin
    if clk'event and clk = '1' and (reset_b = '0') then y_out <= "000";    -- Priority: synch reset
    elsif clk'event and clk = '1' and Run = '1' then y_out <= next_y_out; end if;
  end process;

  process (y_out) begin
    if clk'event and clk = '1' and Run = '1' then
      next_y_out = "000"; -- assign by exception
      case (y_out)
        when "000" => next_y_out <= "010";    -- 2
        when "001" => next_y_out <= "000";
        when "010" => next_y_out <= "100";    -- 4
        when "011" => next_y_out <= "000";
        when "100" => next_y_out <= "110";    -- 6
        when "101" => next_y_out <= "000";
        when "110" => next_y_out <= "000";
        when "111" => next_y_out <= "000"; -- Can be omitted – covered by default
        when others => next_y_out <= "000";
      endcase;
    end process;
  end Behavioral;

```

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted.

## 5.58

### Verilog

```

module Prob_5_58 (output reg y_out, input x_in, clk, reset_b)
  parameter s0 = 2'b00;
  parameter s1 = 2'b01;
  parameter s2 = 2'b10;
  parameter s3 = 2'b11;
  reg [1:0] state, next_state;

  always @ (posedge clk, negedge reset_b)
    if (reset_b == 1'b0) state <= s0;
    else state <= next_state;

  always @ (state, x_in) begin
    y_out = 0;
    next_state = s0;
    case(state)
      s0: if (x_in == 1'b0) next_state = s0; else if (x_in == 1'b1) next_state = s1;
      s1: if (x_in == 1'b0) next_state = s0; else if (x_in == 1'b1) next_state = s2;
      s2: if (x_in == 1'b0) next_state = s0; else if (x_in == 1'b1) next_state = s3;
      s3: begin y_out = 1; if (x_in == 1'b0) next_state = s0;
           else if (x_in == 1'b1) next_state = s3; end;
      default: begin next_state = s0; y_out = 0; end
    endcase;
  end
endmodule

```

```

module t_Prob_5_58 ();
  wire y_out;
  reg x_in, clk, reset_b;

  Prob_5_58 M0 (y_out, x_in, clk, reset_b)

  initial begin clk = 0; forever #5 clk = !clk; end
  initial fork
    reset_b = 0;
    x_in = 0;
    #20 reset_b = 1;
    #40 reset_b = 1;
    #50 x_in = 1;
    #60 x_in = 0;
    #80 x_in = 1;
    #90 x_in = 0;
    #110 x_in = 1;
    #120 x_in = 1;
    #150 x_in = 0;
    #200 x_in = 1;
    #210 reset_b = 0;
    #240 reset_b = 1;
  join
endmodule

```

### VHDL

```

entity Prob_5_58_vhdl is
  port (y_out: out bit; x_in, clk, reset_b: in bit);
end Prob_5_58_vhdl;

architecture Behavioral of Prob_5_58_vhdl is
  constant s0: bit_vector (1 downto 0) := "00";
  constant s1: bit_vector (1 downto 0) := "01";
  constant s2: bit_vector (1 downto 0) := "10";
  constant s3: bit_vector (1 downto 0) := "11";

```



```

    signal state, next_state: bit_vector (1 downto 0);
begin

process (clk, reset_b) begin
    if (reset_b = '0') then state <= s0; -- Asynchronous, active-low reset
    elsif clk'event and clk = '1' then state <= next_state;
end process;

process (state, x_in) begin
    y_out <= '0';
    next_state <= s0;
    case(state) is
        when s0 => if (x_in = '0') then next_state <= s0;
                    elsif (x_in = '1') then next_state = s1; end if;

        when s1 => if (x_in = '0') then next_state <= s0;
                    elsif (x_in = '1') then next_state = s2; end if;

        when s2 => if (x_in = '1') then next_state <= s0;
                    elsif (x_in = '1') then next_state <= s3;

        when s3 => y_out <= 1; if (x_in = '0') then next_state <= s0;
                    elsif (x_in = '1') then next_state <= s3;

        when others => next_state <= s0; y_out <= 0;
    endcase
end process;
end Behavioral;

```

## 5.59

### Verilog

```

module Prob_5_59 (output reg [2:0] count, input enable, clk, reset_b);
    always @ (posedge clk)
        if (reset_b == 1'b0) count <= 3'b000;
        else if (enable) case (count)
            3'b000: count <= 3'b010; // 2
            3'b010: count <= 3'b100; // 4
            3'b100: count <= 3'b110; // 6
            3'b110: count <= 3'b000;
            default: count <= 3'b111; // Use for error detection
        endcase
endmodule

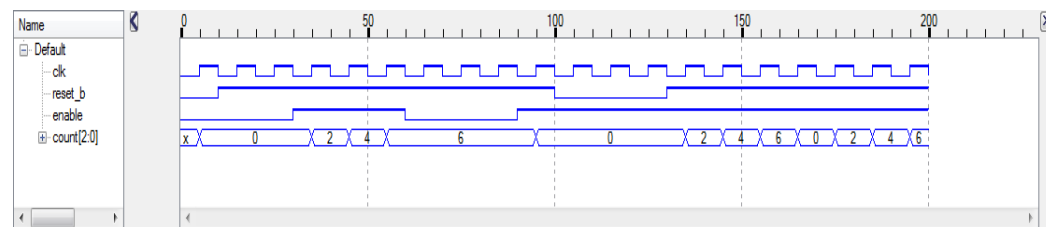
module t_Prob_5_59 ();
    wire [2:0] count;
    reg enable, clk, reset_b;

    Prob_5_59 M0 (count, enable, clk, reset_b);

    initial #200 $finish;
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial fork
        reset_b = 0;
        #10 reset_b = 1;
        #100 reset_b = 0;
        #130 reset_b = 1;
        enable = 0;
        #30 enable = 1;
        #60 enable = 0;
        #90 enable = 1;
    join

```

**endmodule**



### VHDL

```
entity Prob_5_59_vhdl is
    port (count: out bit_vector (2 downto 0); enable, clk, reset_b: in bit);
end Prob_5_59_vhdl;

architecture Behavioral of Prob_5_59_vhdl is
begin
    process (clk)
        if clk'event and clk = '1' then if (reset_b = '0') then count <= "000";
        elsif (enable = '1') then
            count <= '0';
            case (count) is
                when "000" => count <= "010";
                when "001" => count <= "100";
                when "100" => count <= "110";
                when "110" => count <= "000";
                when others => count <= "000"; // Use for error detection
            end case;
        end process;
    end Behavioral;
```

## 5.60

Assume synchronous active-low reset. Assume that counting is controlled by Run.

### Verilog

```
module Prob_5_60 (output reg [3:0] y_out, input Run, clk, reset_b);
    always @ (posedge clk)
        if (reset_b == 1'b0) y_out <= 4'b0000;
        else if (Run && (y_out < 4'b1001)) y_out <= y_out + 4'b0001;
        else if (Run && (y_out == 4'b1001)) y_out <= 4'b0000;
        else y_out <= y_out; // redundant statement and may be omitted
endmodule
```

```
// Verify that counting is prevented while reset_b is asserted, independently of
Run
// Verify that counting is initiated by Run if reset_b is de-asserted
// Verify reset on-the-fly
// Verify that deasserting Run suspends counting
// Verify wrap-around of counter.
```

```
module t_Prob_5_60 ();
    reg Run, clk, reset_b;
    wire [3:0] y_out;

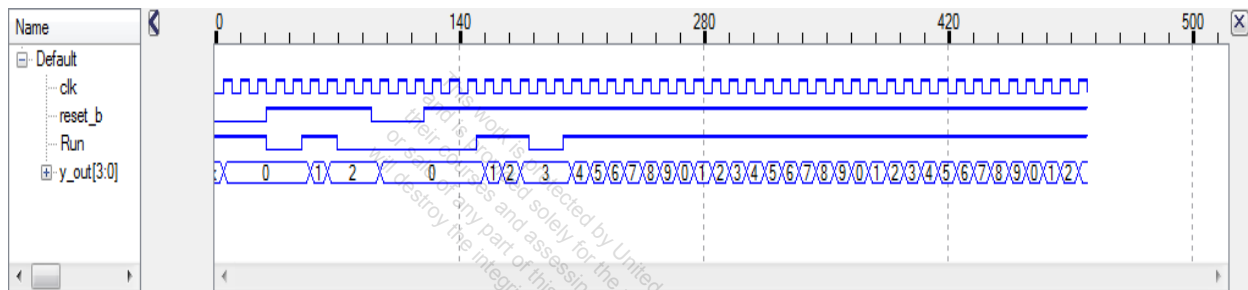
    Prob_5_60 M0 (y_out, Run, clk, reset_b);
```

```

initial #500 $finish;
initial begin clk = 0; forever #5 clk = !clk; end
initial fork
  reset_b = 0;
  #30 reset_b = 1;

  Run = 1;          // Attempt to run is overridden by reset_b
  #30 Run = 0;
  #50 Run = 1;      // Initiate counting
  #70 Run = 0;      // Pause
  #90 reset_b = 0;  // reset on-the-fly
  #120 reset_b = 1; // De-assert reset_b
  #150 Run = 1;     // Resume counting
  #180 Run = 0;     // Pause counting
  #200 Run = 1;     // Resume counting
join
endmodule

```



# VHDL

```

entity Prob_5_60_vhdl is
  port (y_out: buffer bit_vector (3 downto 0); Run, clk, reset_b: in bit);
end Prob_5_60_vhdl;

```

```

architecture Behavioral of Prob_5_60_vhdl is
begin
  process (clk) begin

```

```

    if clk'event and clk = '1' then
      if (reset_b = '0') then y_out <= "0000";
      elsif (Run = '1') then
        case (y_out) is
          when "0000" => y_out <= "0001";
          when "0001" => y_out <= "0010";
          when "0010" => y_out <= "0011";
          when "0011" => y_out <= "0100";
          when "0100" => y_out <= "0101";
          when "0101" => y_out <= "0110";
          when "0110" => y_out <= "0111";
          when "0111" => y_out <= "1000";
          when "1000" => y_out <= "1001";
          when "1001" => y_out <= "0000";
          when others => y_out <= "0001";
        end case;

```

```

      end if;
    end process;
end Behavioral;

```